django-storages Documentation

Release 1.7.2

David Larlet, et. al.

Sep 10, 2019

Contents

django-storages is a collection of custom storage backends for Django.

CHAPTER 1

Amazon S3

1.1 Usage

There is only one supported backend for interacting with Amazon's S3, `S3Boto3Storage`, based on the boto3 library. The backend based on the boto library has now been officially deprecated and is due to be removed shortly.

All current users of the legacy `S3BotoStorage` backend are encouraged to migrate to the `S3Boto3Storage` backend by following the *migration instructions*.

1.1.1 Settings

To upload your media files to S3 set:

```
DEFAULT_FILE_STORAGE = 'storages.backends.s3boto3.S3Boto3Storage'
```

To allow `django-admin.py` collectstatic to automatically put your static files in your bucket set the following in your settings.py:

```
STATICFILES_STORAGE = 'storages.backends.s3boto3.S3Boto3Storage'
```

`AWS_ACCESS_KEY_ID` Your Amazon Web Services access key, as a string.

`AWS_SECRET_ACCESS_KEY` Your Amazon Web Services secret access key, as a string.

Note: If `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` are not set, boto3 internally looks up IAM credentials.

`AWS_STORAGE_BUCKET_NAME` Your Amazon Web Services storage bucket name, as a string.

`AWS_DEFAULT_ACL` (optional, `None` or canned ACL, default `public-read`) Must be either `None` or from the list of canned ACLs. If set to `None` then all files will inherit the bucket's ACL.

Warning: The default value of `public-read` is insecure and will be changing to `None` in a future release of django-storages. Please set this explicitly to `public-read` if that is the desired behavior.

`AWS_BUCKET_ACL` (optional, default `public-read`) Only used if `AWS_AUTO_CREATE_BUCKET=True`. The ACL of the created bucket.

Must be either `None` or from the list of canned ACLs. If set to `None` then the bucket will use the AWS account's default.

Warning: The default value of `public-read` is insecure and will be changing to `None` in a future release of django-storages. Please set this explicitly to `public-read` if that is the desired behavior.

`AWS_AUTO_CREATE_BUCKET` (optional) If set to `True` the bucket specified in `AWS_STORAGE_BUCKET_NAME` is automatically created.

`AWS_HEADERS` (optional - boto only, for boto3 see `AWS_S3_OBJECT_PARAMETERS`) If you'd like to set headers sent with each file of the storage:

```
AWS_HEADERS = {
    'Expires': 'Thu, 15 Apr 2010 20:00:00 GMT',
    'Cache-Control': 'max-age=86400',
}
```

`AWS_S3_OBJECT_PARAMETERS` (optional - boto3 only) Use this to set object parameters on your object (such as CacheControl):

```
AWS_S3_OBJECT_PARAMETERS = {
    'CacheControl': 'max-age=86400',
}
```

`AWS_QUERYSTRING_AUTH` (optional; default is `True`) Setting `AWS_QUERYSTRING_AUTH` to `False` to remove query parameter authentication from generated URLs. This can be useful if your S3 buckets are public.

`AWS_S3_MAX_MEMORY_SIZE` (optional; default is `0` - do not roll over) The maximum amount of memory (in bytes) a file can take up before being rolled over into a temporary file on disk.

`AWS_QUERYSTRING_EXPIRE` (optional; default is 3600 seconds) The number of seconds that a generated URL is valid for.

`AWS_S3_ENCRYPTION` (optional; default is `False`) Enable server-side file encryption while at rest.

`AWS_S3_FILE_OVERWRITE` (optional: default is `True`) By default files with the same name will overwrite each other. Set this to `False` to have extra characters appended.

`AWS_S3_HOST` (optional - boto only, default is `s3.amazonaws.com`)

To ensure you use AWS Signature Version 4 it is recommended to set this to the host of your bucket. See the S3 region list to figure out the appropriate endpoint for your bucket. Also be sure to add `S3_USE_SIGV4 = True` to settings.py

---

Note: The signature versions are not backwards compatible so be careful about url endpoints if making this change for legacy projects.

`AWS_LOCATION` (optional: default is '') A path prefix that will be prepended to all uploads

`AWS_IS_GZIPPED` (optional: default is `False`) Whether or not to enable gzipping of content types specified by `GZIP_CONTENT_TYPES`

GZIP_CONTENT_TYPES (optional: default is text/css, text/javascript, application/javascript, applicatio
When AWS_IS_GZIPPED is set to True the content types which will be gzipped

AWS_S3_REGION_NAME (optional: default is None) Name of the AWS S3 region to use (eg. eu-west-1)

AWS_S3_USE_SSL (optional: default is True) Whether or not to use SSL when connecting to S3.

AWS_S3_VERIFY (optional: default is None - boto3 only) Whether or not to verify the connection to S3. Can be set to False to not verify certificates or a path to a CA cert bundle.

AWS_S3_ENDPOINT_URL (optional: default is None, boto3 only) Custom S3 URL to use when connecting to S3, including scheme. Overrides AWS_S3_REGION_NAME and AWS_S3_USE_SSL. To avoid AuthorizationQueryParametersError error, AWS_S3_REGION_NAME should also be set.

AWS_S3_ADDRESSING_STYLE (default is None, boto3 only) Possible values virtual and path.

AWS_S3_PROXIES (boto3 only, default None) A dictionary of proxy servers to use by protocol or endpoint, e.g.: {'http': 'foo.bar:3128', 'http://hostname': 'foo.bar:4012'}.

Note: The minimum required version of boto3 to use this feature is 1.4.4

AWS_S3_CALLING_FORMAT (optional: default is SubdomainCallingFormat()) Defines the S3 calling format to use to connect to the static bucket.

AWS_S3_SIGNATURE_VERSION (optional - boto3 only)

As of boto3 version 1.4.4 the default signature version is s3v4.

Set this to use an alternate version such as s3. Note that only certain regions support the legacy s3 (also known as v2) version. You can check to see if your region is one of them in the S3 region list.

Note: The signature versions are not backwards compatible so be careful about url endpoints if making this change for legacy projects.

1.1.2 Migrating from Boto to Boto3

Migration from the boto-based to boto3-based backend should be straightforward and painless.

The following adjustments to settings are required:

- Rename AWS_HEADERS to AWS_S3_OBJECT_PARAMETERS and change the format of the key names as in the following example: cache-control becomes CacheControl.
- Raname AWS_ORIGIN to AWS_S3_REGION_NAME
- If AWS_S3_CALLING_FORMAT is set to VHostCallingFormat set AWS_S3_ADDRESSING_STYLE to virtual
- Replace the combination of AWS_S3_HOST and AWS_S3_PORT with AWS_S3_ENDPOINT_URL
- Extract the region name from AWS_S3_HOST and set AWS_S3_REGION_NAME
- Replace AWS_S3_PROXY_HOST and AWS_S3_PROXY_PORT with AWS_S3_PROXIES
- If using signature version s3v4 you can remove S3_USE_SIGV4
- If you persist urls and rely on the output to use the signature version of s3 set AWS_S3_SIGNATURE_VERSION to s3

- Update `DEFAULT_FILE_STORAGE` and/or `STATICFILES_STORAGE` to `storages.backends.s3boto3.S3Boto3Storage`

Additionally, you must install `boto3` (`boto` is no longer required). In order to use all currently supported features, `1.4.4` is the minimum required version although we always recommend the most recent.

Please open an issue on the GitHub repo if any further issues are encountered or steps were omitted.

1.1.3 CloudFront

If you're using S3 as a CDN (via CloudFront), you'll probably want this storage to serve those files using that:

```
AWS_S3_CUSTOM_DOMAIN = 'cdn.mydomain.com'
```

Warning: Django's `STATIC_URL` must end in a slash and the `AWS_S3_CUSTOM_DOMAIN` *must not*. It is best to set this variable independently of `STATIC_URL`.

Keep in mind you'll have to configure CloudFront to use the proper bucket as an origin manually for this to work.

If you need to use multiple storages that are served via CloudFront, pass the *custom_domain* parameter to their constructors.

1.1.4 IAM Policy

The IAM policy permissions needed for most common use cases are:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObjectAcl",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:DeleteObject",
                "s3:PutObjectAcl"
            ],
            "Resource": [
                "arn:aws:s3:::example-bucket-name/*",
                "arn:aws:s3:::example-bucket-name"
            ]
        }
    ]
}
```

1.1.5 Storage

Standard file access options are available, and work as expected:

```
>>> from django.core.files.storage import default_storage
>>> default_storage.exists('storage_test')
False
>>> file = default_storage.open('storage_test', 'w')
>>> file.write('storage contents')
>>> file.close()

>>> default_storage.exists('storage_test')
True
>>> file = default_storage.open('storage_test', 'r')
>>> file.read()
'storage contents'
>>> file.close()

>>> default_storage.delete('storage_test')
>>> default_storage.exists('storage_test')
False
```

1.1.6 Model

An object without a file has limited functionality:: from django.db import models

class MyModel(models.Model): normal = models.FileField()

```
>>> obj1 = MyModel()
>>> obj1.normal
<FieldFile: None>
>>> obj1.normal.size
Traceback (most recent call last):
...
ValueError: The 'normal' attribute has no file associated with it.
```

Saving a file enables full functionality:

```
>>> obj1.normal.save('django_test.txt', ContentFile(b'content'))
>>> obj1.normal
<FieldFile: tests/django_test.txt>
>>> obj1.normal.size
7
>>> obj1.normal.read()
'content'
```

Files can be read in a little at a time, if necessary:

```
>>> obj1.normal.open()
>>> obj1.normal.read(3)
'con'
>>> obj1.normal.read()
'tent'
>>> '-'.join(obj1.normal.chunks(chunk_size=2))
'co-nt-en-t'
```

Save another file with the same name:

```
>>> obj2 = MyModel()
>>> obj2.normal.save('django_test.txt', ContentFile(b'more content'))
```

(continues on next page)

(continued from previous page)

```
>>> obj2.normal
<FieldFile: tests/django_test.txt>
>>> obj2.normal.size
12
```

Push the objects into the cache to make sure they pickle properly:

```
>>> cache.set('obj1', obj1)
>>> cache.set('obj2', obj2)
>>> cache.get('obj2').normal
<FieldFile: tests/django_test.txt>
```

Clean up the temporary files:

```
>>> obj1.normal.delete()
>>> obj2.normal.delete()
```

CHAPTER 2

Apache Libcloud

Apache Libcloud is an API wrapper around a range of cloud storage providers. It aims to provide a consistent API for dealing with cloud storage (and, more broadly, the many other services provided by cloud providers, such as device provisioning, load balancer configuration, and DNS configuration).

Use pip to install apache-libcloud from PyPI:

```
pip install apache-libcloud
```

As of v0.10.1, Libcloud supports the following cloud storage providers:

- Amazon S3
- Google Cloud Storage
- Nimbus.io
- Ninefold Cloud Storage
- Rackspace CloudFiles

Libcloud can also be configured with relatively little effort to support any provider using EMC Atmos storage, or the OpenStack API.

2.1 Settings

2.1.1 `LIBCLOUD_PROVIDERS`

This setting is required to configure connections to cloud storage providers. Each entry corresponds to a single 'bucket' of storage. You can have multiple buckets for a single service provider (e.g., multiple S3 buckets), and you can define buckets at multiple providers. For example, the following configuration defines 3 providers: two buckets (`bucket-1` and `bucket-2`) on a US-based Amazon S3 store, and a third bucket (`bucket-3`) on Google:

```
LIBCLOUD_PROVIDERS = {
    'amazon_1': {
        'type': 'libcloud.storage.types.Provider.S3_US_STANDARD_HOST',
        'user': '<your username here>',
        'key': '<your key here>',
        'bucket': 'bucket-1',
    },
    'amazon_2': {
        'type': 'libcloud.storage.types.Provider.S3_US_STANDARD_HOST',
        'user': '<your username here>',
        'key': '<your key here>',
        'bucket': 'bucket-2',
    },
    'google': {
        'type': 'libcloud.storage.types.Provider.GOOGLE_STORAGE',
        'user': '<Your Google APIv1 username>',
        'key': '<Your Google APIv1 Key>',
        'bucket': 'bucket-3',
    },
}
```

The values for the `type`, `user` and `key` arguments will vary depending on your storage provider:

Amazon S3:

type: `libcloud.storage.types.Provider.S3_US_STANDARD_HOST`,

user: Your AWS access key ID

key: Your AWS secret access key

If you want to use a availability zone other than the US default, you can use one of `S3_US_WEST_HOST`, `S3_US_WEST_OREGON_HOST`, `S3_EU_WEST_HOST`, `S3_AP_SOUTHEAST_HOST`, or `S3_AP_NORTHEAST_HOST` instead of `S3_US_STANDARD_HOST`.

Google Cloud Storage:

type: `libcloud.storage.types.Provider.GOOGLE_STORAGE`,

user: Your Google APIv1 username (20 characters)

key: Your Google APIv1 key

Nimbus.io:

type: `libcloud.storage.types.Provider.NIMBUS`,

user: Your Nimbus.io user ID

key: Your Nimbus.io access key

Ninefold Cloud Storage:

type: `libcloud.storage.types.Provider.NINEFOLD`,

user: Your Atmos Access Token

key: Your Atmos Shared Secret

Rackspace Cloudfiles:

type: `libcloud.storage.types.Provider.CLOUDFIULES_US` or `libcloud.storage.types.Provider.CLOUDFIULES_UK`,

user: Your Rackspace user ID

key: Your Rackspace access key

You can specify any bucket name you want; however, the bucket must exist before you can start using it. If you need to create the bucket, you can use the storage API. For example, to create `bucket-1` from our previous example:

```
>>> from storages.backends.apache_libcloud import LibCloudStorage
>>> store = LibCloudStorage('amazon_1')
>>> store.driver.create_container('bucket-1')
```

2.1.2 DEFAULT_LIBCLOUD_PROVIDER

Once you have defined your Libcloud providers, you have the option of setting one provider as the default provider of Libcloud storage. This is done setting `DEFAULT_LIBCLOUD_PROVIDER` to the key in `LIBCLOUD_PROVIDER` that you want to use as the default provider. For example, if you want the `amazon-1` provider to be the default provider, use:

```
DEFAULT_LIBCLOUD_PROVIDER = 'amazon-1'
```

If `DEFAULT_LIBCLOUD_PROVIDER` isn't set, the Libcloud backend will assume that the default storage backend is named `default`. Therefore, you can avoid settings DEFAULT_LIBCLOUD_PROVIDER by simply naming one of your Libcloud providers `default`:

```
LIBCLOUD_PROVIDERS = {
    'default': {
        'type': ...
    },
}
```

2.1.3 DEFAULT_FILE_STORAGE

If you want your Libcloud storage to be the default Django file store, you can set:

```
DEFAULT_FILE_STORAGE = 'storages.backends.apache_libcloud.LibCloudStorage'
```

Your default Libcloud provider will be used as the file store.

2.2 Certificate authorities

Libcloud uses HTTPS connections, and in order to validate that these HTTPS connections are correctly signed, root CA certificates must be present. On some platforms (most notably, OS X and Windows), the required certificates may not be available by default. To test

```
>>> from storages.backends.apache_libcloud import LibCloudStorage
>>> store = LibCloudStorage('amazon_1')
Traceback (most recent call last):
...
ImproperlyConfigured: Unable to create libcloud driver type libcloud.storage.types.
↪Provider.S3_US_STANDARD_HOST: No CA Certificates were found in CA_CERTS_PATH.
```

If you get this error, you need to install a certificate authority. Download a certificate authority file, and then put the following two lines into your settings.py:

```
import libcloud.security
libcloud.security.CA_CERTS_PATH.append("/path/to/your/cacerts.pem")
```

CHAPTER 3

Azure Storage

A custom storage system for Django using Windows Azure Storage backend.

3.1 Notes

Be aware Azure file names have some extra restrictions. They can't:

- end with a dot (`.`) or slash (`/`)
- contain more than 256 slashes (`/`)
- be longer than 1024 characters

This is usually not an issue, since some file-systems won't allow this anyway. There's `default_storage.get_name_max_len()` method to get the `max_length` allowed. This is useful for form inputs. It usually returns `1024 - len(azure_location_setting)`. There's `default_storage.get_valid_name(...)` method to clean up file names when migrating to Azure.

Gzipping for static files must be done through Azure CDN.

3.2 Install

Install Azure SDK:

```
pip install django-storages[azure]
```

3.3 Private VS Public Access

The `AzureStorage` allows a single container. The container may have either public access or private access. When dealing with a private container, the `AZURE_URL_EXPIRATION_SECS` must be set to get temporary URLs.

A common setup is having private media files and public static files, since public files allow for better caching (i.e: no query-string within the URL).

One way to support this is having two backends, a regular `AzureStorage` with the private container and expiration setting set, and a custom backend (i.e: a subclass of `AzureStorage`) for the public container.

Custom backend:

```
# file: ./custom_storage/custom_azure.py
class PublicAzureStorage(AzureStorage):
    account_name = 'myaccount'
    account_key = 'mykey'
    azure_container = 'mypublic_container'
    expiration_secs = None
```

Then on settings set:

```
DEFAULT_FILE_STORAGE = 'storages.backends.azure_storage.AzureStorage'
STATICFILES_STORAGE = 'custom_storage.custom_azure.PublicAzureStorage'
```

3.4 Settings

The following settings should be set within the standard django configuration file, usually *settings.py*.

Set the default storage (i.e: for media files) and the static storage (i.e: fo static files) to use the azure backend:

```
DEFAULT_FILE_STORAGE = 'storages.backends.azure_storage.AzureStorage'
STATICFILES_STORAGE = 'storages.backends.azure_storage.AzureStorage'
```

The following settings are available:

`AZURE_ACCOUNT_NAME`

> This setting is the Windows Azure Storage Account name, which in many cases is also the first part of the url for instance: http://azure_account_name.blob.core.windows.net/ would mean:
>
> ```
> AZURE_ACCOUNT_NAME = "azure_account_name"
> ```

`AZURE_ACCOUNT_KEY`

> This is the private key that gives Django access to the Windows Azure Account.

`AZURE_CONTAINER`

> This is where the files uploaded through Django will be uploaded. The container must be already created, since the storage system will not attempt to create it.

`AZURE_SSL`

> Set a secure connection (HTTPS), otherwise it makes an insecure connection (HTTP). Default is `True`

`AZURE_UPLOAD_MAX_CONN`

> Number of connections to make when uploading a single file. Default is `2`

`AZURE_CONNECTION_TIMEOUT_SECS`

> Global connection timeout in seconds. Default is `20`

`AZURE_BLOB_MAX_MEMORY_SIZE`

Maximum memory used by a downloaded file before dumping it to disk. Unit is in bytes. Default is `2MB`

`AZURE_URL_EXPIRATION_SECS`

Seconds before a URL expires, set to `None` to never expire it. Be aware the container must have public read permissions in order to access a URL without expiration date. Default is `None`

`AZURE_OVERWRITE_FILES`

Overwrite an existing file when it has the same name as the file being uploaded. Otherwise, rename it. Default is `False`

`AZURE_LOCATION`

Default location for the uploaded files. This is a path that gets prepended to every file name.

`AZURE_EMULATED_MODE`

Whether to use the emulator (i.e Azurite). Defaults to False.

`AZURE_ENDPOINT_SUFFIX`

The host base component of the url, minus the account name. Defaults to Azure (`core.windows.net`). Override this to use the China cloud (`core.chinacloudapi.cn`).

`AZURE_CUSTOM_DOMAIN`

The custom domain to use. This can be set in the Azure Portal. For example, `www.mydomain.com` or `mycdn.azureedge.net`.

It may contain a `host:port` when using the emulator (`AZURE_EMULATED_MODE = True`).

`AZURE_CONNECTION_STRING`

If specified, this will override all other parameters. See http://azure.microsoft.com/en-us/documentation/articles/storage-configure-connection-string/ for the connection string format.

`AZURE_CUSTOM_CONNECTION_STRING`

This is similar to `AZURE_CONNECTION_STRING`, but it's used when generating the file's URL. A custom domain or CDN may be specified here instead of within `AZURE_CONNECTION_STRING`. Defaults to `AZURE_CONNECTION_STRING`'s value.

`AZURE_TOKEN_CREDENTIAL`

A token credential used to authenticate HTTPS requests. The token value should be updated before its expiration.

CHAPTER 4

Digital Ocean

Digital Ocean Spaces implements the S3 protocol. To use it follow the instructions in the *Amazon S3 docs* with the important caveats that you must:

- Set `AWS_S3_REGION_NAME` to your Digital Ocean region (such as `nyc3` or `sfo2`)
- Set `AWS_S3_ENDPOINT_URL` to the value of `${AWS_S3_REGION_NAME}.digitaloceanspaces.com`
- Set the values of `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` to the corresponding values from Digital Ocean

CHAPTER 5

Dropbox

A Django files storage using Dropbox as a backend via the official Dropbox SDK for Python. Currently only v2 of the API is supported.

Before you start configuration, you will need to install the SDK which can be done for you automatically by doing:

```
pip install django-storages[dropbox]
```

5.1 Settings

To use DropBoxStorage set:

```
DEFAULT_FILE_STORAGE = 'storages.backends.dropbox.DropBoxStorage'
```

DROPBOX_OAUTH2_TOKEN Your Dropbox token. You can obtain one by following the instructions in the tutorial.

DROPBOX_ROOT_PATH (optional) Allow to jail your storage to a defined directory.

DROPBOX_TIMEOUT (optional) Timeout in seconds for making requests to the API. If `None`, the client will wait forever. The default is `100` seconds which is the current default in the official SDK.

CHAPTER 6

FTP

Warning: This FTP storage is not prepared to work with large files, because it uses memory for temporary data storage. It also does not close FTP connection automatically (but open it lazy and try to reestablish when disconnected).

This implementation was done preliminary for upload files in admin to remote FTP location and read them back on site by HTTP. It was tested mostly in this configuration, so read/write using FTPStorageFile class may break.

6.1 Settings

LOCATION URL of the server that holds the files. Example `'ftp://<user>:<pass>@<host>:<port>'`

BASE_URL URL that serves the files stored at this location. Defaults to the value of your `MEDIA_URL` setting.

CHAPTER 7

Google Cloud Storage

This backend provides Django File API for Google Cloud Storage using the Python library provided by Google.

7.1 Installation

Use pip to install from PyPI:

```
pip install django-storages[google]
```

7.2 Authentication

By default, this library will try to use the credentials associated with the current Google Compute Engine (GCE) or Google Kubernetes Engine (GKE) instance for authentication. In most cases, the default service accounts are not sufficient to read/write and sign files in GCS.

1. Create a service account. (Google Getting Started Guide)
2. Create the key and download *your-project-XXXXX.json* file.
3. Make sure your service account has access to the bucket and appropriate permissions. (Using IAM Permissions)
4. The key must be mounted/available to your running Django app. Note: a json keyfile will work for developer machines (or other instances outside Google infrastructure).
5. Set an environment variable of GOOGLE_APPLICATION_CREDENTIALS to the path of the json file.

Alternatively, you can use the setting *GS_CREDENTIALS* as described below.

7.3 Getting Started

Set the default storage and bucket name in your settings.py file:

```
DEFAULT_FILE_STORAGE = 'storages.backends.gcloud.GoogleCloudStorage'
GS_BUCKET_NAME = 'YOUR_BUCKET_NAME_GOES_HERE'
```

Once you're done, default_storage will be Google Cloud Storage:

```
>>> from django.core.files.storage import default_storage
>>> print default_storage.__class__
<class 'storages.backends.gcloud.GoogleCloudStorage'>
```

This way, if you define a new FileField, it will use the Google Cloud Storage:

```
>>> from django.db import models
>>> class Resume(models.Model):
...     pdf = models.FileField(upload_to='pdfs')
...     photos = models.ImageField(upload_to='photos')
...
>>> resume = Resume()
>>> print resume.pdf.storage
<storages.backends.gcloud.GoogleCloudStorage object at ...>
```

7.4 Settings

To use gcloud set:

```
DEFAULT_FILE_STORAGE = 'storages.backends.gcloud.GoogleCloudStorage'
```

`GS_BUCKET_NAME`

Your Google Storage bucket name, as a string. Required.

`GS_PROJECT_ID` (optional)

Your Google Cloud project ID. If unset, falls back to the default inferred from the environment.

`GS_CREDENTIALS` (optional)

The OAuth 2 credentials to use for the connection. If unset, falls back to the default inferred from the environment (i.e. GOOGLE_APPLICATION_CREDENTIALS)

```
from google.oauth2 import service_account

GS_CREDENTIALS = service_account.Credentials.from_service_account_file(
    "path/to/credentials.json"
)
```

`GS_AUTO_CREATE_BUCKET` (optional, default is `False`)

If True, attempt to create the bucket if it does not exist.

`GS_AUTO_CREATE_ACL` (optional, default is `projectPrivate`)

ACL used when creating a new bucket, from the list of predefined ACLs. (A "JSON API" ACL is preferred but an "XML API/gsutil" ACL will be translated.)

Note that the ACL you select must still give the service account running the GCE backend to have OWNER permission on the bucket. If you're using the default service account, this means you're restricted to the `projectPrivate` ACL.

`GS_DEFAULT_ACL` (optional, default is None)

ACL used when creating a new blob, from the list of predefined ACLs. (A "JSON API" ACL is preferred but an "XML API/gsutil" ACL will be translated.)

For most cases, the blob will need to be set to the `publicRead` ACL in order for the file to be viewed. If GS_DEFAULT_ACL is not set, the blob will have the default permissions set by the bucket.

`publicRead` files will return a public, non-expiring url. All other files return a signed (expiring) url.

Note: GS_DEFAULT_ACL must be set to 'publicRead' to return a public url. Even if you set the bucket to public or set the file permissions directly in GCS to public.

`GS_FILE_CHARSET` (optional)

Allows overriding the character set used in filenames.

`GS_FILE_OVERWRITE` (optional: default is `True`)

By default files with the same name will overwrite each other. Set this to `False` to have extra characters appended.

`GS_MAX_MEMORY_SIZE` (optional)

The maximum amount of memory a returned file can take up (in bytes) before being rolled over into a temporary file on disk. Default is 0: Do not roll over.

`GS_BLOB_CHUNK_SIZE` (optional: default is `None`)

The size of blob chunks that are sent via resumable upload. If this is not set then the generated request must fit in memory. Recommended if you are going to be uploading large files.

Note: This must be a multiple of 256K (1024 * 256)

`GS_CACHE_CONTROL` (optional: default is `None`)

Sets Cache-Control HTTP header for the file, more about HTTP caching can be found here

`GS_LOCATION` (optional: default is `''`)

Subdirectory in which the files will be stored. Defaults to the root of the bucket.

`GS_EXPIRATION` (optional: default is `timedelta(seconds=86400)`)

The time that a generated URL is valid before expiration. The default is 1 day. Public files will return a url that does not expire. Files will be signed by the credentials provided to django-storages (See GS_CREDENTIALS).

Note: Default Google Compute Engine (GCE) Service accounts are unable to sign urls.

The `GS_EXPIRATION` value is handled by the underlying Google library. It supports *timedelta*, *datetime*, or *integer* seconds since epoch time.

7.5 Usage

7.5.1 Fields

Once you're done, default_storage will be Google Cloud Storage:

```
>>> from django.core.files.storage import default_storage
>>> print default_storage.__class__
<class 'storages.backends.gcloud.GoogleCloudStorage'>
```

This way, if you define a new FileField, it will use the Google Cloud Storage:

```
>>> from django.db import models
>>> class Resume(models.Model):
...     pdf = models.FileField(upload_to='pdfs')
...     photos = models.ImageField(upload_to='photos')
...
>>> resume = Resume()
>>> print resume.pdf.storage
<storages.backends.gcloud.GoogleCloudStorage object at ...>
```

7.5.2 Storage

Standard file access options are available, and work as expected:

```
>>> default_storage.exists('storage_test')
False
>>> file = default_storage.open('storage_test', 'w')
>>> file.write('storage contents')
>>> file.close()

>>> default_storage.exists('storage_test')
True
>>> file = default_storage.open('storage_test', 'r')
>>> file.read()
'storage contents'
>>> file.close()

>>> default_storage.delete('storage_test')
>>> default_storage.exists('storage_test')
False
```

7.5.3 Model

An object without a file has limited functionality:

```
>>> obj1 = Resume()
>>> obj1.pdf
<FieldFile: None>
>>> obj1.pdf.size
Traceback (most recent call last):
...
ValueError: The 'pdf' attribute has no file associated with it.
```

Saving a file enables full functionality:

```
>>> obj1.pdf.save('django_test.txt', ContentFile('content'))
>>> obj1.pdf
<FieldFile: tests/django_test.txt>
>>> obj1.pdf.size
7
>>> obj1.pdf.read()
'content'
```

Files can be read in a little at a time, if necessary:

```
>>> obj1.pdf.open()
>>> obj1.pdf.read(3)
'con'
>>> obj1.pdf.read()
'tent'
>>> '-'.join(obj1.pdf.chunks(chunk_size=2))
'co-nt-en-t'
```

Save another file with the same name:

```
>>> obj2 = Resume()
>>> obj2.pdf.save('django_test.txt', ContentFile('more content'))
>>> obj2.pdf
<FieldFile: tests/django_test_.txt>
>>> obj2.pdf.size
12
```

Push the objects into the cache to make sure they pickle properly:

```
>>> cache.set('obj1', obj1)
>>> cache.set('obj2', obj2)
>>> cache.get('obj2').pdf
<FieldFile: tests/django_test_.txt>
```

Deleting an object deletes the file it uses, if there are no other objects still using that file:

```
>>> obj2.delete()
>>> obj2.pdf.save('django_test.txt', ContentFile('more content'))
>>> obj2.pdf
<FieldFile: tests/django_test_.txt>
```

CHAPTER 8

SFTP

8.1 Settings

SFTP_STORAGE_HOST The hostname where you want the files to be saved.

SFTP_STORAGE_ROOT The root directory on the remote host into which files should be placed. Should work the same way that `STATIC_ROOT` works for local files. Must include a trailing slash.

SFTP_STORAGE_PARAMS (optional) A dictionary containing connection parameters to be passed as keyword arguments to `paramiko.SSHClient().connect()` (do not include hostname here). See paramiko SSHClient.connect() documentation for details

SFTP_STORAGE_INTERACTIVE (optional) A boolean indicating whether to prompt for a password if the connection cannot be made using keys, and there is not already a password in `SFTP_STORAGE_PARAMS`. You can set this to `True` to enable interactive login when running `manage.py collectstatic`, for example.

> **Warning:** DO NOT set SFTP_STORAGE_INTERACTIVE to True if you are using this storage for files being uploaded to your site by users, because you'll have no way to enter the password when they submit the form..

SFTP_STORAGE_FILE_MODE (optional) A bitmask for setting permissions on newly-created files. See Python os.chmod documentation for acceptable values.

SFTP_STORAGE_DIR_MODE (optional) A bitmask for setting permissions on newly-created directories. See Python os.chmod documentation for acceptable values.

Note: Hint: if you start the mode number with a 0 you can express it in octal just like you would when doing "chmod 775 myfile" from bash.

SFTP_STORAGE_UID (optional) UID of the account that should be set as the owner of the files on the remote host. You may have to be root to set this.

SFTP_STORAGE_GID (optional) GID of the group that should be set on the files on the remote host. You have to be a member of the group to set this.

SFTP_KNOWN_HOST_FILE (optional) Absolute path of know host file, if it isn't set `"~/.ssh/known_hosts"` will be used.

CHAPTER 9

Installation

Use pip to install from PyPI:

```
pip install django-storages
```

Each storage backend has its own unique settings you will need to add to your settings.py file. Read the documentation for your storage engine(s) of choice to determine what you need to add.

CHAPTER 10

Contributing

To contribute to django-storages create a fork on GitHub. Clone your fork, make some changes, and submit a pull request.

CHAPTER 11

Issues

Use the GitHub issue tracker for django-storages to submit bugs, issues, and feature requests.

CHAPTER 12

Indices and tables

- genindex
- modindex
- search