django-dbbackup Documentation

Release 2.4.1

Michael Shepanski

June 07, 2016

Contents

This Django application provides management commands to help backup and restore your project database to AmazonS3, Dropbox or Local Disk.

- Keep your important data secure and offsite.
- Use Crontab or Celery to setup automated backups.
- Great to keep your development database up to date.

Contents:

CHAPTER 1

Installation

1.1 Installing on your system

1.1.1 Getting the latest stable release

```
pip install django-dbbackup
```

1.1.2 Getting the latest release from trunk

In general, you should not be downloading and installing stuff directly off repositories – especially not if you are backing up sensitive data.

Security is important, bypassing PyPi repositories is a bad habbit, because it will bypass the fragile key signatures authentication that are at least present when using PyPi repositories.

```
pip install -e git+https://github.com/mjs7231/django-dbbackup.git#egg=django-dbbackup
```

1.2 Add it in your project

In your `settings.py`, make sure you have the following things:

```
INSTALLED_APPS = (
    ...
    'dbbackup',  # django-dbbackup
)

DBBACKUP_STORAGE = 'dbbackup.storage.filesystem_storage'
DBBACKUP_STORAGE_OPTIONS = {'location': '/var/backups'}
```

This configuration uses filesystem storage, but you can use any storage supported by Django API. See *storage* for more information about it.

1.3 Testing that everything worked

Now, you should be able to create your first backup by running:

```
$ python manage.py dbbackup
```

If your database was called `default` which is the normal Django behaviour of a single-database project, you should now see a new file in your backup directory.

CHAPTER 2

Configuration

2.1 General settings

2.1.1 DBBACKUP_DATABASES

List of key entries for `settings.DATABASES` which shall be used to connect and create database backups.

Default: `list(settings.DATABASES.keys())` (keys of all entries listed)

2.1.2 DBBACKUP_BACKUP_DIRECTORY

Where to store backups. String pointing to django-dbbackup location module to use when performing a backup.

Default: `os.getcwd()` (Current working directory)

2.1.3 DBBACKUP_TMP_DIR

Directory to be used for temporary files.

Default: `tempfile.gettempdir()`

2.1.4 DBBACKUP_TMP_FILE_MAX_SIZE

Maximum size in bytes for file handling in memory before write a temporary file on `DBBACKUP_TMP_DIR`.

Default: `10*1024*1024`

2.1.5 DBBACKUP_CLEANUP_KEEP and DBBACKUP_CLEANUP_KEEP_MEDIA

When issueing `dbbackup` and `mediabackup`, old backup files are looked for and removed.

Default: `10` (days)

2.1.6 DBBACKUP_MEDIA_PATH

Default: settings.MEDIA_ROOT

2.1.7 DBBACKUP_DATE_FORMAT

Date format to use for naming files. It must contain only alphanumerical characters, `'_'`, `'-'` or `'%'`.

Default: `'%Y-%m-%d-%H%M%S'`

2.1.8 DBBACKUP_FILENAME_TEMPLATE

The template to use when generating the backup filename. By default this is `'{databasename}-{servername}-{datetime}.{extension}'`. This setting can also be made a function which takes the following keyword arguments:

```
def backup_filename(databasename, servername, datetime, extension):
    pass

DBBACKUP_FILENAME_TEMPLATE = backup_filename
```

This allows you to modify the entire format of the filename, for example, if you want to take advantage of Amazon S3's automatic expiry feature, you need to prefix your backups differently based on when you want them to expire.

`{datetime}` is rendered with `DBBACKUP_DATE_FORMAT`.

2.1.9 DBBACKUP_MEDIA_FILENAME_TEMPLATE

Same as `DBBACKUP_FILENAME_TEMPLATE` but for media files backups.

2.1.10 DBBACKUP_MYSQL_EXTENSION

The file name extension used for MySQL backups.

Default: `'mysql'`

2.1.11 DBBACKUP_POSTGRESQL_EXTENSION

The file name extension used for Postgres and PostGIS backups.

Default: `'psql'`

2.1.12 DBBACKUP_SQLITE_EXTENSION

The file name extension used for SQLite backups.

Default: `'sqlite'`

2.1.13 DBBACKUP_SEND_EMAIL

Controls whether or not django-dbbackup sends an error email when an uncaught exception is received.

Default: `True`

2.1.14 DBBACKUP_HOSTNAME

Hostname needed by django-dbbackup's uncaught exception email sender for well described error reporting. If you are using `ALLOWED_HOSTS` you should set `DBBACKUP_HOSTNAME` to any host from `ALLOWED_HOSTS` setting. Otherwise django-dbbackup can not send email to the `SERVER_EMAIL`.

Default: `socket.gethostname()`

Note: Previously `DBBACKUP_FAKE_HOST` was used for this setting.

DBBACKUP_CLEANUP_KEEP (optional) - The number of backups to keep when specifying the –clean flag. Defaults to keeping 10 + the first backup of each month.

CHAPTER 3

Database settings

The following databases are supported by this application. You can customize the commands used for backup and the resulting filenames with the following settings.

NOTE: The {adminuser} settings below will first check for the variable ADMINUSER specified on the database, then fall back to USER. This allows you supplying a different user to perform the admin commands dropdb, createdb as a different user from the one django uses to connect. If you need more fine grain control you might consider fully customizing the admin commands.

3.1 Postgresql

3.1.1 DBBACKUP_POSTGRESQL_RESTORE_SINGLE_TRANSACTION

When doing a restore with postgres, wrap everything in a single transaction so that errors cause a rollback.

Default: `True`

3.1.2 DBBACKUP_POSTGIS_SPACIAL_REF

When on Postgis, using this setting currently disables `CREATE EXTENSION POSTGIS;`. Ideally, it should run the good old Postgis templates for version 1.5 of Postgis.

CHAPTER 4

Encrypting your backups

Considering that you might be putting secured data on external servers and perhaps untrusted servers where it gets forgotten over time, it's always a good idea to encrypt backups.

Just remember to keep the encryption keys safe, too!

4.1 PGP

You can encrypt a backup with the `--encrypt` option. The backup is done using gpg.

```
python manage.py dbbackup --encrypt
```

...or when restoring from an encrypted backup:

```
python manage.py dbrestore --decrypt
```

Requirements:

- Install the python package python-gnupg: `pip install python-gnupg`.
- You need gpg key.
- Set the setting 'DBBACKUP_GPG_RECIPIENT' to the name of the gpg key.

DBBACKUP_GPG_ALWAYS_TRUST (optional) - The encryption of the backup file fails if gpg does not trust the public encryption key. The solution is to set the option 'trust-model' to 'always'. By default this value is False. Set this to True to enable this option.

DBBACKUP_GPG_RECIPIENT (optional) - The name of the key that is used for encryption. This setting is only used when making a backup with the `--encrypt` or `--decrypt` option.

CHAPTER 5

Remote storage

django-dbbackup comes with a variety of remote storage options and it can deal with Django Storage API for extend its possibilities.

You can choose your storage backend by set `settings.DBBACKUP_STORAGE`, it must point to module containing the chosen Storage class. For example: `dbbackup.storage.filesystem_storage` for use file system storage. Below, we'll list some of the available solutions and their options.

Storage's option are gathered in `settings.DBBACKUP_STORAGE_OPTIONS` which is a dictionary of keywords representing how to configure it.

Note: A lot of changes has been made for use Django Storage API as primary source of backends and due to this task, some settings has been deprecated but always functionnal until removing. Please take care of notes and warnings in this documentation and at your project's launching.

Warning: Do not configure backup storage with the same configuration than your media files, you'll risk to share backups inside public directories.

5.1 Local disk

Dbbackup uses built-in file system storage to manage files on a local directory.

Note: Storing backups to local disk may also be useful for Dropbox if you already have the offical Dropbox client installed on your system.

5.1.1 Setup

To store your backups on the local file system, simply setup the required settings below.

```
DBBACKUP_STORAGE = 'dbbackup.storage.filesystem_storage'
DBBACKUP_STORAGE_OPTIONS = {'location': '/my/backup/dir/'}
```

5.1.2 Available Settings

location - Default: Current working directory (`os.getcwd`)

Absolute path to the directory that will hold the files.

> **Warning:** `settings.DBBACKUP_BACKUP_DIRECTORY` was used before but is deprecated. Backup location must no be in `settings.MEDIA_ROOT`, it will raise an `StorageError` if `settings.DEBUG` is `False` else a warning.

file_permissions_mode - Default: `settings.FILE_UPLOAD_PERMISSIONS`

The file system permissions that the file will receive when it is saved.

5.2 Amazon S3

Our S3 backend uses Django Storage Redux which uses boto.

5.2.1 Setup

In order to backup to Amazon S3, you'll first need to create an Amazon Webservices Account and setup your Amazon S3 bucket. Once that is complete, you can follow the required setup below.

```
pip install boto django-storages-redux
```

Add the following to your project's settings:

```
DBBACKUP_STORAGE = 'dbbackup.storage.s3_storage'
DBBACKUP_STORAGE_OPTIONS = {
    'access_key': 'my_id',
    'secret_key': 'my_secret',
    'bucket_name': 'my_bucket_name'
}
```

5.2.2 Available Settings

> **Note:** More settings are available but without clear official documentation about it, you can refer to source code and look at `S3BotoStorage`'s attributes.

access_key - Required

Your AWS access key as string. This can be found on your Amazon Account Security Credentials page.

> **Note:** `settings.DBBACKUP_S3_ACCESS_KEY` was used before but is deprecated.

secret_key - Required

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

> **Note:** `settings.DBBACKUP_S3_SECRET_KEY` was used before but is deprecated.

bucket_name - Required

Your Amazon Web Services storage bucket name, as a string. This directory must exist before attempting to create your first backup.

Note: `settings.DBBACKUP_S3_BUCKET` was used before but is deprecated.

host - Default: `'s3.amazonaws.com'` (`boto.s3.connection.S3Connection.DefaultHost`)

Specify the Amazon domain to use when transferring the generated backup files. For example, this can be set to `'s3-eu-west-1.amazonaws.com'`.

Note: `settings.DBBACKUP_S3_DOMAIN` was used before but is deprecated.

use_ssl - Default: `True`

Note: `settings.DBBACKUP_S3_IS_SECURE` was used before but is deprecated.

default_acl - Required

If bucket doesn't exist, it will be created with the given ACL.

Warning: The default ACL is *'public-read'*, please take care of this possible security issue.

5.3 Dropbox

In order to backup to Dropbox, you'll first need to create a Dropbox Account and set it up to communicate with the Django-DBBackup application. Don't worry, all instructions are below.

5.3.1 Setup Your Dropbox Account

1. Login to Dropbox and navigate to Developers » MyApps. https://www.dropbox.com/developers/start/setup#python
2. Click the button to create a new app and name it whatever you like. For reference, I named mine 'Website Backups'.
3. After your app is created, note the options button and more importantly the 'App Key' and 'App Secret' values inside. You'll need those later.

5.3.2 Setup Your Django Project

```
pip install dropbox
```

...And make sure you have the following required project settings:

```
DBBACKUP_STORAGE = 'dbbackup.storage.dropbox_storage'
DBBACKUP_TOKENS_FILEPATH = '<local_tokens_filepath>'
DBBACKUP_DROPBOX_APP_KEY = '<dropbox_app_key>'
DBBACKUP_DROPBOX_APP_SECRET = '<dropbox_app_secret>'
```

5.4 FTP

To store your database backups on the remote filesystem via FTP, simply setup the required settings below.

5.4.1 Setup Your Django Project

Note: This storage will be updated for use Django Storage's one.

Warning: This storage doesn't use private connection for communcation, don't use it if you're not sure about the link between client and server.

Using FTP does not require any external libraries to be installed, simply use the below project settings:

```
DBBACKUP_STORAGE = 'dbbackup.storage.ftp_storage'
DBBACKUP_FTP_HOST = 'ftp.host'
DBBACKUP_FTP_USER = 'user, blank if anonymous'
DBBACKUP_FTP_PASSWORD = 'password, can be blank'
DBBACKUP_FTP_PATH = 'path, blank for default'
```

5.4.2 Available Settings

DBBACKUP_FTP_HOST - Required

Hostname for the server you wish to save your backups.

DBBACKUP_FTP_USER - Default: `None`

Authentication login, do not use if anonymous.

DBBACKUP_FTP_PASSWORD - Default: `None`

Authentication password, do not use if there's no password.

DBBACKUP_FTP_PATH - Default: `'.'`

The directory on remote FTP server you wish to save your backups.

Note: As other updated storages, this settings will be deprecated in favor of dictionary `settings.DBBACKUP_STORAGE_OPTIONS`.

5.5 Django built-in storage API

Django has its own storage API for managing media files. Dbbackup allows you to use (third-part) Django storage backends. The default backend is `FileSystemStorage`, which is integrated in Django but we invite you to take a look at django-storages-redux which has a great collection of storage backends.

5.5.1 Setup using built-in storage API

To use Django's built-in FileSystemStorage, add the following lines to your `settings.py`:

```
DBBACKUP_STORAGE = 'dbbackup.storage.builtin_django'
# Default
# DBBACKUP_DJANGO_STORAGE = 'django.core.file.storages.FileSystemStorage'
DBBACKUP_STORAGE_OPTIONS = {'location': '/mybackupdir/'}
```

`'dbbackup.storage.builtin_django'` is a wrapper for use the Django storage defined in `DBBACKUP_DJANGO_STORAGE` with the options defined in `DBBACKUP_STORAGE_OPTIONS`.

5.5.2 Used settings

DBBACKUP_DJANGO_STORAGE - Default: `'django.core.file.storages.FileSystemStorage'`

Path to a Django Storage class (in Python dot style).

> **Warning:** Do not use a Django storage backend without configuring its options, otherwise you will risk mixing media files (with public access) and backups (strictly private).

DBBACKUP_STORAGE_OPTIONS - Default: `{}`

Dictionary used to instantiate a Django Storage class. For example, the `location` key customizes the directory for `FileSystemStorage`.

5.6 Write your custom storage

If you wish to build your own, extend `dbbackup.storage.base.BaseStorage` and point your `settings.DBBACKUP_STORAGE` to `'my_storage.backend.ClassName'`.

CHAPTER 6

Contributing guide

Dbbackup is a free license software where all help are welcomed. This documentation aims to help users or developers to bring their contributions to this project.

6.1 Submit a bug, issue or enhancement

All communication are made with GitHub issues. Do not hesitate to open a issue if:

- You have an improvement idea
- You found a bug
- You've got a question
- More generaly something seems wrong for you

6.2 Make a patch

We use GitHub pull requests for manage all patches. For a better handling of requests we advise you to:

1. Fork the project and make a new branch
2. Make your changes with tests if possible and documentation if needed
3. Push changes to your fork repository and test it with Travis
4. If succeed, open a pull request
5. Upset us until we give you an answer

6.2.1 Test code

You can test your code in local machine with the `runtests.py` script:

```
cd tests
python runtests.py
```

We advise you to launch it with Python 2 & 3 before push and try it in Travis.

6.3 Online CI

We use Travis for tests Dbbackup with a matrix of components' version: Several version of Django and several versions of Python including 2, 3 and PyPy. Code coverage is ensured with Coveralls and has not yet minimum coverage limit.

> **Warning:** django-dbbackup is currently under heavy refactoring, stay tuned for new versions and a final 2.0 release.

CHAPTER 7

Compatibility

Django Database Backup supports PyPy, Python 2.7, 3.2 to 3.4 and Django greater than 1.6.

CHAPTER 8

Management Commands

8.1 dbbackup

Backup your database to the specified storage. By default this will backup all databases specified in your settings.py file and will not delete any old backups. You can optionally specify a server name to be included in the backup filename.

```
dbbackup [-s <servername>] [-d <database>] [--clean] [--compress] [--encrypt] [--backup-extension <fi
```

8.2 dbrestore

Restore your database from the specified storage. By default this will lookup the latest backup and restore from that. You may optionally specify a servername if you you want to backup a database image that was created from a different server. You may also specify an explicit local file to backup from.

```
dbrestore [-d <database>] [-s <servername>] [-f <localfile>] [--uncompress] [--backup-extension <file
```

8.3 mediabackup

Backup media files. Default this will backup the files in the MEDIA_ROOT. Optionally you can set the DBBACKUP_MEDIA_PATH setting.

```
mediabackup [--encrypt] [--clean] [--servername <servername>]
```

CHAPTER 9

Examples

If you run dbbackup out of the box, it will be able to create and restore from a local file dump of your database as configured in your Django project's setup.

Here's how we create a simple dump of the database:

```
$ python manage.py dbbackup

Backing Up Database: /home/user/django-project/db.sqlite3
  Reading: /home/user/django-project/db.sqlite3
  Backup tempfile created: 38.0 KB
  Writing file to Filesystem: /home/user/django-project/, filename: default.backup
```

...and here's how we load that dump again (WARNING! Doing that of course overwrites the entire existing database)

```
$ python manage.py dbrestore

Restoring backup for database: /home/user/django-project/db.sqlite3
  Finding latest backup
  Restoring: /home/user/django-project/default.backup
  Restore tempfile created: 38.0 KB
Are you sure you want to continue? [Y/n]y
  Writing: /home/user/django-project/db.sqlite3
```

Now, databases are not the only thing you should remember to backup. Your `settings.MEDIA_ROOT` is where user contributed uploads reside, and it should also be backed up.

```
$ python manage.py mediabackup

Backing up media files
  Backup tempfile created: None (233.0 B)
  Writing file to Filesystem: /home/user/django-project/
```

CHAPTER 10

MongoDB backup example (BETA)

You can backup a mongodb database defined in your DATABASES settings.

```
DATABASES['my_mongo'] = {
    'USER': 'dumper_user',
    'PASSWORD': '******',
    'ENGINE': 'django_mongodb_engine',
    'NAME': 'db_to_dump',
    'HOST': 'localhost',
    'PORT': '27017',
}
```

```
$ python manage.py dbbackup -d my_mongo

Backing Up Database: db_to_dump
 Running: mongodump --username=dumper_user --password=****** --host=localhost --port=27017 -db db_to_
 Running: tar -C /tmp/tmpxf8P7M -cf - .
 Backup tempfile created: 10.0 KB
 Writing file to  Filesystem: /home/user/django-project/, filename: db_to_dump-2015-07-05-150629.tar
```

You can then restore the backup using the opposite command. (backup_extension currently have to be given)

```
$ python manage.py dbrestore -d my_mongo

Restoring backup for database: db_to_dump
  Finding latest backup
  Restoring: /home/user/django-project/db_to_dump-2015-07-05-150629.tar
  Restore tempfile created: 10.0 KB
Are you sure you want to continue? [Y/n]Y
  Running: tar -C /tmp/tmpiaeb0O -x
  Running: mongorestore --username=dumper_user --password=****** --authenticationDatabase db_to_dump
```

CHAPTER 11

Other Resources

Source code here:

https://github.com/django-dbbackup/django-dbbackup

PyPi project:

https://pypi.python.org/pypi/django-dbbackup/

CHAPTER 12

Indices and tables

- genindex
- modindex
- search