django-dbbackup Documentation

Release 3.3.0

Michael Shepanski

Sep 26, 2020

Contents

This Django application provides management commands to help backup and restore your project database and media files with various storages such as Amazon S3, DropBox or local file system.

It is made for:

- Ensure yours backup with GPG signature and encryption
- Archive with compression
- Deal easily with remote archiving
- Use Crontab or Celery to setup automated backups.
- Great to keep your development database up to date.

Warning: Django DBBackup version 3 make great changements see Upgrade documentation to help to 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 habit, 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 = 'django.core.files.storage.FileSystemStorage'
DBBACKUP_STORAGE_OPTIONS = {'location': '/my/backup/dir/'}
```

Create the backup directory:

```
mkdir /var/backups
```

Note: 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_TMP_DIR

Directory to be used in local filesystem for temporary files.

Default: `tempfile.gettempdir()`

2.1.3 DBBACKUP_TMP_FILE_MAX_SIZE

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

Default: `10*1024*1024`

2.1.4 DBBACKUP_CLEANUP_KEEP and DBBACKUP_CLEANUP_KEEP_MEDIA

When issueing `dbbackup` and `mediabackup` with `--clean` option, the number of old backup files are looked for and removed.

Default: `10` (backups)

2.1.5 DBBACKUP_CLEANUP_FILTER

A callable that takes a filename (of an old backup, to be cleaned) and returns a boolean indicating whether the backup should be kept (`True`) or deleted (`False`).

Default: `lambda filename: False`

This can be used to keep monthly backups, for example.

2.1.6 DBBACKUP_DATE_FORMAT

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

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

2.1.7 DBBACKUP_HOSTNAME

Used to identify backup by server name in their file name..

Default: `socket.gethostname()`

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, content_type):
    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.

2.2 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!

2.2.1 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. (*GPG manual*)
- Set the setting `DBBACKUP_GPG_RECIPIENT` to the name of the GPG key.

2.2.2 DBBACKUP_GPG_ALWAYS_TRUST

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.

2.2.3 DBBACKUP_GPG_RECIPIENT

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.

2.3 Email configuration

Note: Django 1.6 won't send the full traceback

2.3.1 DBBACKUP_SEND_EMAIL

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

Default: `True`

2.3.2 DBBACKUP_SERVER_EMAIL

The email address that error messages come from, such as those sent to `DBBACKUP_ADMINS`.

Default: `django.conf.settings.SERVER_EMAIL`

2.3.3 DBBACKUP_ADMINS

A list of all the people who get code error notifications. When `DEBUG=False` and an operation raises an exception, DBBackup will email these people with the full exception information. This should be a tuple of (Full name, email address).

Default: `django.conf.settings.ADMINS`

> **Warning:** `DBBACKUP_FAILURE_RECIPIENTS` was used before and is deprecated

2.3.4 DBBACKUP_EMAIL_SUBJECT_PREFIX

Subject-line prefix for email messages sent by DBBackup.

Default: `'[dbbackup] '`

2.4 Database configuration

By default, DBBackup uses parameters from `settings.DATABASES` but you can make an independant configuration, see Database settings

2.5 Storage configuration

You have to use a storage for your backups, see Storage settings for more.

CHAPTER 3

Database settings

The following databases are supported by this application:

- SQLite
- MySQL
- PostgreSQL
- MongoDB
- And the ones you will implement

By default, DBBackup will try to use your database settings in `DATABASES` for handle database, but some databases require custom options so you could want to use different parameters for backup. That's why we included a `DBBACKUP_CONNECTORS` setting; it acts like the `DATABASES` one:

```
DBBACKUP_CONNECTORS = {
    'default': {
        'USER': 'backupuser',
        'PASSWORD': 'backuppassword',
        'HOST': 'replica-for-backup'
    }
}
```

This configuration will allow you to use a replica with a different host and user, which is a great pratice if you don't want to overload your main database.

DBBackup uses `Connectors` for creating and restoring backups; below you'll see specific parameters for the built-in ones.

3.1 Common

All connectors have the following parameters:

3.1.1 CONNECTOR

Absolute path to a connector class by default is:

- `dbbackup.db.sqlite.SqliteConnector` for `'django.db.backends.sqlite3'`
- `dbbackup.db.mysql.MysqlDumpConnector` for `django.db.backends.mysql`
- `dbbackup.db.postgresql.PgDumpConnector` for `django.db.backends.postgresql`
- `dbbackup.db.postgresql.PgDumpGisConnector` for `django.contrib.gis.db.backends.postgis`
- `dbbackup.db.mongodb.MongoDumpConnector` for `django_mongodb_engine`

All supported built-in connectors are listed below.

3.1.2 EXCLUDE

Tables to exclude from backup as list. This option can be unavailable for connectors making snapshots.

3.1.3 EXTENSION

Extension of backup file name, default `'dump'`.

3.2 Command connectors

Some connectors use command line tools as dump engine, `mysqldump` for example. These kinds of tools have common attributes:

3.2.1 DUMP_CMD

Path to the command used to create a backup; default is the appropriate command supposed to be in your PATH, for example: `'mysqldump'` for MySQL.

This setting is useful only for connectors using command line tools (children of `dbbackup.db.base.BaseCommandDBConnector`)

3.2.2 RESTORE_CMD

Same as `DUMP_CMD` but for restoring action.

3.2.3 DUMP_PREFIX and RESTORE_PREFIX

String to include as prefix of dump or restore command. It will be add with a space between launched command and its prefix.

3.2.4 DUMP_SUFFIX and RESTORE_PREFIX

String to include as suffix of dump or restore command. It will be add with a space between launched command and its suffix.

3.2.5 ENV, DUMP_ENV and RESTORE_ENV

Environment variables used during command running, default are `{}`. `ENV` is used for every command, `DUMP_ENV` and `RESTORE_ENV` override the values defined in `ENV` during the dedicated commands.

3.2.6 USE_PARENT_ENV

Specify if the connector will use its parent's environment variables. By default it is `True` to keep `PATH`.

3.3 SQLite

SQLite uses by default `dbbackup.db.sqlite.SqliteConnector`.

3.3.1 SqliteConnector

It is in pure Python and copys the behavior of `.dump` command for creating a SQL dump.

3.3.2 SqliteCPConnector

You can also use `dbbackup.db.sqlite.SqliteCPConnector` for making a simple raw copy of your database file, like a snapshot.

In-memory database aren't dumpable with it.

3.4 MySQL

MySQL uses by default `dbbackup.db.mysql.MysqlDumpConnector`. It uses `mysqldump` and `mysql` for its job.

3.5 PostgreSQL

Postgres uses by default `dbbackup.db.postgres.PgDumpConnector`, but we advise you to use `dbbackup.db.postgres.PgDumpBinaryConnector`. The first one uses `pg_dump` and `pqsl` for its job, creating RAW SQL files.

The second uses `pg_restore` with binary dump files.

They can also use `psql` for launch administration command.

3.5.1 SINGLE_TRANSACTION

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

This corresponds to `--single-transaction` argument of `psql` and `pg_restore`.

Default: `True`

3.5.2 DROP

With `PgDumpConnector`, it includes tables dropping statements in dump file. `PgDumpBinaryConnector` drops at restoring.

This corresponds to `--clean` argument of `pg_dump` and `pg_restore`.

Default: `True`

3.6 PostGIS

Set in `dbbackup.db.postgres.PgDumpGisConnector`, it does the same as PostgreSQL but launchs `CREATE EXTENSION IF NOT EXISTS postgis;` before restore database.

3.6.1 PSQL_CMD

Path to `psql` command used for administration tasks like enable PostGIS for example, default is `psql`.

3.6.2 PASSWORD

If you fill this settings `PGPASSWORD` environment variable will be used with every commands. For security reason, we advise to use `.pgpass` file.

3.6.3 ADMIN_USER

Username used for launch action with privileges, extension creation for example.

3.6.4 ADMIN_PASSWORD

Password used for launch action with privileges, extension creation for example.

3.7 MongoDB

MongoDB uses by default `dbbackup.db.mongodb.MongoDumpConnector`. it uses `mongodump` and `mongorestore` for its job.

3.7.1 OBJECT_CHECK

Validate documents before inserting in database (option `--objcheck` in command line), default is `True`.

3.7.2 DROP

Replace objects that are already in database, (option `--drop` in command line), default is `True`.

3.8 Custom connector

Creating your connector is easy; create a children class from `dbbackup.db.base.BaseDBConnector` and create `_create_dump` and `_restore_dump`. If your connector uses a command line tool, inherit it from `dbbackup.db.base.BaseCommandDBConnector`

3.9 Connecting a Custom connector

Here is an example, on how to easily connect a custom connector that you have created or even that you simply want to reuse:

```
DBBACKUP_CONNECTOR_MAPPING = {
    'transaction_hooks.backends.postgis': 'dbbackup.db.postgresql.PgDumpGisConnector',
}
```

Obviously instead of `dbbackup.db.postgresql.PgDumpGisConnector` you can use the custom connector you have created yourself and `transaction_hooks.backends.postgis` is simply the database engine name you are using.

CHAPTER 4

Storage settings

One of the most helpful feature of django-dbbackup is the avaibility to store and retrieve backups from a local or remote storage. This functionality is mainly based on Django Storage API and extend its possibilities.

You can choose your backup storage backend by set `settings.DBBACKUP_STORAGE`, it must be a full path of a storage class. For example: `django.core.files.storage.FileSystemStorage` 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.

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

DBBackup uses by default the built-in file system storage to manage files on a local directory. Feel free to use any Django storage, you can find a variety of them at Django Packages.

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

4.1 File system storage

4.1.1 Setup

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

```
DBBACKUP_STORAGE = 'django.core.files.storage.FileSystemStorage'
DBBACKUP_STORAGE_OPTIONS = {'location': '/my/backup/dir/'}
```

4.1.2 Available settings

location

Absolute path to the directory that will hold the files.

base_url

URL that serves the files stored at this location.

file_permissions_mode

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

directory_permissions_mode

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

See FileSystemStorage's documentation for a full list of available settings.

4.2 Amazon S3

Our S3 backend uses Django Storages which uses boto.

4.2.1 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 boto3 django-storages
```

Add the following to your project's settings:

```
DBBACKUP_STORAGE = 'storages.backends.s3boto3.S3Boto3Storage'
DBBACKUP_STORAGE_OPTIONS = {
    'access_key': 'my_id',
    'secret_key': 'my_secret',
    'bucket_name': 'my_bucket_name',
    'default_acl': 'private'
}
```

4.2.2 Available settings

Note: More settings are available see official documentation for get more about.

access_key - Required

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

secret_key - Required

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

bucket_name - Required

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

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'`.

use_ssl - Default: `True`

default_acl - Required

This setting can either be `'private'` or `'public'`. Since you want your backups to be secure you'll want to set `'default_scl'` to `'private'`.

location - Optional

If you want to store your backups inside a particular folder in your bucket you need to specify the `'location'`. The folder can be specified as `'folder_name/'`. You can specify a longer path with `'location': 'root_folder/sub_folder/sub_sub_folder/'`.

See Django Storage S3 storage official documentation for more information about available settings.

4.3 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.

4.3.1 Setup your Dropbox account

1. Login to Dropbox and navigate to Developers » MyApps. https://www.dropbox.com/developers/apps
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.

4.3.2 Setup your Django project

```
pip install dropbox django-storages
```

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

```
DBBACKUP_STORAGE = 'storages.backends.dropbox.DropBoxStorage'
DBBACKUP_STORAGE_OPTIONS = {
    'oauth2_access_token': 'my_token',
}
```

4.3.3 Available settings

Note: See django-storages dropbox official documentation for get more details about.

oauth2_access_token - Required

Your OAuth access token

root_path

Jail storage to this directory

4.4 FTP

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

4.4.1 Setup

```
pip install django-storages
```

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.

```
DBBACKUP_STORAGE = 'storages.backends.ftp.FTPStorage
DBBACKUP_STORAGE_OPTIONS = {
    'location': 'ftp://user:pass@server:21'
}
```

4.4.2 Settings

location - Required

A FTP URI with optional user, password and port. example: `'ftp://anonymous@myftp.net'`

base_url

URL that serves with HTTP(S) the files stored at this location.

4.4.3 Setup

We use FTP backend from Django-Storages (again).

```
pip install django-storages
```

Here a simple configuration:

```
DBBACKUP_STORAGE = 'storages.backends.ftp.FTPStorage'
DBBACKUP_STORAGE_OPTIONS = {'location': ftp://myftpserver/}
```

4.5 SFTP

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

4.5.1 Setup

This backend is from Django-Storages with paramiko under.

```
pip install paramiko django-storages
```

The next configuration admit SSH server grant a the local user:

```
DBBACKUP_STORAGE = 'storages.backends.sftpstorage.SFTPStorage'
DBBACKUP_STORAGE_OPTIONS = {'host': 'myserver'}
```

4.5.2 Available settings

host - Required

Hostname or adress of the SSH server

root_path - Default `~/`

Jail storage to this directory

params - Default `{}`

Argument used by method:*paramikor.SSHClient.connect()*. See paramiko SSHClient.connect() documentation for details.

interactive - Default `False`

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 `params`.

file_mode

UID of the account that should be set as owner of the files on the remote.

dir_mode

GID of the group that should be set on the files on the remote host.

known_host_file

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

CHAPTER 5

Commands

The primary usage of DBBackup is made with command line tools. By default, commands will create backups and upload to your defined storage or download and restore the latest.

Commands provide arguments for compress/uncompress and encrypt/decrypt.

5.1 dbbackup

Backup of database.

```
$ ./manage.py dbbackup
Backing Up Database: /tmp/tmp.x0kN9sYSqk
Backup size: 3.3 KiB
Writing file to tmp-zuluvm-2016-07-29-100954.dump
```

5.1.1 Help

usage: manage.py dbbackup [-h] [–noinput] [-q] [-c] [-d DATABASE] [-s SERVERNAME] [-z] [-e] [-o OUTPUT$_FILENAME$][$-OOUTPUT_PATH$][$--version$][$-v0,1,2,3$][$--settingsSETTINGS$][$--pythonpathPYTHONPATH$][$--traceback$][$--no-color$][$--force-color$][$--skip-checks$]

Backup a database, encrypt and/or compress and write to storage.

optional arguments: -h, –help show this help message and exit –noinput Tells Django to NOT prompt the user for input of any kind. -q, –quiet Tells Django to NOT output other text than errors. -c, –clean Clean up old backup files -d DATABASE, –database DATABASE Database(s) to backup specified by key separated by commas(default: all) -s SERVERNAME, –servername SERVERNAME Specify server name to include in backup filename -z, –compress Compress the backup files -e, –encrypt Encrypt the backup files -o OUTPUT$_FILENAME$, $--output$ $filenameOUTPUT_FILENAMESpecifyfilenameonstorage$ $-OOUTPUT_PATH$, $--output$ $pathOUTPUT_PATHSpecifywheretostoreonlocalfilesystem$ $--versionshowprogram'sversionnumberandexit$ $-v0,1,2,3$, $--verbosity0,1,2,3Verbositylevel;0$ $=minimaloutput,1$ $=$ $normaloutput,2$ $=$ $verboseoutput,3$ $=$ $veryverboseoutput$ $-$

$-settingsSETTINGSThePythonpathtoasettingsmodule, e.g."myproject.settings.main".Ifthisisn'tprovided, theDJANG$
$-pythonpathPYTHONPATHAdirectorytoaddtothePythonpath, e.g."/home/djangoprojects/myproject".-$
$-tracebackRaiseonCommandErrorexceptions--no-colorDon'tcolorizethecommandoutput.--force-$
$colorForcecolorizationofthecommandoutput.--skip-checksSkipsystemchecks.$

5.2 dbrestore

Restore a database.

```
Restoring backup for database: /tmp/tmp.x0kN9sYSqk
Finding latest backup
Restoring: tmp-zuluvm-2016-07-29-100954.dump
Restore tempfile created: 3.3 KiB
```

5.2.1 Help

usage: manage.py dbrestore [-h] [–noinput] [-q] [-d DATABASE] [-i INPUT$_FILENAME$][$-IINPUT_PATH$][$-sSERVERNAM$
$version$][$-v0,1,2,3$][$--settingsSETTINGS$][$--pythonpathPYTHONPATH$][$--traceback$][$--no-$
$color$][$--force-color$][$--skip-checks$]

Restore a database backup from storage, encrypted and/or compressed.

optional arguments: -h, –help show this help message and exit –noinput Tells Django to NOT prompt the user for input of any kind. -q, –quiet Tells Django to NOT output other text than errors. -d DATABASE, –database DATABASE Database to restore -i INPUT$_FILENAME$,$--$
$input-filenameINPUT_FILENAMESpecifyfilenametobackupfrom-IINPUT_PATH$,$--$
$input-pathINPUT_PATHSpecifypathonlocalfilesystemtobackupfrom-sSERVERNAME$,$--$
$servernameSERVERNAMEIfbackupfileisnotspecified, filtertheexistingoneswiththegivenservername-$
c,$--decryptDecryptdatabaseforerestoring-pPASSPHRASE$,$--passphrasePASSPHRASEPassphrasefordecryptfile-$
z,$--uncompressUncompressgzipdatabaseforerestoring--versionshowprogram'sversionnumberandexit-$
$v0,1,2,3$,$--verbosity0,1,2,3Verbositylevel;0=minimaloutput,1=normaloutput,2=$
$verboseoutput,3=veryverboseoutput--settingsSETTINGSThePythonpathtoasettingsmodule, e.g."myproject.settings.$
$-pythonpathPYTHONPATHAdirectorytoaddtothePythonpath, e.g."/home/djangoprojects/myproject".-$
$-tracebackRaiseonCommandErrorexceptions--no-colorDon'tcolorizethecommandoutput.--force-$
$colorForcecolorizationofthecommandoutput.--skip-checksSkipsystemchecks.$

5.3 mediabackup

Backup media files, gather all in a tarball and encrypt or compress.

```
$ ./manage.py mediabackup
Backup size: 10.0 KiB
Writing file to zuluvm-2016-07-04-081612.tar
```

5.3.1 Help

usage: manage.py mediabackup [-h] [–noinput] [-q] [-c] [-s SERVERNAME] [-z] [-e] [-o OUTPUT$_FILENAME$][$-OOUTPUT_PATH$][$--version$][$-v0,1,2,3$][$--settingsSETTINGS$][$--$
$pythonpathPYTHONPATH$][$--traceback$][$--no-color$][$--force-color$][$--skip-checks$]

Backup media files, gather all in a tarball and encrypt or compress.

optional arguments: -h, –help show this help message and exit –noinput Tells Django to NOT prompt the user for input of any kind. -q, –quiet Tells Django to NOT output other text than errors. -c, –clean Clean up old backup files -s SERVERNAME, –servername SERVERNAME Specify server name to include in backup filename -z, –compress Compress the archive -e, –encrypt Encrypt the backup files -o OUTPUT$_FILENAME, - - output - filenameOUTPUT_FILENAMESpecifyfilenameonstorage - OOUTPUT_PATH, - - output - pathOUTPUT_PATHSpecifywheretostoreonlocalfilesystem - -versionshowprogram'sversionnumberandexit - v0, 1, 2, 3, - - verbosity0, 1, 2, 3Verbositylevel; 0 = minimaloutput, 1 = normaloutput, 2 = verboseoutput, 3 = veryverboseoutput - -settingsSETTINGSThePythonpathtoasettingsmodule, e.g."myproject.settings.main".Ifthisisn'tprovided, theDJANG -pythonpathPYTHONPATHAdirectorytoaddtothePythonpath, e.g."/home/djangoprojects/myproject".- -tracebackRaiseonCommandErrorexceptions - -no - colorDon'tcolorizethecommandoutput. - -force - colorForcecolorizationofthecommandoutput. - -skip - checksSkipsystemchecks.

5.4 mediarestore

Restore media files, extract files from archive and put into media storage.

```
$ ./manage.py mediarestore
Restoring backup for media files
Finding latest backup
Reading file zuluvm-2016-07-04-082551.tar
Restoring: zuluvm-2016-07-04-082551.tar
Backup size: 10.0 KiB
Are you sure you want to continue? [Y/n]
2 file(s) restored
```

5.4.1 Help

usage: manage.py mediarestore [-h] [–noinput] [-q] [-i INPUT$_FILENAME][-IINPUT_PATH][-sSERVERNAME][-e][-pPA version][-v0, 1, 2, 3][- - settingsSETTINGS][- - pythonpathPYTHONPATH][- - traceback][- - no - color][- - force - color][- - skip - checks]

Restore a media backup from storage, encrypted and/or compressed.

optional arguments: -h, –help show this help message and exit –noinput Tells Django to NOT prompt the user for input of any kind. -q, –quiet Tells Django to NOT output other text than errors. -i INPUT$_FILENAME, - - input - filenameINPUT_FILENAMESpecifyfilenametobackupfrom - IINPUT_PATH, - - input - pathINPUT_PATHSpecifypathonlocalfilesystemtobackupfrom - sSERVERNAME, - - servernameSERVERNAMEIfbackupfileisnotspecified, filtertheexistingoneswiththegivenservername- e, - -decryptDecryptdatabaforerestoring - pPASSPHRASE, - -passphrasePASSPHRASEPassphrasefordecryptfile- z, - - uncompressUncompressgzipdatabaforerestoring - r, - - replaceReplaceexistingfiles - -versionshowprogram'sversionnumberandexit - v0, 1, 2, 3, - - verbosity0, 1, 2, 3Verbositylevel; 0 = minimaloutput, 1 = normaloutput, 2 = verboseoutput, 3 = veryverboseoutput - -settingsSETTINGSThePythonpathtoasettingsmodule, e.g."myproject.settings.main".Ifthisisn'tprovided, theDJANG -pythonpathPYTHONPATHAdirectorytoaddtothePythonpath, e.g."/home/djangoprojects/myproject".- -tracebackRaiseonCommandErrorexceptions - -no - colorDon'tcolorizethecommandoutput. - -force - colorForcecolorizationofthecommandoutput. - -skip - checksSkipsystemchecks.

5.5 listbackups

This command helps to list backups filtered by type (`'media'` or `'db'`), by compression or encryption.

5.5.1 Help

usage: manage.py listbackups [-h] [–noinput] [-q] [-d DATABASE] [-z] [-Z] [-e] [-E] [-c CONTENT$_TYPE$][$--version$][$-v0,1,2,3$][$--settingsSETTINGS$][$--pythonpathPYTHONPATH$][$--traceback$][$--no-color$][$--force-color$][$--skip-checks$]

optional arguments: -h, –help show this help message and exit –noinput Tells Django to NOT prompt the user for input of any kind. -q, –quiet Tells Django to NOT output other text than errors. -d DATABASE, –database DATABASE Filter by database name -z, –compressed Exclude non-compressed -Z, –not-compressed Exclude compressed -e, –encrypted Exclude non-encrypted -E, –not-encrypted Exclude encrypted -c CONTENT$_TYPE$, $--content-typeCONTENT_TYPEFilterbycontenttype'db'or'media'$ $--versionshowprogram'sversionnumberandexit$ $-v0,1,2,3,--verbosity0,1,2,3Verbositylevel;0=minimaloutput,1=normaloutput,2=verboseoutput,3=veryverboseoutput$ $--settingsSETTINGSThePythonpathtoasettingsmodule,e.g."myproject.settings.main".Ifthisisn'tprovided,theDJANG$ $--pythonpathPYTHONPATHAdirectorytoaddtothePythonpath,e.g."/home/djangoprojects/myproject".--tracebackRaiseonCommandErrorexceptions--no-colorDon'tcolorizethecommandoutput.--force-colorForcecolorizationofthecommandoutput.--skip-checksSkipsystemchecks.$

CHAPTER 6

Integration tutorials

Note: If you have a custom and/or interesting way of use DBBackup, do not hesitate to make us a pull request.

6.1 Django-cron

Example of cron job with django-cron with file system storage:

```
import os
from django.core import management
from django.conf import settings
from django_cron import CronJobBase, Schedule


class Backup(CronJobBase):
    RUN_AT_TIMES = ['6:00', '18:00']
    schedule = Schedule(run_at_times=RUN_AT_TIMES)
    code = 'my_app.Backup'

    def do(self):
        management.call_command('dbbackup')
```

CHAPTER 7

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.

7.1 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

7.2 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

Note: We advise you to launch it with Python 2 & 3 before push and try it in Travis. DBBackup uses a lot of file operations, so breaks between Python versions are easy.

7.3 Test environment

We provides tools for helps developers to quickly test and dev on DBBackup. There are 2 majors scripts:

- `runtests.py`: Unit tests launcher and equivalent of `manage.py` in the test project.
- `functional.sh`: Shell script that use `runtests.py` to create a database backup and restore it, the same with media, and test if they are restored.

7.3.1 `runtests.py`

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

```
python runtests.py
```

But if argument are provided, it acts as `manage.py` so you can simply launch other command to test deeply, example:

```
# Enter in Python shell
python runtests.py shell

# Launch a particular test module
python runtests.py test dbbackup.tests.test_utils
```

All tests are stored in `dbbackup.tests`.

7.3.2 `functional.sh`

It tests at the higher level if backup/restore mechanism is alright. It becomes powerful because of the configuration you can give to it. See the next chapter for explanation about it.

7.3.3 Configuration

DBBackup contains a test Django project at `dbbackup.tests` and its `settings` module. This configuration takes care of the following environment variables:

DB_ENGINE - Default: `django.db.backends.sqlite3`

Databank-Engine to use. See in django.db.backends for default backends.

DB_NAME - Default: `:memory:`

Database name. Should be set correctly if an other db is used than sqlite3

DB_USER - Default: `None`

DB Username

DB_PASSWORD - Default: `None`

DB Password

DB_HOST - Default: `None`

DB Host

Why is this more complicated than the earlier solution? For mongodb tests dj-database-url had no ENGINE defined.

MEDIA_ROOT - Default= `tempfile.mkdtemp()`

Django's `MEDIA_ROOT`, useful if you want test media backup from filesystem

STORAGE - Default: `dbbackup.tests.utils.FakeStorage`

Storage used for backups

STORAGE_OPTIONS

Options for instanciate the chosen storage. It must be in format `"key1=foo,key2=bar"` and will be convert into a `dict`.

7.4 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.

Code health is checked with Landscape

CHAPTER 8

Upgrade from 2.5.x

8.1 Settings

The following settings are now useless, you can remove them:

- `DBBACKUP_BACKUP_ENVIRONMENT`: Must be set in `CONNECTORS['dbname']['env']`
- `DBBACKUP_RESTORE_ENVIRONMENT`: Same than `BACKUP_ENVIRONMENT`
- `DBBACKUP_FORCE_ENGINE`
- `DBBACKUP_READ_FILE`
- `DBBACKUP_WRITE_FILE`
- `DBBACKUP_BACKUP_DIRECTORY`: Was used by Filesystem storage, use `location` parameter
- `DBBACKUP_SQLITE_BACKUP_COMMANDS`: Was used by SQLite database, use `CONNECTORS`'s parameters.
- `DBBACKUP_SQLITE_RESTORE_COMMANDS`: Same than `SQLITE_BACKUP_COMMANDS`
- `DBBACKUP_MYSQL_BACKUP_COMMANDS`: Same than `SQLITE_BACKUP_COMMANDS` but for MySQL
- `DBBACKUP_MYSQL_RESTORE_COMMANDS`: Same than `MYSQL_BACKUP_COMMANDS`
- `DBBACKUP_POSTGRESQL_BACKUP_COMMANDS` Same than `MYSQL_BACKUP_COMMANDS` but for PostgreSQL
- `DBBACKUP_POSTGRESQL_RESTORE_COMMANDS`: Same than `DBBACKUP_POSTGRESQL_BACKUP_COMMANDS`: Was used for activate PostGIS, use `PgDumpGisConnector` connector for enable this option
- `DBBACKUP_POSTGRESQL_RESTORE_SINGLE_TRANSACTION`: Must be set in `CONNTECTORS['dbname']['single_transaction']`
- `DBBACKUP_BUILTIN_STORAGE`

8.2 Commands

8.2.1 dbrestore

`--backup-extension` has been removed, DBBackup should automaticaly know the appropriate file.

Listing from this command, `--list`, has been removed in favor of `listbackups` command.

8.2.2 mediabackup

`mediabackup`'s `--no-compress` option has been replaced by `--compress` for keep consistency with other commands.

Now this command can backup remote storage, not only filesystem's `DBBACKUP_BACKUP_DIRECTORY`.

8.2.3 mediarestore

You are now able to restore your media files backups. Unfortunately you'll not be able to restore old backup files.

8.3 Database connector

We made a total refactoring of DBCommands system. It is now easier to use, configure and implement a custom one.

All database configuration for backups are defined in settings `DBBACKUP_CONNECTORS`. By default, the `DATABASES` parameters are used but can be overrided in this new constant.

This dictionnary stores configuration about how backups are made, what is the path of backup command (`/bin/mysqldump`), add suffix or prefix to the command line, environment variable, etc.

The system stay pretty simple and can detect alone how to backup your DB. If it can't just submit us what is your Django DB Engine and we'll try to fix it.

8.3.1 SQLite

Previously backup was made by copy the database file, now you have the choice between make a raw snaphot or make a real SQL dump. It can be useful to exclude tables or don't overwrite data.

If you want to restore your old backups choose `dbbackup.db.sqlite.SqliteCPConnector`.

8.4 Storage engine

All storage engines has been removed from DBBackup except the basic. Now this object will use Django storages as driver.

`settings.DBBACKUP_STORAGE` must now be a full path to a Django storage, for example `'django.core.files.storage.FileSystemStorage'`. `settings.DBBACKUP_STORAGE_OPTIONS` hold its function of gather storage's options.

If you was using a removed storage backend, don't worry, we ensure you'll have a solution by test and write equivalent in Django-Storages.

CHAPTER 9

Compatibility

As we want to ensure a lot of platforms will be able to save data before upgrading, Django-DBBackup supports PyPy, Python 2.7, 3.2 to 3.5 and Django greater than 1.6.

CHAPTER 10

Other Resources

- GitHub repository
- PyPi project
- Read The Docs
- GitHub issues
- GitHub pull requests
- Travis CI
- Coveralls

CHAPTER 11

Indices and tables

- genindex
- modindex
- search