Airtable Local Backup Documentation

Release 0.1a1

Rick Henry

Mar 24, 2023

Contents

CHAPTER 1

Introduction

Airtable Local Backup provides a (somewhat) user friendly way to backup data from Airtable locally. There are a number of useful functions to generate your own custom backup scripts or you can use the provided `Runner` class and a yaml-style configuration file to orchestrate the backup.

1.1 Table Of Contents

1.1.1 Installation

You will need Python 3.6 or later installed, as well as pip.

For now, clone the repo and run

```
$ pip install -r requirements.txt

$ python3 setup.py install
```

from inside it. You may want to do it in a virtual environment.

1.1.2 Configuration

Getting Started

A great starting point is the example configuration file.

What You'll Need

- Your Airtable base key (If you go to the api documentation for your base, it's in the url `https://api.airtable.com/v0/YOUR_BASE_KEY`. It is usually app then an alphanumeric string)

- Your Airtable API key (This would be on your account page after enabling api access. (It is preferable to store this in the environment variable `AIRTABLE_API_KEY` rather than in the config file if possible). You may want to read more about the Airtable API.
- Somewhere to put the backups. This can be a local directory, S3 or S3 compatible service, or any builtin in filesystem of pyfilesystem2. If pyfilesystem2 has an extension for the filesystem you want to use, it should work but you will need to install the extension separately.

Configuration File

There are several configuration options available:

- `Base Name:` This is just a reference name for the database, something you'll recognize.
- `Airtable Base Key:` *Required* (see above). This begins 'app' and then a bunch of numbers and letters.
- `Airtable API Key:` *Required* This will be stored in plain text, so it is better to save it in an environment variable ($AIRTABLE_API_KEY) and leave this `null`.
- `Store As:` *Required* This is the configuration for storing the downloaded data.
  - `Type:` *Required* Store the files as a `tar` archive, `zip` file, or just as `files`
  - `Compression:` This is available only with tar. Choose your favorite of `xz`, `bz2` , or `gz`.
  - `Path:` Where to put files on the system temporarily before writing out to the `Backing Store`.
- `Backing Store:` Where to put the backups. See sample config for more details.
  - `Prefix:` Something to prepend to the backup files
  - `Date:` (boolean) Whether to include the date on backup files
- `Tables:` This is the most important part of the config. Because of the nature of the Airtable API, you must specify each table you want to back up and keep this up-to-date.
  - `- Name:` Each table requires at minimum the name to be specified (precisely, case sensitive)
  - `Fields:` You can specify the fields here for easier recovery (once implemented)
- `Attachments:` Currently attachments are downloaded, base64 encoded, and embedded into the json backup files. Other solutions may be possible in the future
  - `Discard:` Set this to true to remove the attachments from the backups.
  - `Compress:` Whether to compress the attachments. Compression somewhat slows down backups, but can save quite a bit of disk space.

1.1.3 Usage

Creating Backups

Create a simple python script, for instance `backup.py` with the contents:

```
from airtable_local_backup import Runner

run = Runner(path='/path/to/config/file.yaml')
run.backup()
```

Configuration is discussed more below and you can download an example configuration file.

Restoring from Backups

Not Implemented yet

1.1.4 Api Documentation

This section contains more granular documentation of the classes and functions available to the api.

Automated Runner

class Runner(*path*, *, *filesystem=None*)

This class handles orchestration of downloading and storing the backup. Options are set in a yaml configuration file. There is an `example` you can use as a starting point.

Parameters

- **path** – (required) absolute path to the file on the system or relative to the FS object supplied in the filesystem parameter
- **filesystem** – (keyword only) a pyfilesystem2 FS object where the yaml config file is located.

Useful Methods/Classes

Download Table

class DownloadTable(*base_key*, *table_name*, *api_key=None*, *progress=False*, *compression=True*, *fields={}*, *discard_attach=False*)

Downloads all data from a table including atachments.

Parameters

- **base_key** – base id from airtable api url (starts 'app')
- **table_name** – the table name to download
- **api_key** – the airtable api key. If an environment variable 'AIRTABLE_API_KEY' is set this is not required.
- **compression** – whether to compress attachment data
- **fields** – Store the field
- **discard_attach** – if true, attachment data will not be downloaded, url and other info will be preservered

download()

Download the data in the table.

Returns A generator that will download each item in the table as it is iterated based on the options configured.

File I/O

join_files(*tmpfs*, *outfs*)

Join the backup json files into a single package (tarball, zip).

Parameters

- **tmpfs** – the temporary fs where the backup is stored.
- **outfs** – the filesystem to copy to (should be *TarFS* or *ZipFS*). things like compression and encoding should be specified at instantiation.

write_out_backup(*backing_store_fs*, ***, *filepath=None*, *filesystem=None*, *prefix=''*)

Write the backup data to its final location. A backing store is required and either a filepath to the packaged backup or the tmp filesystem is required.

Parameters

- **backing_store_fs** (*required*) – a pyfilesystem2 object to be the final storage location of the backup. (should be *OSFS*, *S3FS*, *FTPFS*, etc.) Can be a single object or list of filesystem objects for copying to multiple backing stores.
- **filepath** – path to the zip or tar file containing the backup data (if desired). Can be a path object or str.
- **filesystem** – the *TmpFS* containing the backup data.
- **prefix** – a parent directory for the files to be saved under. This is can be a good place to encode some information about the backup. A slash will be appended to the prefix to create a directory or pseudo-directory structure.

write_to_file(*downloadtable*, *tmpfs*, *prefix=''*, *suffix=''*)

Write out the table data to a file.

Parameters

- **downloadtable** – A *download.DownloadTable* object for the table to be saved
- **tmpfs** – the temporary filesystem (from pyfilesystem2) to write the file to.
- **prefix** – A prefix for a the file name. include a / for directories
- **suffix** – A suffix to append to the file name

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index