Flask-Alembic

Release 2.0.1.dev20170309

Mar 09, 2017

Contents

Flask-Alembic is a Flask extension that provides a configurable Alembic migration environment around a Flask-SQLAlchemy database.

CHAPTER 1

Installation

Releases are available from the PyPI project page. Install with pip:

```
pip install Flask-Alembic
```

The latest code is hosted on BitBucket. Install with pip:

```
pip install https://bitbucket.org/davidism/flask-alembic/get/tip.zip
```

CHAPTER 2

Configuration

Configuration for Alembic and its migrations is pulled from the following Flask config keys.

- `ALEMBIC` is a dictionary containing general configuration, mostly used by `alembic.config.Config` and `alembic.script.ScriptDirectory`. See Alembic's docs on config.
- `ALEMBIC_CONTEXT` is a dictionary containing options passed to `alembic.environment.EnvironmentContext` and `alembic.migration.MigrationContext`. See Alembic's docs on context.

The only required configuration is `ALEMBIC['script_location']`, which is the location of the migrations directory. If it is not an absolute path, it will be relative to the instance folder. This defaults to `migrations` relative to the application root.

CHAPTER 3

Basic Usage

First, set up your Flask app (or app factory) and the Flask-SQLAlchemy extension and models.

This extension follows the common pattern of Flask extension setup. Either immediately pass an app to *Alembic*, or call *init_app()* later.

```
from flask_alembic import Alembic

alembic = Alembic()
alembic.init_app(app)  # call in the app factory if you're using that pattern
```

When an app is registered, *mkdir()* is called to set up the migrations directory if it does not already exist. To prevent this, call `init_app` with `run_mkdir=False`.

The `alembic` instance provides an interface between the current app and Alembic. It exposes similar commands to the command line available from Alembic, but Flask-Alembic's methods return data rather than produce output. You can use this interface to do what the command line commands do, from inside your app.

```
# generate a new revision
# same as flask db revision 'made changes'
alembic.revision('made changes')

# run all available upgrades
# same as ./manage.py db upgrade
alembic.upgrade()
```

You can also get at the Alembic internals that enable these commands. See the Alembic API docs for more information.

```
alembic.script.get_revision('head')  # locate a revision by name
alembic.context.get_current_revision()  # could compare this to the 'head' revision␣
↪above to see if upgrades are needed
alembic.op.drop_column('my_table', 'my_column')  # probably don't want to do this␣
↪outside a revision, but it'll work
alembic.compare_metadata()  # see that the column you just dropped will be added back␣
↪next revision
```

CHAPTER 4

Independent Named Branches

Alembic supports named branches, but its syntax is hard to remember and verbose. Flask-Alembic makes it easier by providing a central configuration for branch names and revision directories and simplifying the syntax to the `revision` command.

Alembic allows configuration of multiple version locations. `version_locations` is a list of directories to search for migration scripts. Flask-Alembic extends this to allow tuples as well as strings in the list. If a tuple is added, it specifies a `(branch, directory)` pair. The `script_location` is automatically given the label `default` and added to the `version_locations`.

```
ALEMBIC = {
    'version_locations': [
        'other_migrations',  # not a branch, just another search location
        ('posts', '/path/to/posts_extension/migrations'),  # posts branch migrations␣
↪will be placed here
        ('users', 'users/migrations'),  # relative paths are relative to the␣
↪application root
    ]
}
```

The `revision` command takes a `--branch` option (defaults to `default`). This takes the place of specifying `--parent`, `--label`, and `--path`. This will automatically start the branch from the base revision, label the revision correctly, and place the revisions in the correct location.

```
flask db revision --branch users 'create user model'
# equivalent to (if branch is new)
# alembic revision --autogenerate --head base --branch-label users --version-path␣
↪users/migrations -m 'create user model'
# or (if branch exists)
# alembic revision --autogenerate --head users@head -m 'create user model'
```

CHAPTER 5

Command Line

Currently, Click and Flask-Script are supported. The commands are the same for either one.

If you are using Flask 0.11, the preferred interface is Click. If you are using Flask 0.10, you can use backported integration via Flask-CLI. Flask-Alembic will automatically add the command to the cli if it detects that Click is available.

```
flask db --help
```

If you have set up a `flask_script.Manager` for your project using Flask-Script, you can add Alembic commands like this:

```
from flask_alembic import alembic_script
app_manager.add_command('db', alembic_manager)
```

```
python manage.py db
```

CHAPTER 6

Differences from Alembic

Flask-Alembic is opinionated and was designed to enable specific workflows. If you're looking for a more direct wrapper around Alembic, you may be interested in Flask-Migrate instead.

- Configuration is taken from `Flask.config` instead of `alembic.ini` and `env.py`.
- The migrations are stored directly in the migrations folder instead of the versions folder.
- Provides the migration environment instead of `env.py` and exposes Alembic's internal API objects.
- Differentiates between CLI commands and Python functions. The functions return revision objects and don't print to stdout.
- Allows operating Alembic at any API level while the app is running, through the exposed objects and functions.
- Does not (currently) support offline migrations. I don't plan to add this but am open to patches.
- Does not (currently) support multiple databases. I plan on adding support for Flask-SQLAlchemy binds and external bases eventually, or am open to patches.
- Adds a system for managing independent migration branches and makes it easier to work with named branches.

CHAPTER 7

API Reference

flask_alembic

class `flask_alembic.`**`Alembic`**(*app=None*, *run_mkdir=True*, *command_name='db'*)
Provide an Alembic environment and migration API.

If instantiated without an app instance, *`init_app()`* is used to register an app at a later time.

Configures basic logging to `stderr` for the `sqlalchemy` and `alembic` loggers if they do not already have handlers.

Parameters

- **`app`** – call `init_app` on this app
- **`run_mkdir`** – whether to run *`mkdir()`* during `init_app`
- **`command_name`** – register a Click command with this name during `init_app`, or skip if `False`

`init_app`(*app*, *run_mkdir=None*, *command_name=None*)
Register this extension on an app. Will automatically set up migration directory by default.

Keyword arguments on this method override those set during `__init__()` if not `None`.

Parameters

- **`app`** – app to register
- **`run_mkdir`** – whether to run *`mkdir()`*
- **`command_name`** – register a Click command with this name, or skip if `False`

`rev_id`()
Generate a unique id for a revision.

By default this uses `alembic.util.rev_id()`. Override this method, or assign a static method, to change this.

For example, to use the current timestamp:

```
alembic = Alembic(app)
alembic.rev_id = lambda: datetime.utcnow().timestamp()
```

config
Get the Alembic `Config` for the current app.

script_directory
Get the Alembic `ScriptDirectory` for the current app.

environment_context
Get the Alembic `EnvironmentContext` for the current app.

migration_context
Get the Alembic `MigrationContext` for the current app.

Accessing this property opens a database connection but can't close it automatically. Make sure to call `migration_context.connection.close()` when you're done.

op
Get the Alembic `Operations` context for the current app.

Accessing this property opens a database connection but can't close it automatically. Make sure to call `migration_context.connection.close()` when you're done.

run_migrations(*fn*, ***kwargs*)
Configure an Alembic `MigrationContext` to run migrations for the given function.

This takes the place of Alembic's env.py file, specifically the `run_migrations_online` function.

Parameters

- **fn** – use this function to control what migrations are run
- **kwargs** – extra arguments passed to `upgrade` or `downgrade` in each revision

mkdir()
Create the script directory and template.

current()
Get the list of current revisions.

heads(*resolve_dependencies=False*)
Get the list of revisions that have no child revisions.

Parameters **resolve_dependencies** – treat dependencies as down revisions

branches()
Get the list of revisions that have more than one next revision.

log(*start='base'*, *end='heads'*)
Get the list of revisions in the order they will run.

Parameters

- **start** – only get since this revision
- **end** – only get until this revision

stamp(*target='heads'*)
Set the current database revision without running migrations.

Parameters **target** – revision to set to, default 'heads'

upgrade(*target='heads'*)
Run migrations to upgrade database.

Parameters **target** – revision to go to, default 'heads'

downgrade(*target=-1*)
Run migrations to downgrade database.

Parameters **target** – revision to go down to, default -1

revision(*message*, *empty=False*, *branch='default'*, *parent='head'*, *splice=False*, *depend=None*, *label=None*, *path=None*)
Create a new revision. By default, auto-generate operations by comparing models and database.

Parameters

- **message** – description of revision
- **empty** – don't auto-generate operations
- **branch** – use this independent branch name
- **parent** – parent revision(s) of this revision
- **splice** – allow non-head parent revision
- **depend** – revision(s) this revision depends on
- **label** – label(s) to apply to this revision
- **path** – where to store this revision

Returns list of new revisions

merge(*revisions='heads'*, *message=None*, *label=None*)
Create a merge revision.

Parameters

- **revisions** – revisions to merge
- **message** – description of merge, will default to revisions param
- **label** – label(s) to apply to this revision

Returns new revision

produce_migrations()
Generate the `MigrationScript` object that would generate a new revision.

compare_metadata()
Generate a list of operations that would be present in a new revision.

flask_alembic.cli

Provide integration with cli libraries. Integration with Click:

```
from flask_alembic import alembic_click
app.cli.add_command(alembic_click, 'db')
```

This is done automatically by Flask-Alembic if it detects `app.cli`.

Integration with Flask-Script:

```
from flask_alembic import alembic_script
manager.add_command('db', alembic_script)
```

Base functions that write information to the terminal.

flask_alembic.cli.base.**mkdir**()
Create the migration directory if it does not exist.

flask_alembic.cli.base.**current**(*verbose=False*)
Show the list of current revisions.

flask_alembic.cli.base.**heads**(*resolve_dependencies=False*, *verbose=False*)
Show the list of revisions that have no child revisions.

flask_alembic.cli.base.**branches**(*verbose=False*)
Show the list of revisions that have more than one next revision.

flask_alembic.cli.base.**log**(*start='base'*, *end='heads'*, *verbose=False*)
Show the list of revisions in the order they will run.

flask_alembic.cli.base.**show**(*revisions*)
Show the given revisions.

flask_alembic.cli.base.**stamp**(*target='heads'*)
Set the current revision without running migrations.

flask_alembic.cli.base.**upgrade**(*target='heads'*)
Run migrations to upgrade the database.

flask_alembic.cli.base.**downgrade**(*target=-1*)
Run migration to downgrade the database.

flask_alembic.cli.base.**revision**(*message*, *empty=False*, *branch='default'*, *parent='head'*, *splice=False*, *depend=None*, *label=None*, *path=None*)
Generate a new revision.

flask_alembic.cli.base.**merge**(*revisions*, *message=None*, *label=None*)
Generate a merge revision.

Python Module Index

f

Index