Flask-Migrate Documentation

Miguel Grinberg

Jan 12, 2025

CONTENTS

Flask-Migrate is an extension that handles SQLAlchemy database migrations for Flask applications using Alembic. The database operations are made available through the Flask command-line interface.

CHAPTER

ONE

WHY USE FLASK-MIGRATE VS. ALEMBIC DIRECTLY?

Flask-Migrate is an extension that configures Alembic in the proper way to work with your Flask and Flask-SQLAlchemy application. In terms of the actual database migrations, everything is handled by Alembic so you get exactly the same functionality.

CHAPTER

TWO

INSTALLATION

Install Flask-Migrate with *pip*:

```
pip install Flask-Migrate
```

CHAPTER

THREE

EXAMPLE

This is an example application that handles database migrations through Flask-Migrate:

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from flask_migrate import Migrate

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///app.db'

db = SQLAlchemy(app)
migrate = Migrate(app, db)

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(128))
```

With the above application you can create a migration repository with the following command:

```
$ flask db init
```

This will add a *migrations* folder to your application. The contents of this folder need to be added to version control along with your other source files.

You can then generate an initial migration:

```
$ flask db migrate -m "Initial migration."
```

The migration script needs to be reviewed and edited, as Alembic is not always able to detect every change you make to your models. In particular, Alembic is currently unable to detect table name changes, column name changes, or anonymously named constraints. A detailed summary of limitations can be found in the Alembic autogenerate documentation. Once finalized, the migration script also needs to be added to version control.

Then you can apply the changes described by the migration script to your database:

```
$ flask db upgrade
```

Each time the database models change, repeat the `migrate` and `upgrade` commands.

To sync the database in another system just refresh the *migrations* folder from source control and run the `upgrade` command.

To see all the commands that are available run this command:

```
$ flask db --help
```

Note that the application script must be set in the `FLASK_APP` environment variable for all the above commands to work, as required by the `flask` command.

If the `db` command group name is inconvenient, it can be changed to a different with the `command` argument passed to the `Migrate` class:

```
migrate = Migrate(app, db, command='migrate')
```

CHAPTER

FOUR

ALEMBIC CONFIGURATION OPTIONS

Starting with version 4.0, Flask-Migrate automatically enables the following options that are disabled by default in Alembic:

- `compare_type=True`: This option configures the automatic migration generation subsystem to detect column type changes.
- `render_as_batch=True`: This option generates migration scripts using batch mode, an operational mode that works around limitations of many `ALTER` commands in the SQLite database by implementing a "move and copy" workflow. Enabling this mode should make no difference when working with other databases.

To manually configure these or other Alembic options, pass them as keyword arguments to the `Migrate` constructor. Example:

```
migrate = Migrate(app, db, render_as_batch=False)
```

CHAPTER

FIVE

CONFIGURATION CALLBACKS

Sometimes applications need to dynamically insert their own settings into the Alembic configuration. A function decorated with the `configure` callback will be invoked after the configuration is read, and before it is applied. The function can modify the configuration object, or replace it with a different one.

```
@migrate.configure
def configure_alembic(config):
    # modify config object
    return config
```

Multiple configuration callbacks can be defined simply by decorating multiple functions. The order in which multiple callbacks are invoked is undetermined.

CHAPTER

SIX

MULTIPLE DATABASE SUPPORT

Flask-Migrate can integrate with the binds feature of Flask-SQLAlchemy, making it possible to track migrations to multiple databases associated with an application.

To create a multiple database migration repository, add the `--multidb` argument to the `init` command:

```
$ flask db init --multidb
```

With this command, the migration repository will be set up to track migrations on your main database, and on any additional databases defined in the `SQLALCHEMY_BINDS` configuration option.

CHAPTER

SEVEN

COMMAND REFERENCE

Flask-Migrate exposes one class called `Migrate`. This class contains all the functionality of the extension.

The following example initializes the extension with the standard Flask command-line interface:

```
from flask_migrate import Migrate
migrate = Migrate(app, db)
```

The two arguments to `Migrate` are the application instance and the Flask-SQLAlchemy database instance. The `Migrate` constructor also takes additional keyword arguments, which are passed to Alembic's `EnvironmentContext.configure()` method. As is standard for all Flask extensions, Flask-Migrate can be initialized using the `init_app` method as well:

```
from flask_sqlalchemy import SQLAlchemy
from flask_migrate import Migrate

db = SQLAlchemy()
migrate = Migrate()

def create_app():
    """Application-factory pattern"""
    ...
    ...
    db.init_app(app)
    migrate.init_app(app, db)
    ...
    ...
    return app
```

After the extension is initialized, a `db` group will be added to the command-line options with several sub-commands. Below is a list of the available sub-commands:

- **`flask db --help`**
  Shows a list of available commands.
- **`flask db list-templates`**
  Shows a list of available database repository templates.
- **`flask db init [--multidb] [--template TEMPLATE] [--package]`**
  Initializes migration support for the application. The optional `--multidb` enables migrations for multiple databases configured as Flask-SQLAlchemy binds. The `--template` option allows you to explicitly select a database repository template, either from the stock templates provided by this package, or a custom one, given as a path to the template directory. The `--package` option tells Alembic to add `__init__.py` files in the migrations and versions directories.

- **flask db revision [--message MESSAGE] [--autogenerate] [--sql] [--head HEAD] [--splice] [--branch-label BRANCH_LABEL] [--version-path VERSION_PATH] [--rev-id REV_ID]**
  Creates an empty revision script. The script needs to be edited manually with the upgrade and downgrade changes. See Alembic's documentation for instructions on how to write migration scripts. An optional migration message can be included.
- **flask db migrate [--message MESSAGE] [--sql] [--head HEAD] [--splice] [--branch-label BRANCH_LABEL] [--version-path VERSION_PATH] [--rev-id REV_ID]**
  Equivalent to `revision --autogenerate`. The migration script is populated with changes detected automatically. The generated script should to be reviewed and edited as not all types of changes can be detected automatically. This command does not make any changes to the database, just creates the revision script.
- **flask db check**
  Checks that a `migrate` command would not generate any changes. If pending changes are detected, the command exits with a non-zero status code.
- **flask db edit <revision>**
  Edit a revision script using $EDITOR.
- **flask db upgrade [--sql] [--tag TAG] <revision>**
  Upgrades the database. If `revision` isn't given then `"head"` is assumed.
- **flask db downgrade [--sql] [--tag TAG] <revision>**
  Downgrades the database. If `revision` isn't given then `-1` is assumed.
- **flask db stamp [--sql] [--tag TAG] <revision>**
  Sets the revision in the database to the one given as an argument, without performing any migrations.
- **flask db current [--verbose]**
  Shows the current revision of the database.
- **flask db history [--rev-range REV_RANGE] [--verbose]**
  Shows the list of migrations. If a range isn't given then the entire history is shown.
- **flask db show <revision>**
  Show the revision denoted by the given symbol.
- **flask db merge [--message MESSAGE] [--branch-label BRANCH_LABEL] [--rev-id REV_ID] <revisions>**
  Merge two revisions together. Creates a new revision file.
- **flask db heads [--verbose] [--resolve-dependencies]**
  Show current available heads in the revision script directory.
- **flask db branches [--verbose]**
  Show current branch points.

Notes:

- All commands take one or more `--x-arg ARG=VALUE` or `-x ARG=VALUE` options with custom arguments that can be used in `env.py`.
- All commands take a `--directory DIRECTORY` option that points to the directory containing the migration scripts. If this argument is omitted the directory used is `migrations`.
- A directory can also be specified as a `directory` argument to the `Migrate` constructor, or in the `FLASK_DB_DIRECTORY` environment variable.
- The `--sql` option present in several commands performs an 'offline' mode migration. Instead of executing the database commands the SQL statements that need to be executed are printed to the console.
- Detailed documentation on these commands can be found in the Alembic's command reference page.

CHAPTER

EIGHT

API REFERENCE

The commands exposed by Flask-Migrate's command-line interface can also be accessed programmatically by importing the functions from module `flask_migrate`. The available functions are:

- **init(directory='migrations', multidb=False)**
  Initializes migration support for the application.
- **revision(directory='migrations', message=None, autogenerate=False, sql=False, head='head', splice=False, branch_label=None, version_path=None, rev_id=None)**
  Creates an empty revision script.
- **migrate(directory='migrations', message=None, sql=False, head='head', splice=False, branch_label=None, version_path=None, rev_id=None)**
  Creates an automatic revision script.
- **edit(directory='migrations', revision='head')**
  Edit revision script(s) using $EDITOR.
- **merge(directory='migrations', revisions='', message=None, branch_label=None, rev_id=None)**
  Merge two revisions together. Creates a new migration file.
- **upgrade(directory='migrations', revision='head', sql=False, tag=None)**
  Upgrades the database.
- **downgrade(directory='migrations', revision='-1', sql=False, tag=None)**
  Downgrades the database.
- **show(directory='migrations', revision='head')**
  Show the revision denoted by the given symbol.
- **history(directory='migrations', rev_range=None, verbose=False)**
  Shows the list of migrations. If a range isn't given then the entire history is shown.
- **heads(directory='migrations', verbose=False, resolve_dependencies=False)**
  Show current available heads in the script directory.
- **branches(directory='migrations', verbose=False)**
  Show current branch points
- **current(directory='migrations', verbose=False, head_only=False)**
  Shows the current revision of the database.
- **stamp(directory='migrations', revision='head', sql=False, tag=None)**
  Sets the revision in the database to the one given as an argument, without performing any migrations.

Notes:

- These commands will invoke the same functionality that runs from the command-line, including output to the terminal. The logging configuration of the process will be overridden by Alembic according to the contents of the alembic.ini file.
- For greater scripting flexibility you can also use the API exposed by Alembic directly.