Seamless Integration: Building Applications That Leverage Airflow's Database Migration Framework

ASTRONOMER

Ephraim Anierobi

Senior Software Engineer @ astronomer.io

Committer

PMC Member

ASTRONOMER

Airflow metadata Database

Backbone:

- **SQLAlchemy (models)**
- **Alembic (migrations)**

Targets:

- **Postgres**
- **MySQL**
- (SQLite for dev/test)

Airflow DB migrations: where • how

Location: `airflow/migrations/versions/`

State table: `alembic_version` (tracks current revision)

Use cases: first-run init, schema changes across upgrades/downgrades

```
# migrate to latest
airflow db migrate
# migrate to a version
airflow db migrate -n 3.1.0
# check migrations done
airflow db check-migrations
# rollback example
airflow db downgrade -n 2.7.0
# reset
airflow db reset
```

Airflow 2.x: What airflow db commands Actually Migrate

In scope

- Core Airflow metadata schema (Alembic migrations in core)

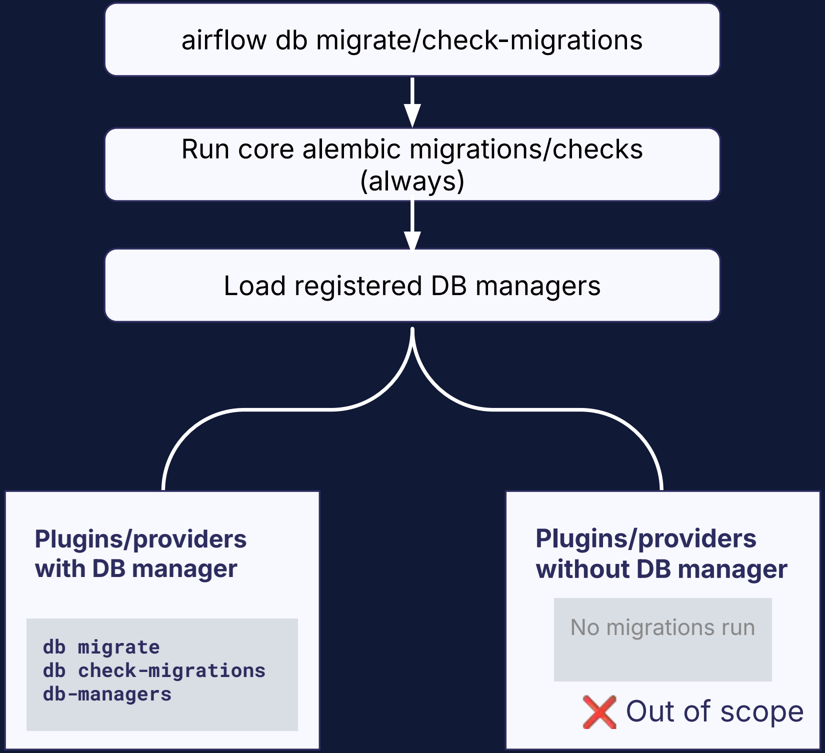
Out of scope

- Provider/plugin-defined tables
- Non-core schemas must ship/run their own migrations (no auto-discovery)

Implication

- **Startup wait for migrations, applies only to core**. Airflow components wait for core DB migrations, not your provider's/plugin's migrations.

Airflow 3: When do airflow db commands run non-core migrations?


airflow db migrate/check-migrations
Run core alembic migrations/checks (always)
Load registered DB managers
Plugins/providers with DB manager
Plugins/providers without DB manager
No migrations run
❌ Out of scope

```
db migrate
db check-migrations
db-managers
```

✅ **Scope gate; only DB managers registered are invoked; otherwise nothing happens; core only**

ASTRONOMER

When to add a DB Manager (Provider/Plugin)

Add a DB Manager if...

- Your provider/plugin owns tables (or modifies its own schema) inside the Airflow metadata DB.
- You want airflow to wait for your migrations to be done via airflow db check-migrations (and invoke them during db migrate).

You don't need it if...

- You don't create tables (use only core models).
- Your state lives in an **external service DB** (outside Airflow's metadata DB).

Decision rule

- If you own schema in Airflow's metadata DB → define a DB Manager. Otherwise → skip it.

How do we integrate providers/plugins migrations in Airflow?

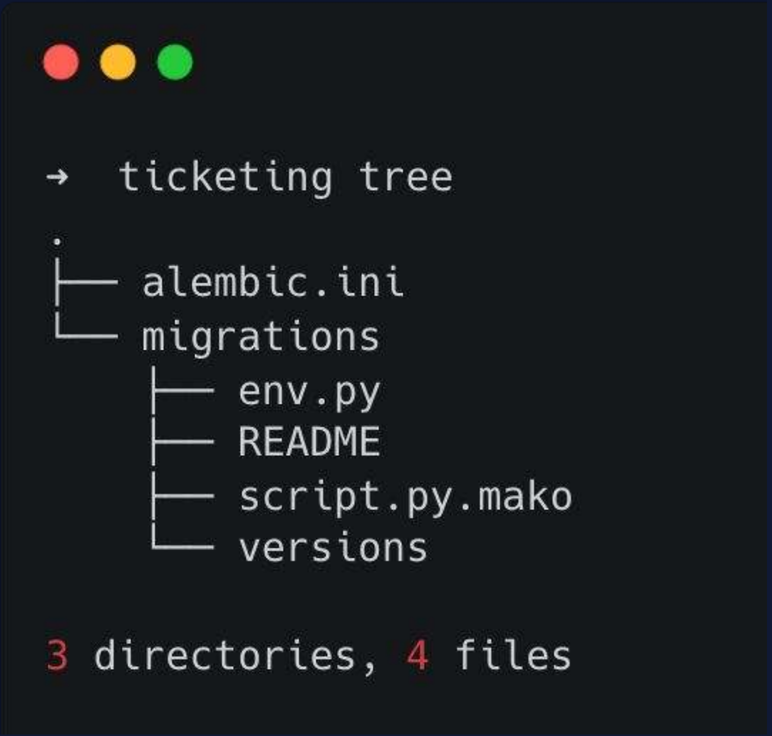
Demo app for this talk

A simple plugin including a listener plugin that mocks creating a ticket for every failed dag and logs these tickets with their URL in a database table (that's why we need a custom db manager!).

Github:
https://github.com/ephraimbuddy/ticketing

Scan the QR code or use this link to open it on any device

```
alembic init migrations
```



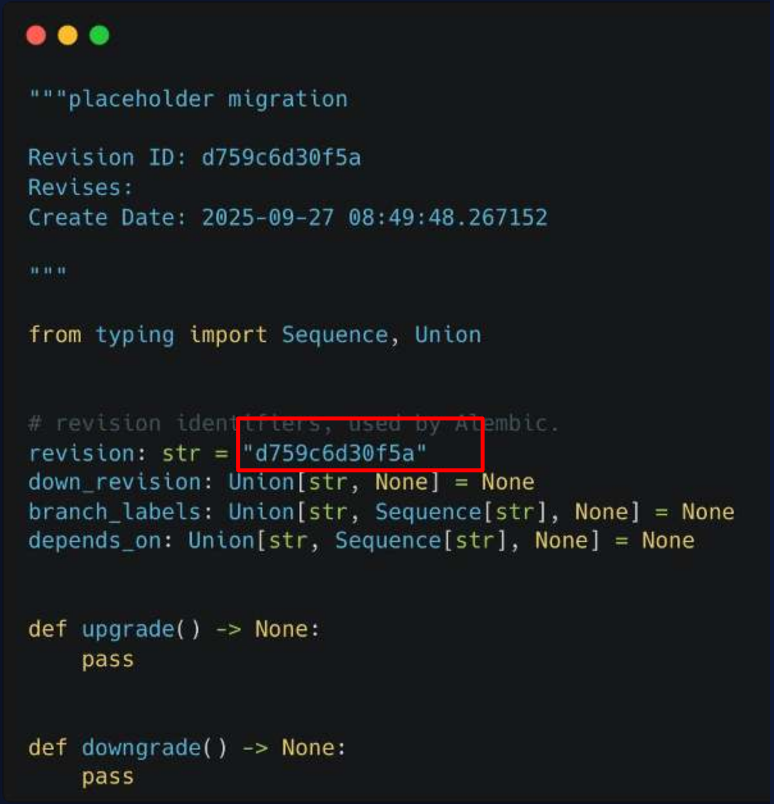
```
➜  ticketing tree
.
├── alembic.ini
└── migrations
    ├── env.py
    ├── README
    ├── script.py.mako
    └── versions

3 directories, 4 files
```

Replace the new alembic.ini with Airflow's alembic.ini file

*Airflow's alembic.ini file is located at airflow-core/src/airflow/alembic.ini

```
alembic revision -m “Placeholder migration”
```

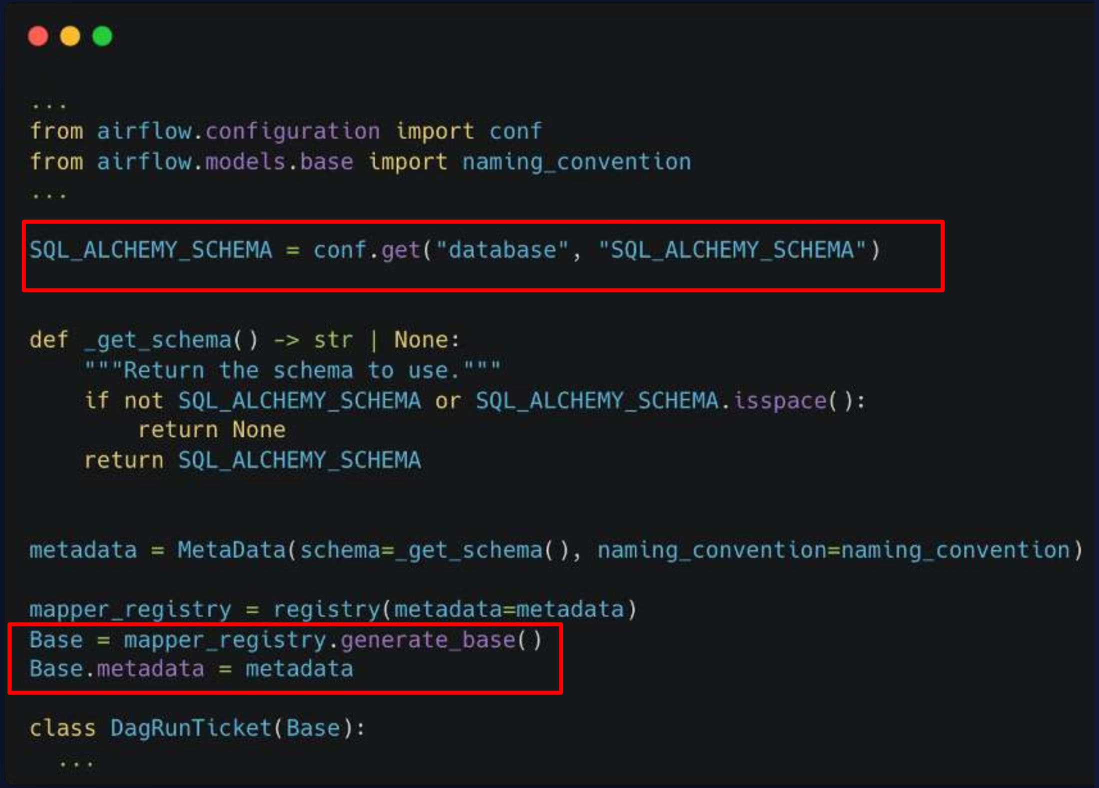
ticketing/migrations/versions/d759c6d30f5a_placeholder_migration.py



```python
"""placeholder migration

Revision ID: d759c6d30f5a
Revises:
Create Date: 2025-09-27 08:49:48.267152

"""

from typing import Sequence, Union


# revision identifiers, used by Alembic.
revision: str = "d759c6d30f5a"
down_revision: Union[str, None] = None
branch_labels: Union[str, Sequence[str], None] = None
depends_on: Union[str, Sequence[str], None] = None


def upgrade() -> None:
    pass


def downgrade() -> None:
    pass
```

Use Airflow's schema and naming conventions

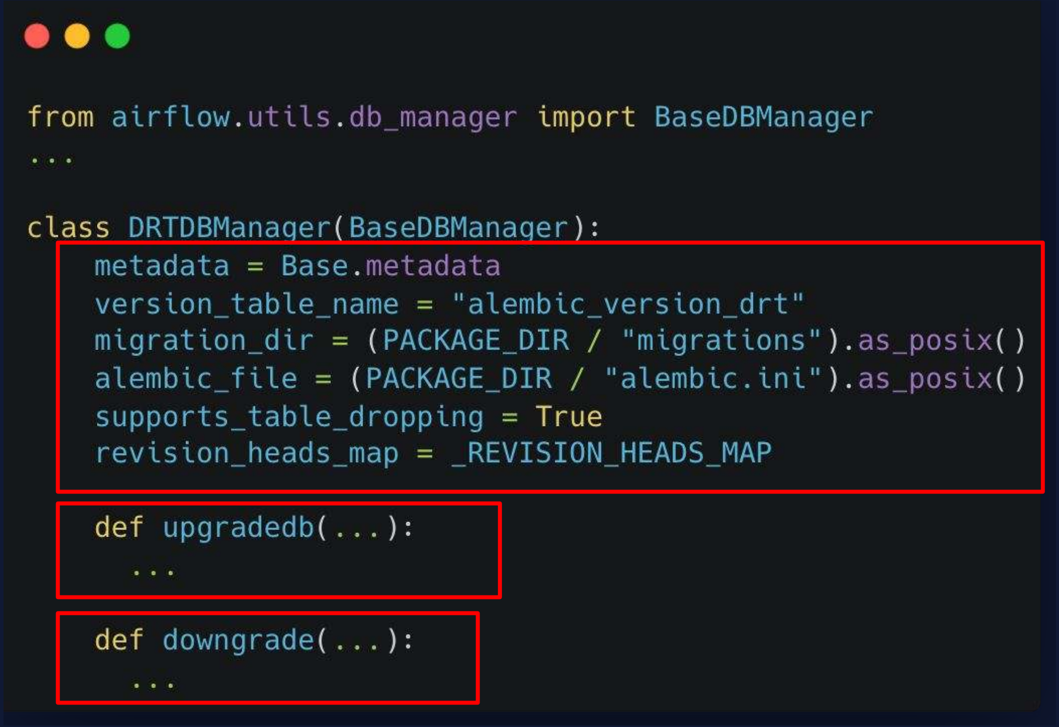
ticketing/models.py



```
...
from airflow.configuration import conf
from airflow.models.base import naming_convention
...

SQL_ALCHEMY_SCHEMA = conf.get("database", "SQL_ALCHEMY_SCHEMA")


def _get_schema() -> str | None:
    """Return the schema to use."""
    if not SQL_ALCHEMY_SCHEMA or SQL_ALCHEMY_SCHEMA.isspace():
        return None
    return SQL_ALCHEMY_SCHEMA


metadata = MetaData(schema=_get_schema(), naming_convention=naming_convention)

mapper_registry = registry(metadata=metadata)
Base = mapper_registry.generate_base()
Base.metadata = metadata

class DagRunTicket(Base):
  ...
```

Implement a custom DB manager

ticketing/db_manager.py

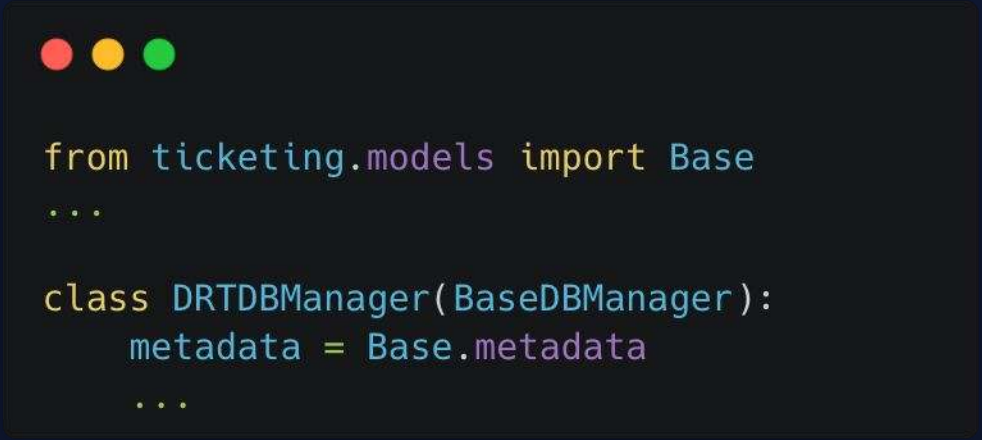


```python
from airflow.utils.db_manager import BaseDBManager
...

class DRTDBManager(BaseDBManager):
    metadata = Base.metadata
    version_table_name = "alembic_version_drt"
    migration_dir = (PACKAGE_DIR / "migrations").as_posix()
    alembic_file = (PACKAGE_DIR / "alembic.ini").as_posix()
    supports_table_dropping = True
    revision_heads_map = _REVISION_HEADS_MAP

    def upgradedb(...):
      ...

    def downgrade(...):
      ...
```

ASTRONOMER

Use the base metadata

ticketing/db_manager.py

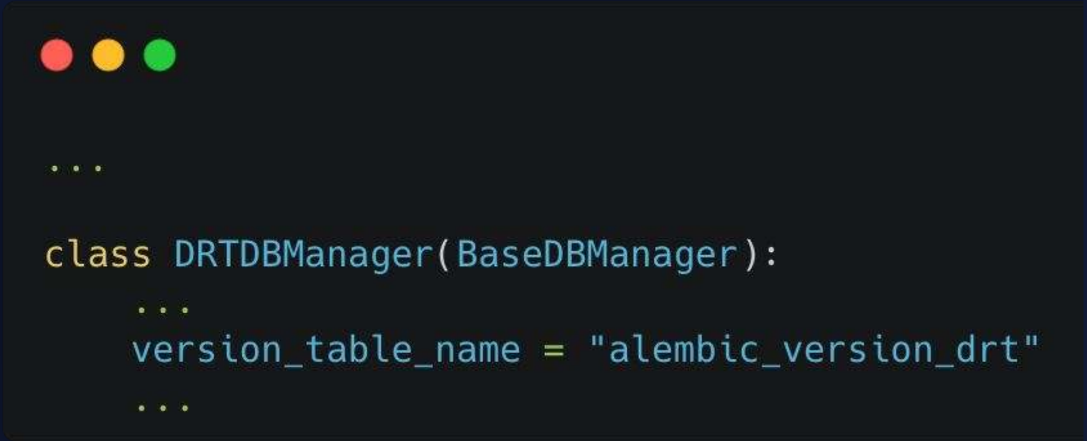


```
from ticketing.models import Base
...

class DRTDBManager(BaseDBManager):
    metadata = Base.metadata
    ...
```

ASTRONOMER

Unique version table name

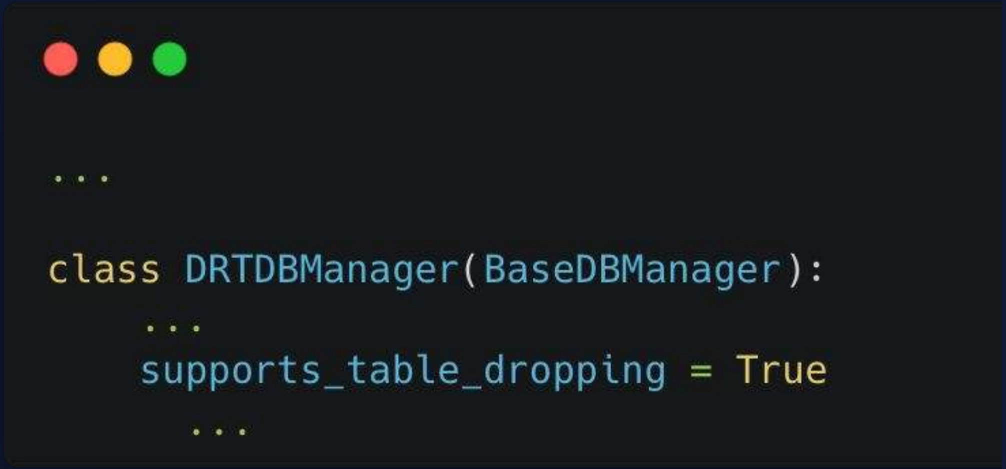
ticketing/db_manager.py



```
...

class DRTDBManager(BaseDBManager):
    ...
    version_table_name = "alembic_version_drt"
    ...
```

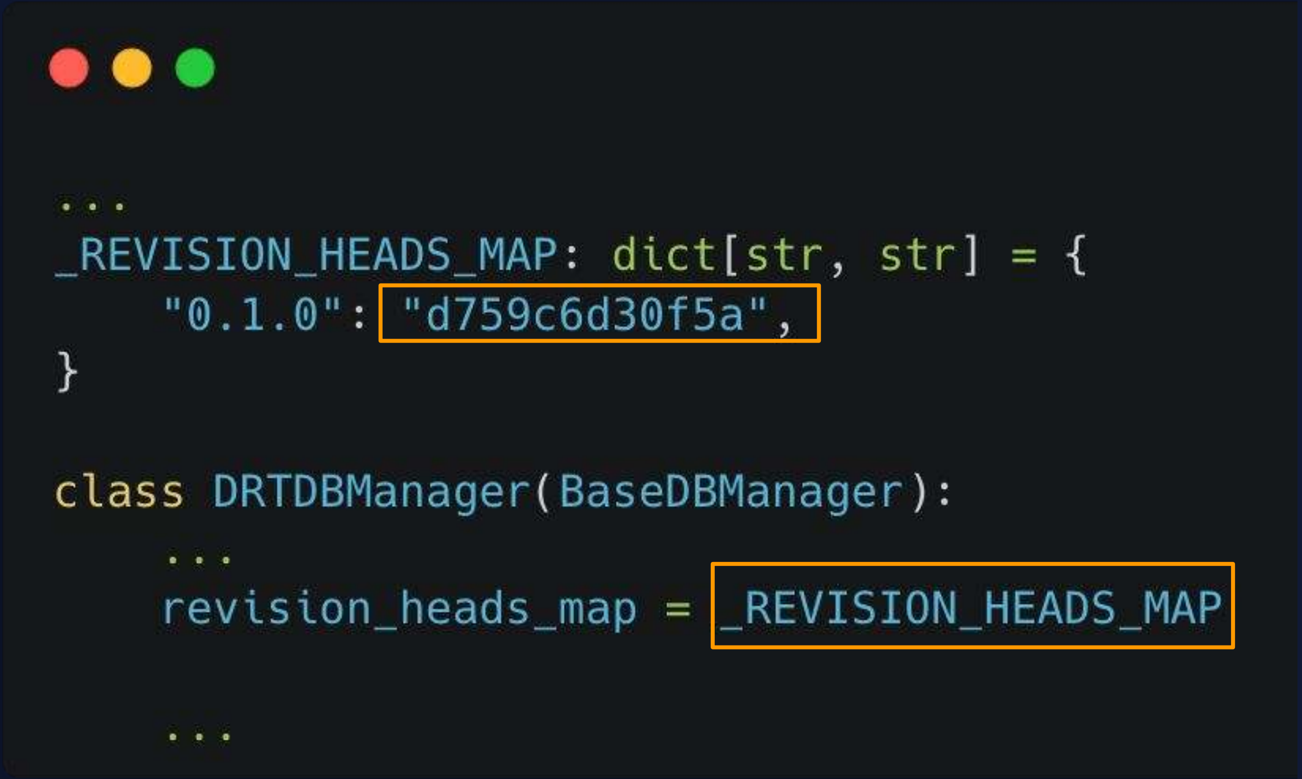
ASTRONOMER

ticketing/db_manager.py



```
...

class DRTDBManager(BaseDBManager):
    ...
    supports_table_dropping = True
      ...
```

ticketing/db_manager.py

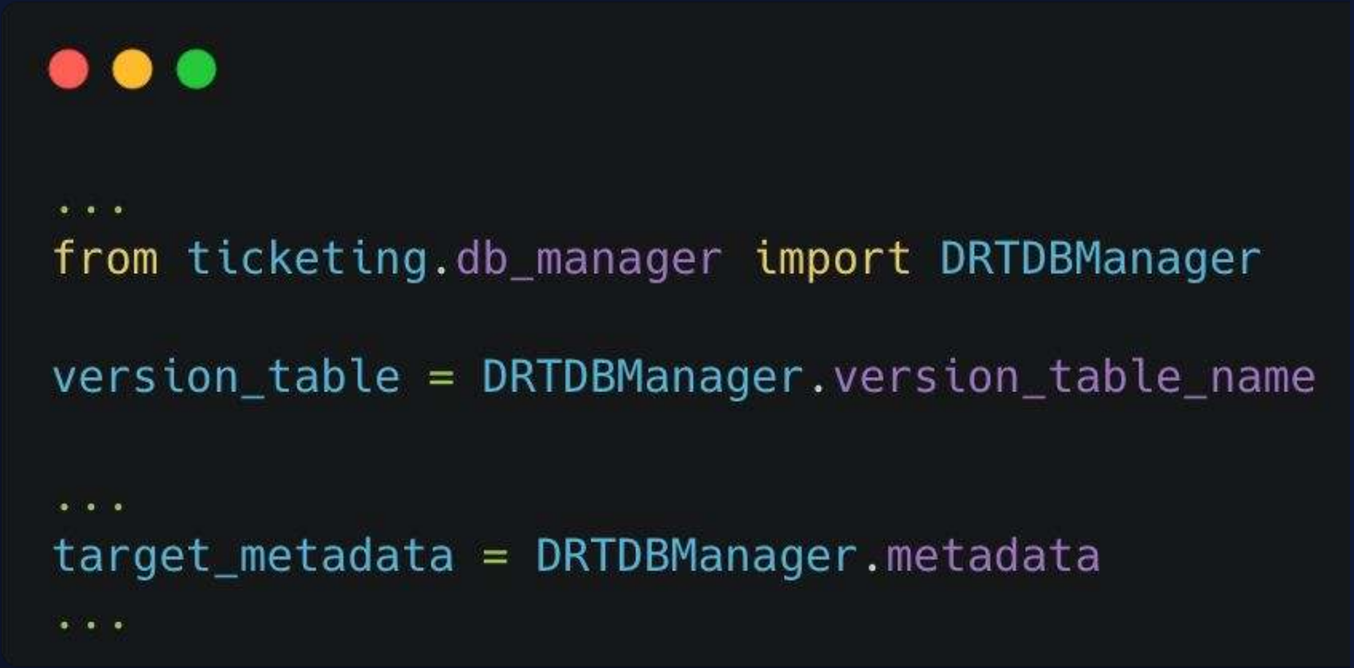


```
...
_REVISION_HEADS_MAP: dict[str, str] = {
    "0.1.0": "d759c6d30f5a",
}

class DRTDBManager(BaseDBManager):
    ...
    revision_heads_map = _REVISION_HEADS_MAP

    ...
```

ASTRONOMER

Linking the custom DB manager to the alembic migrations

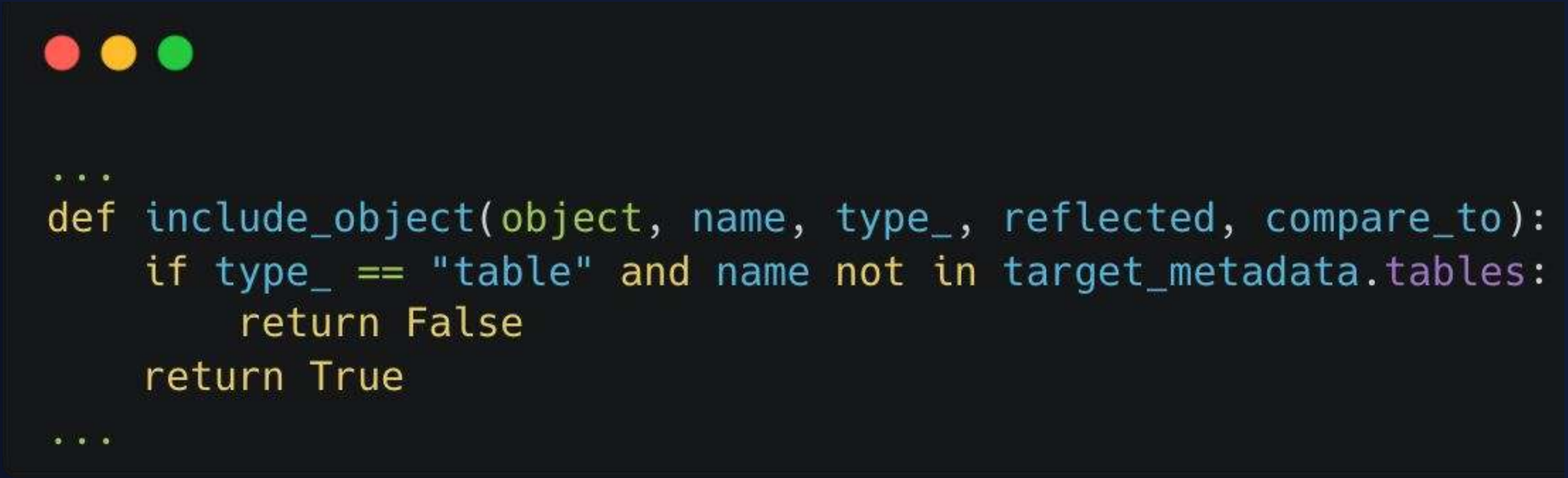
ticketing/migrations/env.py



```python
...
from ticketing.db_manager import DRTDBManager

version_table = DRTDBManager.version_table_name


...
target_metadata = DRTDBManager.metadata
...
```

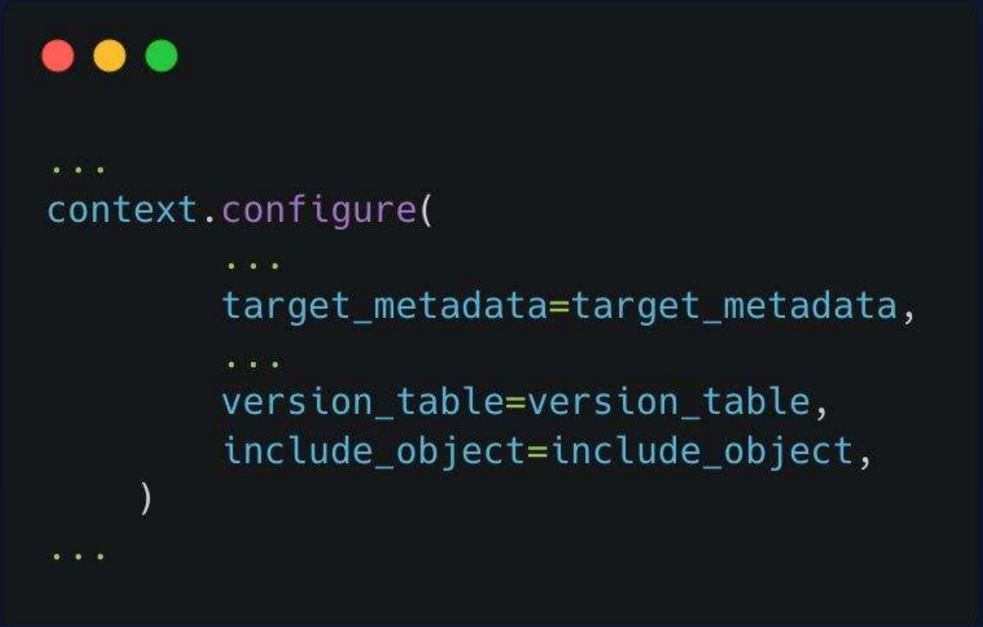
ASTRONOMER

ticketing/migrations/env.py



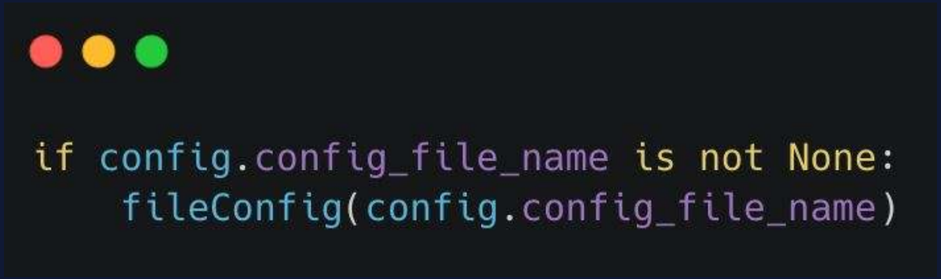
```python
...
def include_object(object, name, type_, reflected, compare_to):
    if type_ == "table" and name not in target_metadata.tables:
        return False
    return True
...
```

ticketing/migrations/env.py

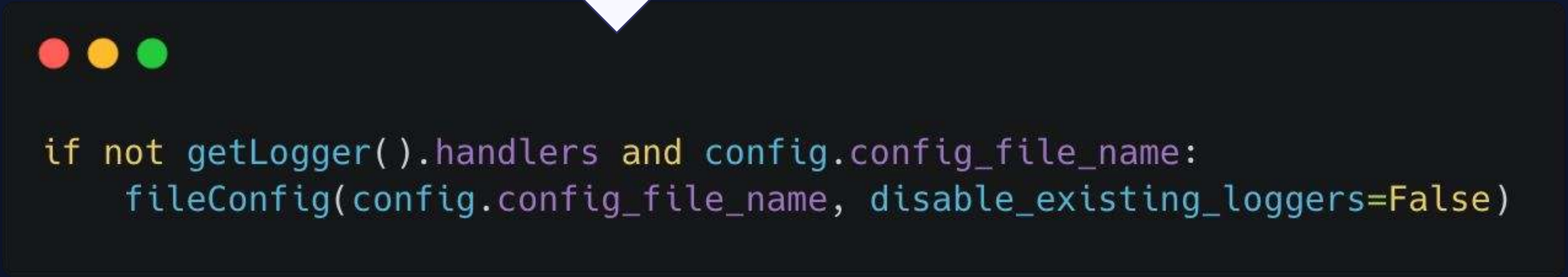


```
...
context.configure(
        ...
        target_metadata=target_metadata,
        ...
        version_table=version_table,
        include_object=include_object,
    )
...
```

ticketing/migrations/env.py

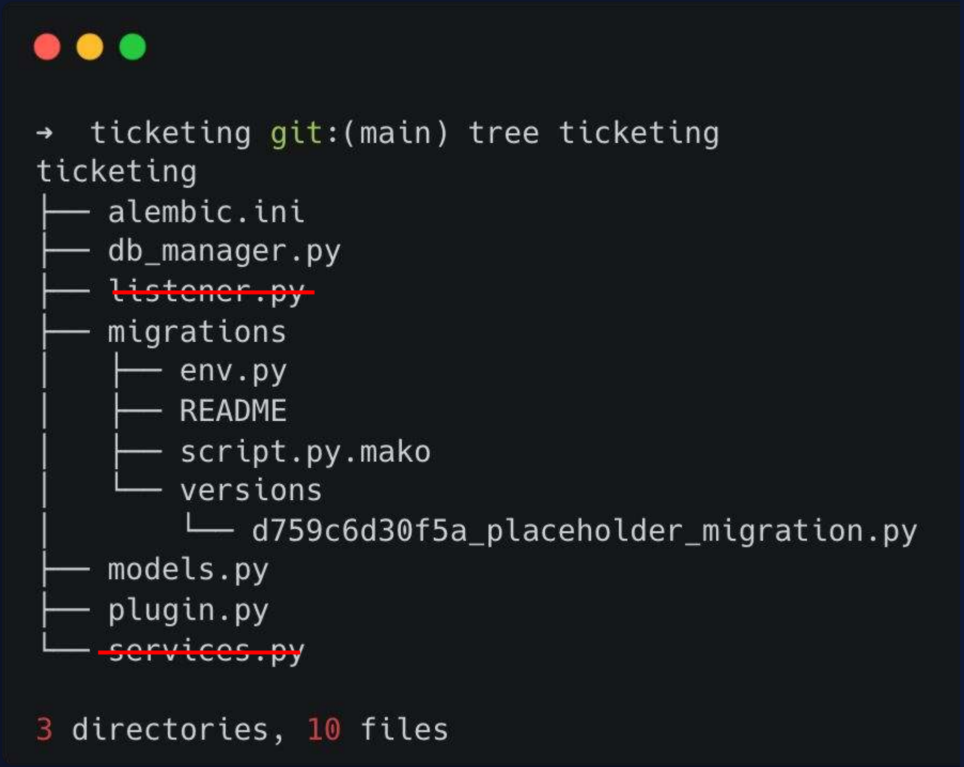


```python
if config.config_file_name is not None:
    fileConfig(config.config_file_name)
```



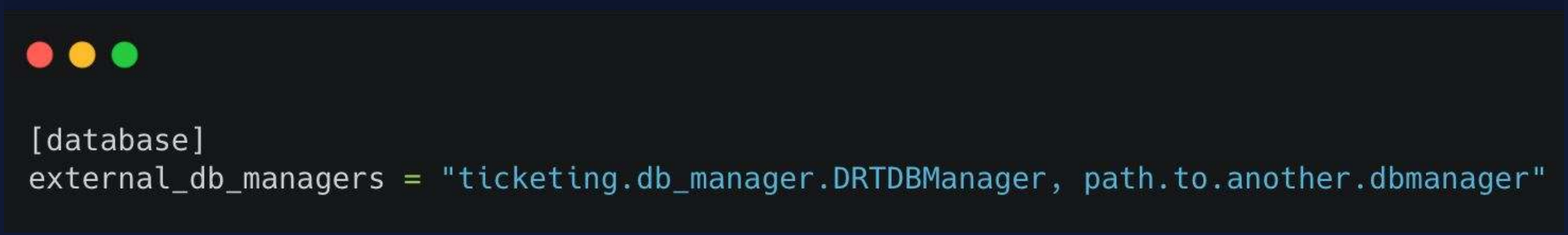
```python
if not getLogger().handlers and config.config_file_name:
    fileConfig(config.config_file_name, disable_existing_loggers=False)
```

ASTRONOMER



```
➜  ticketing git:(main) tree ticketing
ticketing
├── alembic.ini
├── db_manager.py
├── ~~listener.py~~
├── migrations
│   ├── env.py
│   ├── README
│   ├── script.py.mako
│   └── versions
│       └── d759c6d30f5a_placeholder_migration.py
├── models.py
├── plugin.py
└── ~~services.py~~

3 directories, 10 files
```

Linking the app DBManager to Airflow



```
[database]
external_db_managers = "ticketing.db_manager.DRTDBManager, path.to.another.dbmanager"
```

ASTRONOMER

Summary : Enabling Non-Core Migrations

1. Initialized Alembic for the plugin schema.
2. Implemented a DB Manager that points to that Alembic environment
3. Registered the DB Manager so Airflow can discover it.

i Note: If your plugin doesn’t own tables, you don’t need a DB Manager.

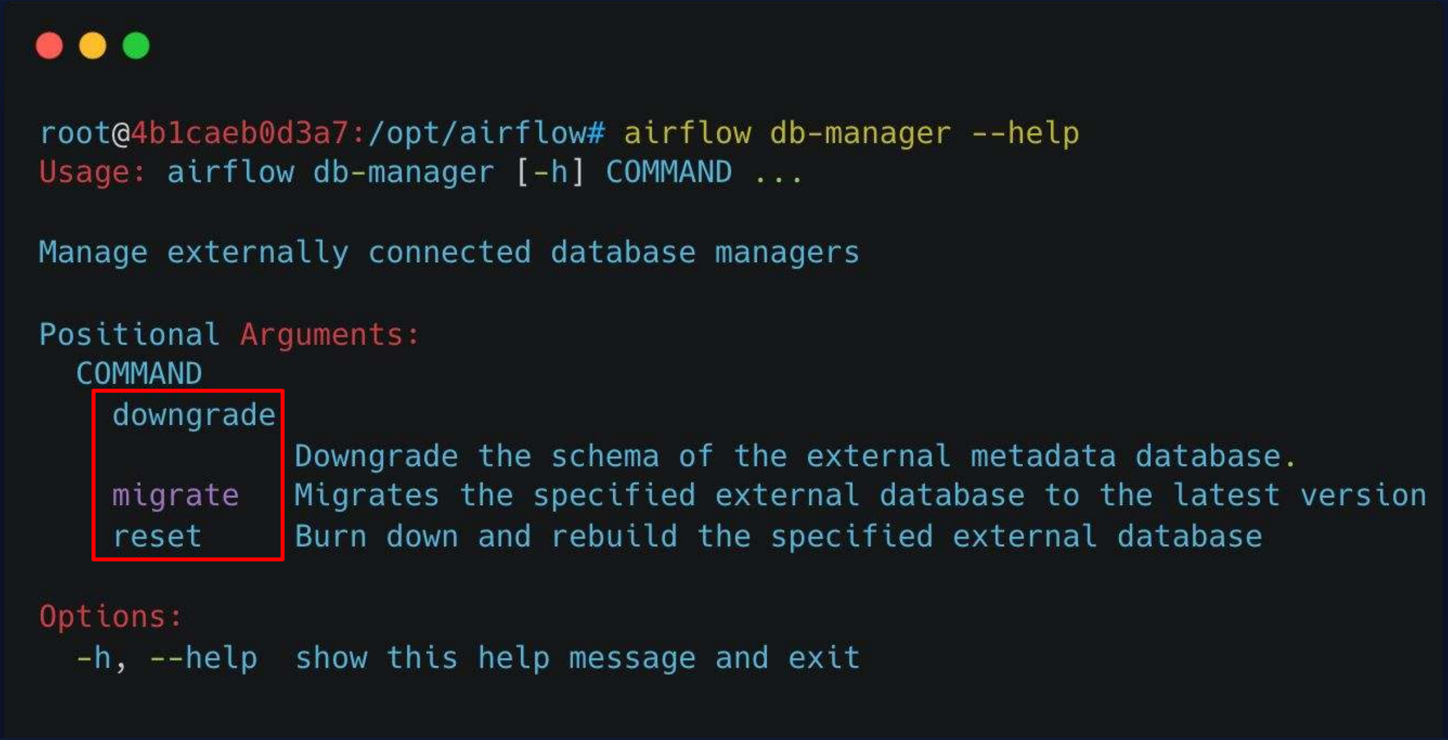
Possibilities

airflow db check-migrations will wait for your apps migration

✅ **airflow db migrate** will run your app’s db migration

❌ **airflow db downgrade** will not run your app’s db migration

DB Manager Commands



```
root@4b1caeb0d3a7:/opt/airflow# airflow db-manager --help
Usage: airflow db-manager [-h] COMMAND ...

Manage externally connected database managers

Positional Arguments:
  COMMAND
    downgrade
              Downgrade the schema of the external metadata database.
    migrate   Migrates the specified external database to the latest version
    reset     Burn down and rebuild the specified external database

Options:
  -h, --help  show this help message and exit
```

ASTRONOMER

```
airflow db-manager migrate "ticketing.db_manager.DRTDBManager" --to-version 0.1.0
airflow db-manager downgrade "ticketing.db_manager.DRTDBManager" --to-version 0.0.1
airflow db-manager reset "ticketing.db_manager.DRTDBManager" --skip-init
```

FAB provider is another example implementation of this integration

Questions?

Scan to access the demo App for the talk

ASTRONOMER

The 2025 Apache Airflow® Survey is here!

Fill it out to for a free Airflow 3 Fundamentals or DAG Authoring in Airflow 3 certification code

ASTRONOMER