Seamless Airflow Upgrades: Migrating from 2.x to 3

Ankit Chaurasia

Senior Software Engineer

ASTRONOMER

Why Upgrade to Airflow 3?

- Run anywhere, in any language
- Scheduler-managed backfills
- Modern UI & UX
- Event driven scheduling & Data assets
- Task SDK
- Lean core and Providers
- DAG Versioning
- Security upgrades
- Seamless upgrades
- Edge Executor
- Inference and hyperparameter tuning
- Developer experience

ASTRONOMER

Seamless Airflow 3 Upgrade: Step-by-Step Checklist

01 Take care of the Prerequisites

02 Backup & Clean your existing Airflow instance

03 Verify DAG Compatibility using Ruff

04 Update Airflow Configuration

05 Address known Breaking Changes

06 Upgrade and test in Development

07 Production Upgrade

08 Post-Upgrade Validation

ASTRONOMER

Step 1: Versions Check and Prepare for upgrade

Airflow Version: Must be 2.6.3 or higher (ideally 2.7.x) for a smooth Airflow 3 upgrade path

Python Version: Airflow 3 requires Python 3.9+; support for 3.7 and 3.8 is dropped

Database Compatibility: Metadata DB must be PostgreSQL 13+ or MySQL 8+ due to SQLAlchemy 2.x migration

Providers & Dependencies: Install apache-airflow-providers-standard plus any additional DAG providers needed

Astronomer users benefit from **managed runtimes**; open-source users must verify and upgrade manually

Step 2: Clean and Back Up Your Existing Airflow Instance

Backup Your Database

Make a backup of your Airflow metadata database before starting the migration. Shut down Airflow instances if no hot backup is available to ensure consistency.

Avoid Migration Risks

A backup prevents issues from failed migrations or network interruptions that could leave your system in a half-migrated state.

Clean Up Old Data

Use the airflow db clean CLI command to remove unnecessary data like old XComs to reduce database size and speed up schema changes.

Resolve DAG Errors

Ensure no DAG processing errors remain, such as AirflowDagDuplicatedIdException. Run airflow dags reserialize without errors before proceeding.

Step 3 – Ensure DAG Code Compatibility with Ruff

- Install Ruff v0.13.1 or later: ***pip install --upgrade ruff***
- Run lint checks for breaking changes: `***ruff check dags/ --select AIR301***`.
- Preview automated corrections with `**--show-fixes**` to verify suggested code updates
- Apply safe fixes instantly using `**--fix**`, updating imports and parameters without manual edits
- Use `**--fix --unsafe-fixes**` cautiously for import path changes that may alter behavior
- Re-run checks until no AIR rule violations remain, confirming code is fully upgraded
- Astronomer users can execute `**astro dev upgrade-test**` to automate all steps including dependency verification

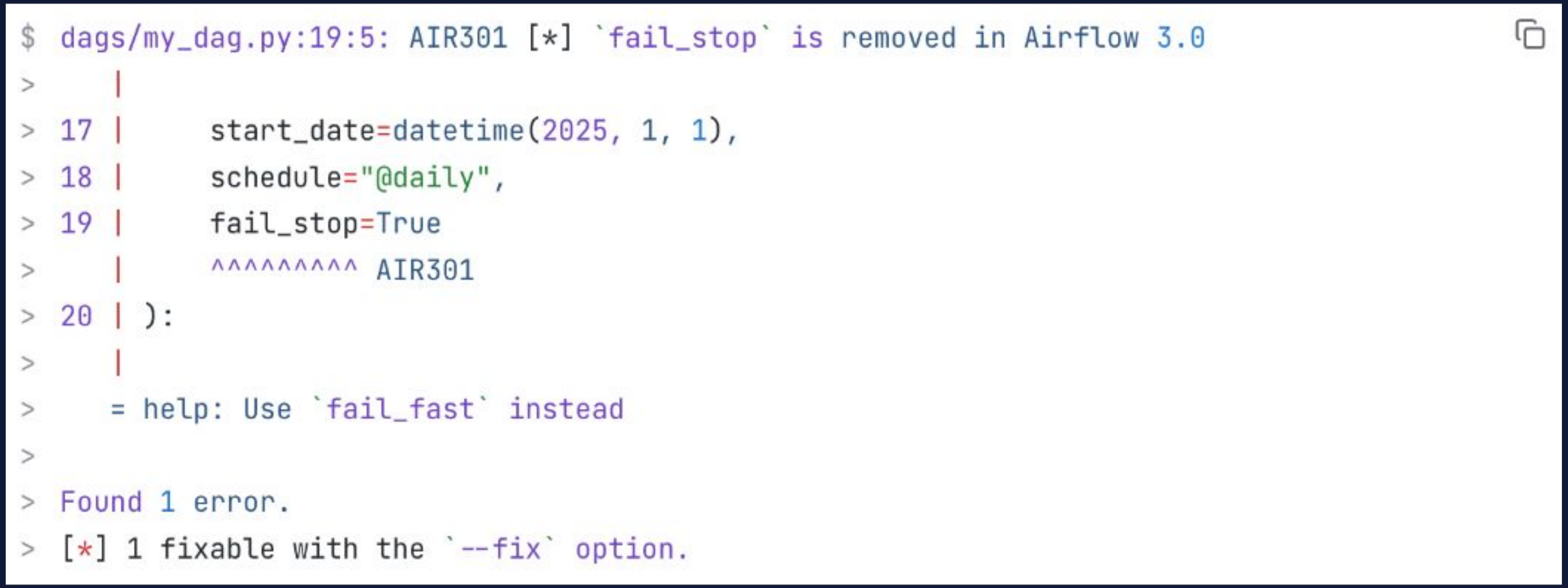
Ruff Rules for Airflow 3 Upgrade

AIR30 Rules (Mandatory)

1. Check for removed parameters and imports no longer available in Airflow 3
2. Mandatory changes to ensure DAGs function correctly in Airflow 3

AIR31 Rules

1. Check for deprecated parameters and imports still available in Airflow 3
2. Recommended changes to maintain compatibility with future Airflow versions



```
$ dags/my_dag.py:19:5: AIR301 [*] `fail_stop` is removed in Airflow 3.0
>     |
> 17 |     start_date=datetime(2025, 1, 1),
> 18 |     schedule="@daily",
> 19 |     fail_stop=True
>     |     ^^^^^^^^^ AIR301
> 20 | ):
>     |
>     = help: Use `fail_fast` instead
>
> Found 1 error.
> [*] 1 fixable with the `--fix` option.
```

ASTRONOMER

Airflow OSS Ruff linter vs. Astro CLI

Using Ruff Linter (OSS)

1. Install the latest Ruff linter via pip: `pip install --upgrade ruff`
2. Run Ruff on your DAG code: `ruff check --preview --select AIR30 `
3. Use `--fix` flag to automatically fix some mandatory issues
4. Ruff outputs errors with suggestions, e.g., renaming deprecated DAG params like `fail_stop` to `fail_fast`. Fix the mandatory issues.
5. Requires manual review of breaking changes and Airflow release notes for full compatibility

Using Astro CLI

1. Run `astro dev upgrade-test` which also includes Ruff linter as part of command and fix the ruff issues.
2. Automatic detection of Airflow 3 compatibility issues within Astro environment

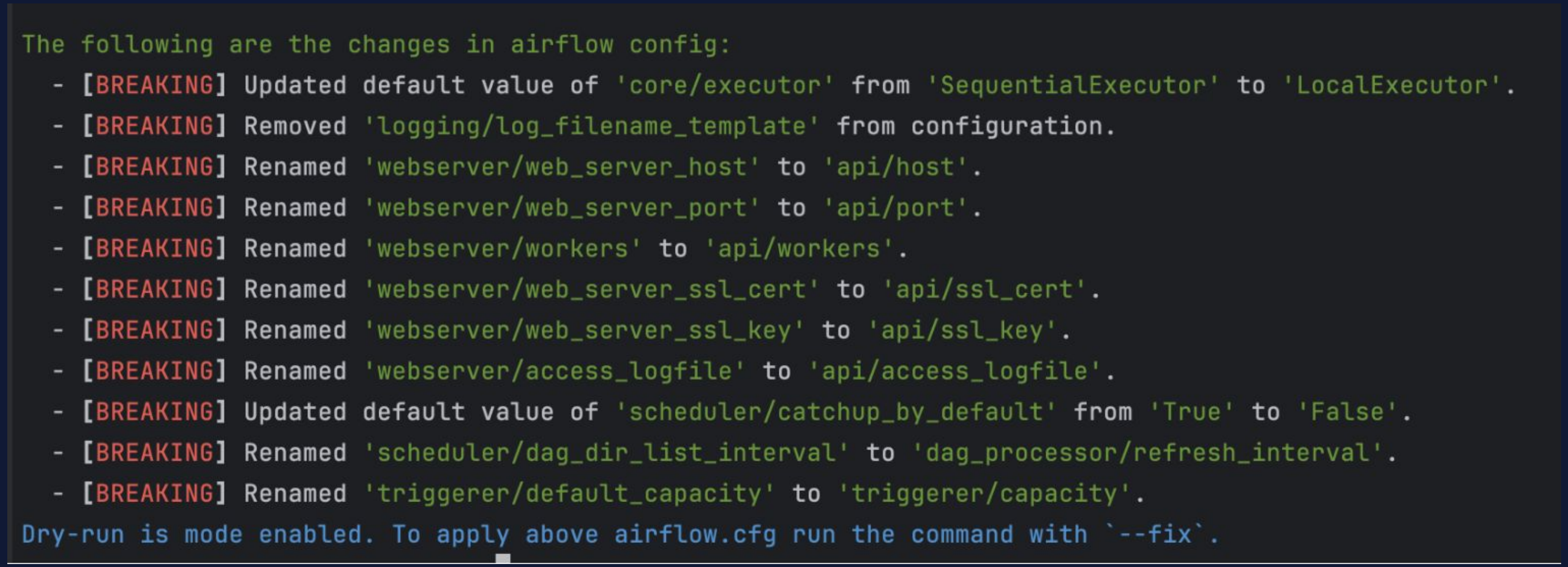
Step 4: Check and fix breaking Airflow configs

Upgrade to **Airflow 2.11.0** as **airflow config update** is available from this version.

Run **airflow config update** to detect deprecated, moved, or invalid configs in airflow.cfg

Review both **breaking** and **recommended** config changes

Use **airflow config update --fix** to auto-apply fixes with backup creation for old airflow configuration file



```
The following are the changes in airflow config:
  - [BREAKING] Updated default value of 'core/executor' from 'SequentialExecutor' to 'LocalExecutor'.
  - [BREAKING] Removed 'logging/log_filename_template' from configuration.
  - [BREAKING] Renamed 'webserver/web_server_host' to 'api/host'.
  - [BREAKING] Renamed 'webserver/web_server_port' to 'api/port'.
  - [BREAKING] Renamed 'webserver/workers' to 'api/workers'.
  - [BREAKING] Renamed 'webserver/web_server_ssl_cert' to 'api/ssl_cert'.
  - [BREAKING] Renamed 'webserver/web_server_ssl_key' to 'api/ssl_key'.
  - [BREAKING] Renamed 'webserver/access_logfile' to 'api/access_logfile'.
  - [BREAKING] Updated default value of 'scheduler/catchup_by_default' from 'True' to 'False'.
  - [BREAKING] Renamed 'scheduler/dag_dir_list_interval' to 'dag_processor/refresh_interval'.
  - [BREAKING] Renamed 'triggerer/default_capacity' to 'triggerer/capacity'.
Dry-run is mode enabled. To apply above airflow.cfg run the command with `--fix`.
```

ASTRONOMER

Types of Airflow 3 configuration changes

Default Values Changes

Renamed Options

Removed Options

Invalidated Previously Valid Options

Step 5 – Assess Breaking Changes

Direct DB Access Removed: Replace task code that opens DB sessions or queries metadata directly with Airflow public APIs or REST calls.

Scheduling & Timetables Changes: Replace `schedule_interval` with `schedule`. New cron triggers disable automatic backfills by default; enable with `scheduler create_cron_data_intervals=True`.

Execution Date Context: Update DAGs to replace `execution_date` and related variables with `logical_date` and `data_interval_start/end`.

Removed Features: Migrate from SubDAGs to TaskGroups or Datasets; replace SLA miss alerts with external monitoring; update plugins to new system as FAB-based plugins need compatibility provider.

XCom Serialization: Disallow pickled XComs by default for security. Migrate to custom backends. Old pickled XComs archived during DB migration.

Provider Changes: Operators and hooks moved to separate provider packages requiring explicit installation.

Step 6 – Upgrade & Test Airflow 3 Locally

01

Set up a local or staging Airflow environment

02

Upgrade Airflow to version 3, apply all config and DAG code changes, then run airflow db upgrade to migrate metadata schema

03

Start Airflow 3 components: API server replaces webserver; run dag-processor alongside scheduler

04

Trigger and monitor all critical DAGs, verifying task execution, scheduling, XComs, and templated variables

05

Fix errors iteratively and re-test until all DAGs run cleanly

Upgrade and Test locally (OSS method)

01

Install Airflow 3

Install Airflow 3 (e.g., pip install apache-airflow==3.0.0 with constraints, or use the official Airflow 3 Docker image). Apply the same configuration changes using the updated airflow.cfg and include the DAG code fixed via Ruff.

02

Run Ruff linter and airflow config update

Run Ruff's AIR rules linter to identify breaking code changes. Use airflow config update tool to migrate config files to Airflow 3 format.

03

Database Migrations

Run database migrations with airflow db upgrade (equivalent to airflow db migrate) to upgrade the metadata DB schema.

04

Change your startup script

Run airflow api-server and airflow dag-processor

05

Run airflow and test the DAGs

Run database migrations with airflow db upgrade (equivalent to airflow db migrate) to upgrade the metadata DB schema.

ASTRONOMER

Demo Upgrade and Test locally (OSS method)

Upgrade and test locally: Using astro cli

01

Update Astro Project

Change your Astro project's Dockerfile base image to the new Astro Runtime (Airflow 3) tag from astronomer.io.

02

Run upgrade compatibility tests and fix

Use Astro CLI command 'astro dev upgrade-test --runtime-version ' to run compatibility tests including dependencies.

03

Start Airflow 3 locally

Execute 'astro dev start' or 'astro dev restart' to launch Airflow 3 locally with your project's DAGs and config.

04

Verify Local Setup and Run DAGs

Access the Airflow 3 web UI at http://localhost:8080 to verify that components like API server, scheduler, triggerer, and local Postgres are running.

ASTRONOMER

Demo Upgrade and test locally: Using astro cli

Choose Your Production Upgrade Strategy: Blue-Green vs In-Place

Blue-Green Deployment (Recommended)

1. Set up new Airflow 3 environment alongside existing Airflow 2
2. Deploy updated DAGs and config with fresh or migrated metadata DB
3. Test fully before switching traffic to Airflow 3 to minimize downtime
4. Supports easy rollback by switching back to Airflow 2
5. Astronomer users can upgrade via UI or CLI seamlessly
6. Open-source users must provision new infrastructure and handle DB migration

In-Place Upgrade (Advanced)

1. Upgrade existing environment by installing Airflow 3 and migrating live DB
2. Requires downtime during migration and service restarts
3. Rollback is complex
4. Must update service scripts for new API server and dag-processor

Essential Q&A & Upgrade Resources

Astronomer’s Upgrade Guide (Airflow 2 → 3)

Astronomer Support & Community Slack

Official Apache Airflow Documentation – “Upgrading to Airflow 3”

Apache Airflow Slack Channel (#airflow-upgrade)

Astral (Ruff) Airflow Rules Documentation: Details on AIR301, AIR302 (mandatory) and AIR311, AIR312 (recommended) linter rules

QUESTION?

ASTRONOMER

The 2025 Apache Airflow® Survey is here!

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

ASTRONOMER