Event-based DAG parsing No more F5ing in the UI

Airflow Summit 2023
Bas Harenslak

ASTRONOMER

Event-based DAG parsing

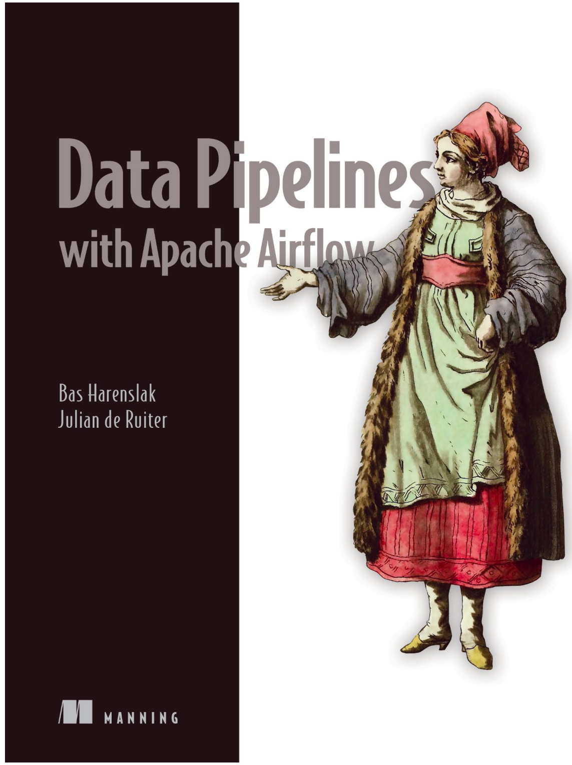
Have you ever:

- Experienced your new DAG not showing up in the Airflow UI?
- Experienced your code changes not showing up in the Airflow UI?
- Found yourself F5ing and waiting for changes to show in the Airflow UI?

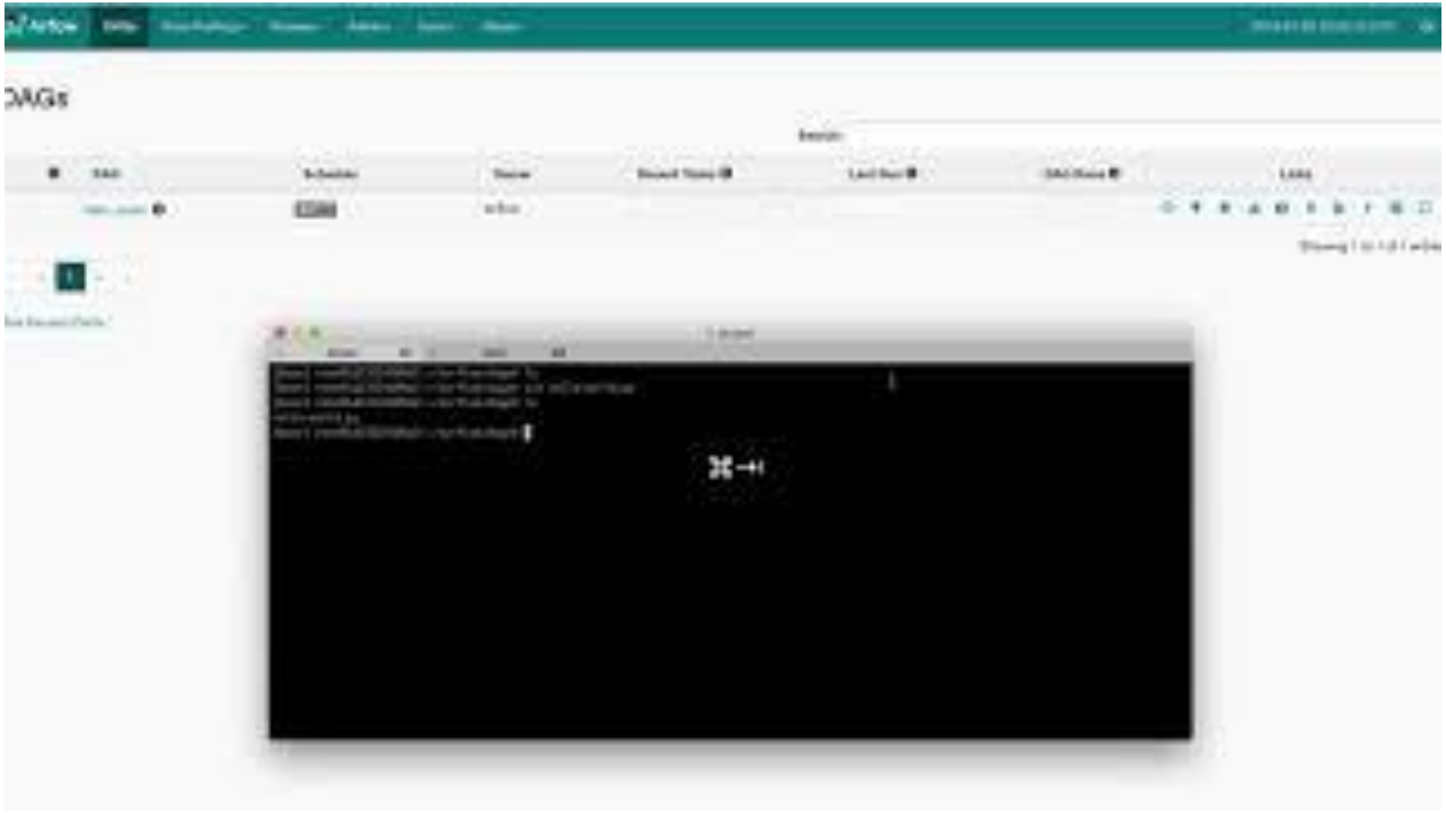
Whoami

Bas Harenslak

- Senior Solutions Architect at Astronomer
- Co-author of book Data Pipelines with Apache Airflow
- Committer on the Apache Airflow project



Airflow without event-based DAG parsing



[illegible]

ASTRONOMER

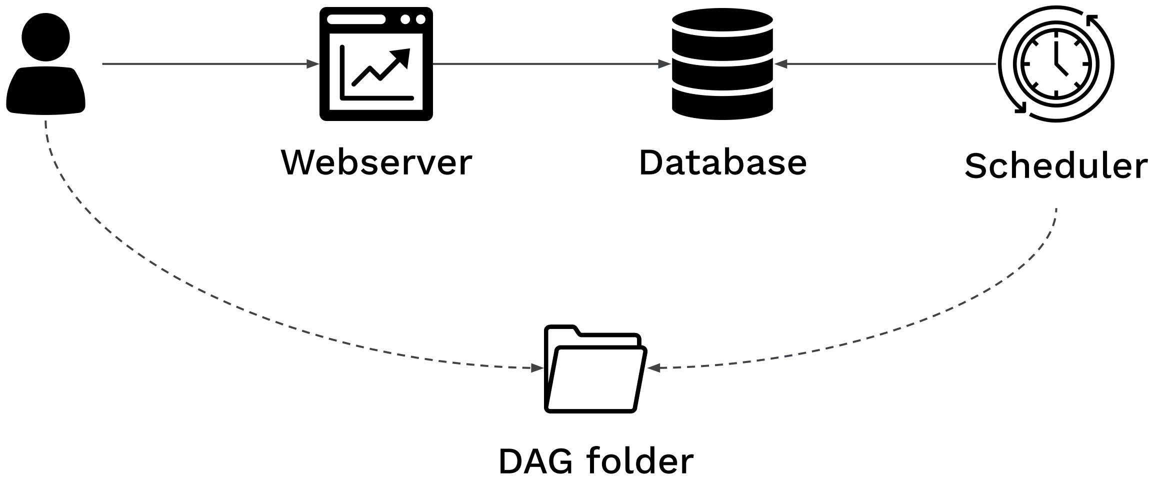
What this talk is about

1. How the current DAG parser implementation and configuration works
2. How an event-based DAG parser is implemented
3. A demonstration of an Airflow UI without having to refresh

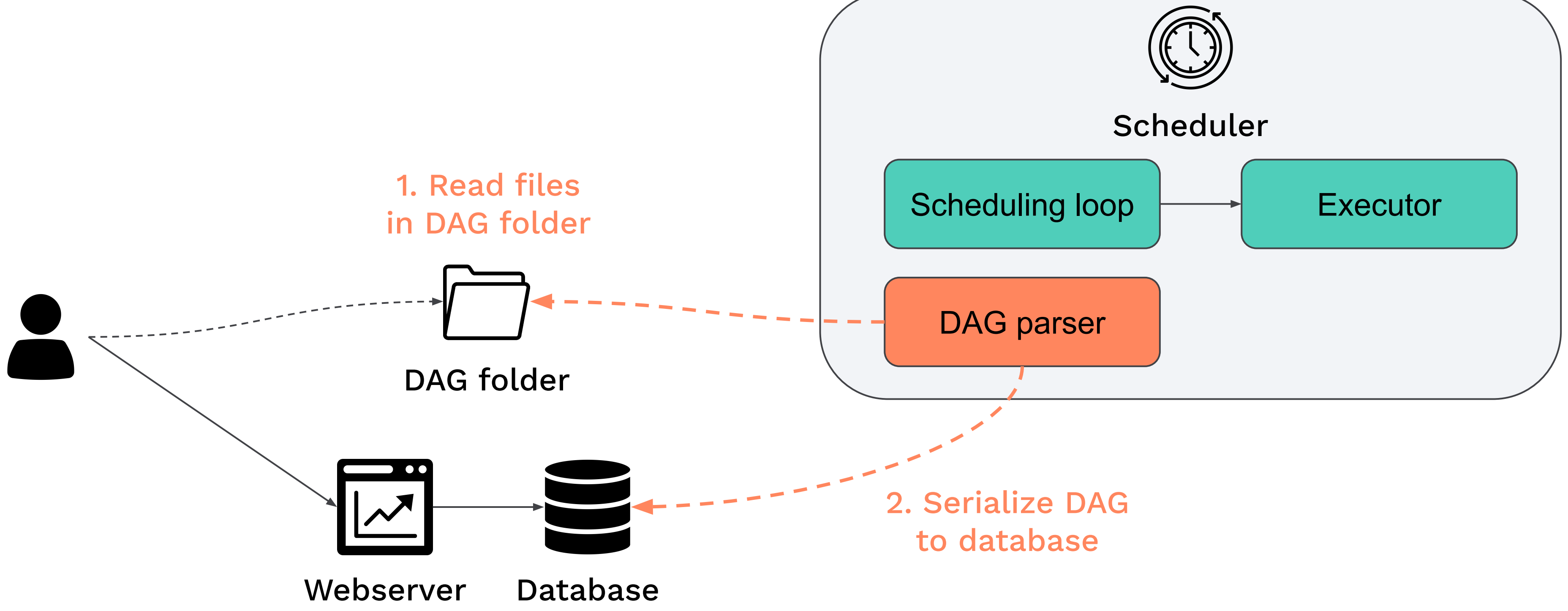
What is the “DAG parser” in Airflow?

Bare minimum Airflow:

```
airflow db init

airflow webserver

airflow scheduler
```



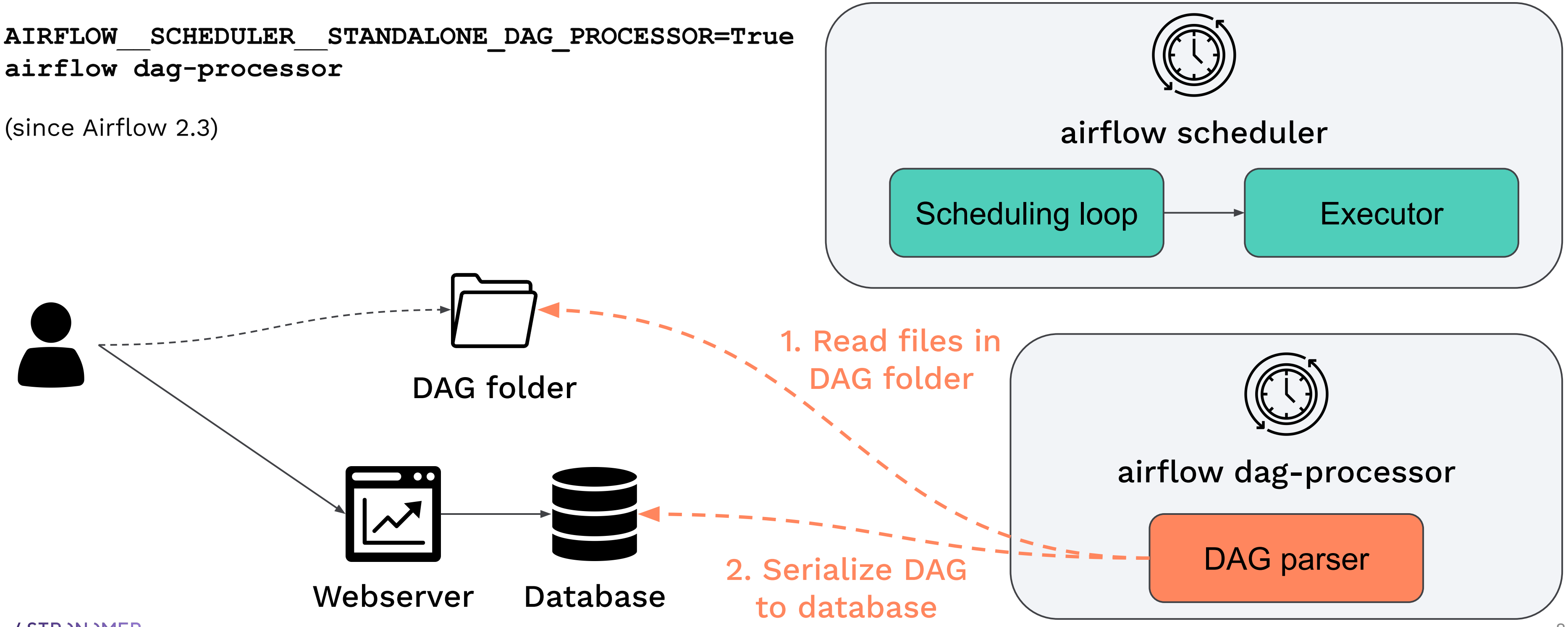
A closer look at the scheduler



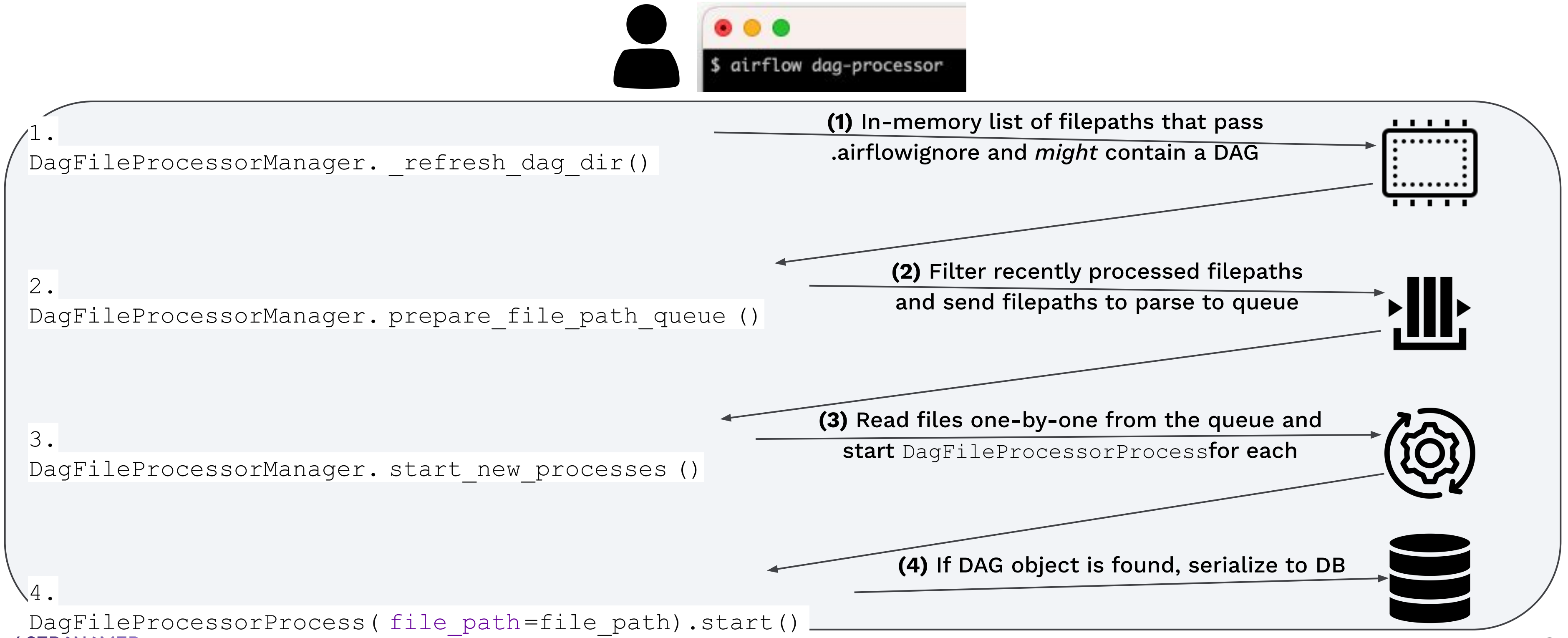
A closer look at the scheduler

```
AIRFLOW__SCHEDULER__STANDALONE_DAG_PROCESSOR=True
airflow dag-processor
```

(since Airflow 2.3)



A closer look at the DAG parser



```
$ airflow dag-processor
```

```
1.
DagFileProcessorManager._refresh_dag_dir()
```

(1) In-memory list of filepaths that pass .airflowignore and *might* contain a DAG

```
2.
DagFileProcessorManager.prepare_file_path_queue()
```

(2) Filter recently processed filepaths and send filepaths to parse to queue

```
3.
DagFileProcessorManager.start_new_processes()
```

(3) Read files one-by-one from the queue and start `DagFileProcessorProcess`for each

```
4.
DagFileProcessorProcess(file_path=file_path).start()
```

(4) If DAG object is found, serialize to DB

Configuring the DAG parser

Configuring the function listing files as "potential DAG-containing files":

`DagFileProcessorManager._refresh_dag_dir()`

- **`AIRFLOW__SCHEDULER__DAG_DIR_LIST_INTERVAL`**
    - Defined in seconds
    - Default `300` (=5 minutes)

- **`AIRFLOW__CORE__MIGHT_CONTAIN_DAG_CALLABLE`**
    - Default `airflow.utils.file.might_contain_dag_via_default_heuristic`
    - Default looks for words "`dag`" and "`airflow`" in a file

Configuring the DAG parser

Configuring the function parsing "potential DAG-containing files" for DAG objects:

`DagFileProcessorManager.prepare_file_path_queue()`

- **`AIRFLOW__SCHEDULER__FILE_PARSING_SORT_MODE`**
  - Default `"modified_time"`
  - Options are:
    - `"modified_time"`→ (default) Sort by last modified time. Best for fastest processing of changes.
    - `"random_seeded_by_host"`→ Sort randomly per DAG parser (useful if HA)
    - `"alphabetical"`→ Sort alphabetically by filename

- **`AIRFLOW__SCHEDULER__MIN_FILE_PROCESS_INTERVAL`**
  - Default `30`
  - Number of seconds after which a file is selected for parsing. Disregarded (i.e. 0) if file was modified. Lower == more processing == faster updates.

Configuring the DAG parser

Configuring the function starting processes to parse files for DAG objects:

`DagFileProcessorManager.start_new_processes()`

- `AIRFLOW__SCHEDULER__PARSING_PROCESSES`
    - Default 2
    - Defines maximum processes to start for parsing files (1 file per process)

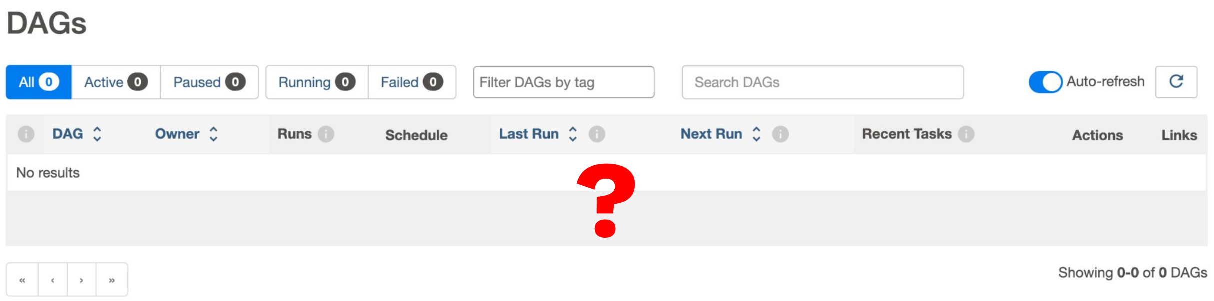
Configuring the webserver

There's a webserver setting related to auto-refreshing:

- **`AIRFLOW__WEBSERVER__AUTO_REFRESH_INTERVAL`**
  - Default 3
  - Defines # of seconds between requests to the Airflow API
    - Lower == faster refreshes == higher load on the Airflow webserver

Challenges with the current DAG parser

- New files can take up to 5 minutes (default setting) to get processed
- The user doesn't know where in this 5 minute cycle they are


DAGs

All 0 | Active 0 | Paused 0 | Running 0 | Failed 0 | Filter DAGs by tag | Search DAGs | Auto-refresh

	DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions	Links
No results									

?

Showing **0-0** of **0** DAGs

Now, how about event-based DAG parsing?

Main goal is not having to refresh anymore after changes in the DAG folder

Now, how about event-based DAG parsing?

watchdog package

- https://pypi.org/project/watchdog
- https://github.com/gorakhargosh/watchdog

Provides an API to monitor file system events

- Cross-platform
  - Linux inotify
  - MacOS FSEvents
  - Windows ReadDirectoryChangesW
  - Or OS-independent polling
- Is actively maintained
- Apache License 2.0 (just like Airflow)

Hello world with watchdog

```
import logging
import time

from watchdog.events import LoggingEventHandler
from watchdog.observers import Observer

if   name   == "  main  ":
    logging.basicConfig(level=logging.INFO, format="%(asctime)s - %(message)s", datefmt="%Y-%m-%d %H:%M:%S")
    dags_folder = "/Users/basharenslak/git/airflow/dags"

    event_handler = LoggingEventHandler()
    observer = Observer()
    observer.schedule(event_handler, dags_folder, recursive=True)
    observer.start()
    logging.info("Started watching directory %s for changes.", dags_folder)

    try:
        while True:
            time.sleep(1)
    finally:
        observer.stop()
        observer.join()
        logging.info("Stopped")
```

Hello world with watchdog

Import watchdog modules

```
import logging
import time

from watchdog.events import LoggingEventHandler
from watchdog.observers import Observer

if   name   == "  main  ":
    logging.basicConfig(level=logging.INFO, format="%(asctime)s - %(message)s", datefmt="%Y-%m-%d %H:%M:%S")
    dags_folder = "/Users/basharenslak/git/airflow/dags"

    event_handler = LoggingEventHandler()
    observer = Observer()
    observer.schedule(event_handler, dags_folder, recursive=True)
    observer.start()
    logging.info("Started watching directory %s for changes.", dags_folder)

    try:
        while True:
            time.sleep(1)
    finally:
        observer.stop()
        observer.join()
        logging.info("Stopped")
```

Hello world with watchdog

```python
import logging
import time

from watchdog.events import LoggingEventHandler
from watchdog.observers import Observer

if   name   == "  main  ":
    logging.basicConfig(level=logging.INFO, format="%(asctime)s - %(message)s", datefmt="%Y-%m-%d %H:%M:%S")
    dags_folder = "/Users/basharenslak/git/airflow/dags"

    event_handler = LoggingEventHandler()
    observer = Observer()
    observer.schedule(event_handler, dags_folder, recursive=True)
    observer.start()
    logging.info("Started watching directory %s for changes.", dags_folder)

    try:
        while True:
            time.sleep(1)
    finally:
        observer.stop()
        observer.join()
        logging.info("Stopped")
```

EventHandler dispatches events to user-provided functions, for example:

`ON_CREATED -> do_something_on_created_file()`

Hello world with watchdog

Observer is the “main” watchdog component

```
import logging
import time

from watchdog.events import LoggingEventHandler
from watchdog.observers import Observer

if   name   == "  main  ":
    logging.basicConfig(level=logging.INFO, format="%(asctime)s - %(message)s", datefmt="%Y-%m-%d %H:%M:%S")
    dags_folder = "/Users/basharenslak/git/airflow/dags"

    event_handler = LoggingEventHandler()
    observer = Observer()
    observer.schedule(event_handler, dags_folder, recursive=True)
    observer.start()
    logging.info("Started watching directory %s for changes.", dags_folder)

    try:
        while True:
            time.sleep(1)
    finally:
        observer.stop()
        observer.join()
        logging.info("Stopped")
```

Hello world with watchdog

If event X, then run Y

Path to watch

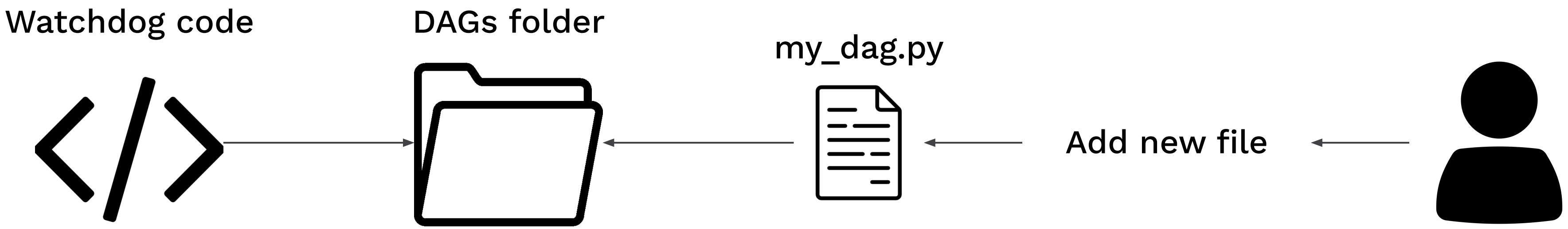
```python
import logging
import time

from watchdog.events import LoggingEventHandler
from watchdog.observers import Observer

if __name__ == "__main__":
    logging.basicConfig(level=logging.INFO, format="%(asctime)s - %(message)s", datefmt="%Y-%m-%d %H:%M:%S")
    dags_folder = "/Users/basharenslak/git/airflow/dags"

    event_handler = LoggingEventHandler()
    observer = Observer()
    observer.schedule(event_handler, dags_folder, recursive=True)
    observer.start()
    logging.info("Started watching directory %s for changes.", dags_folder)

    try:
        while True:
            time.sleep(1)
    finally:
        observer.stop()
        observer.join()
        logging.info("Stopped")
```

Hello world with watchdog

```python
import logging
import time

from watchdog.events import LoggingEventHandler
from watchdog.observers import Observer

if __name__ == "__main__":
    logging.basicConfig(level=logging.INFO, format="%(asctime)s - %(message)s", datefmt="%Y-%m-%d %H:%M:%S")
    dags_folder = "/Users/basharenslak/git/airflow/dags"

    event_handler = LoggingEventHandler()
    observer = Observer()
    observer.schedule(event_handler, dags_folder, recursive=True)
    observer.start()
    logging.info("Started watching directory %s for changes.", dags_folder)

    try:
        while True:
            time.sleep(1)
    finally:
        observer.stop()
        observer.join()
        logging.info("Stopped")
```

Start watching

Hello world with watchdog



```
2023-09-01 10:21:06 - Created file: /Users/basharenslak/git/airflow/dags/my_dag.py
```

Implementing an AirflowEventHandler

```python
import logging
import time

from watchdog.events import PatternMatchingEventHandler, FileCreatedEvent
from watchdog.observers import Observer


def handle_created_file(filepath: str):
    logging.info("... Check if %s contains DAG ...", filepath)


class AirflowEventHandler(PatternMatchingEventHandler):
    def __init__(self):
        PatternMatchingEventHandler.__init__(
            self,
            patterns=["*.py", "*.zip", "*.airflowignore"],
            ignore_directories=True
        )

    def on_created(self, event: FileCreatedEvent):
        logging.info("Detected creation of %s", event.src_path)
        handle_created_file(filepath=event.src_path)
```

Implementing an AirflowEventHandler

```
import logging
import time

from watchdog.events import PatternMatchingEventHandler, FileCreatedEvent
from watchdog.observers import Observer


def handle_created_file(filepath: str):
    logging.info("... Check if %s contains DAG ...", filepath)


class AirflowEventHandler(PatternMatchingEventHandler):
    def   init   (self):
        PatternMatchingEventHandler. __init__(
            self,
            patterns=["*.py", "*.zip", "*.airflowignore"],
            ignore_directories=True
        )

    def on_created(self, event: FileCreatedEvent):
        logging.info("Detected creation of %s", event.src_path)
        handle_created_file(filepath=event.src_path)
```

Code to run after file is created

Implementing an AirflowEventHandler

```python
import logging
import time

from watchdog.events import PatternMatchingEventHandler, FileCreatedEvent
from watchdog.observers import Observer


def handle_created_file(filepath: str):
    logging.info("... Check if %s contains DAG ...", filepath)


class AirflowEventHandler(PatternMatchingEventHandler):
    def __init__(self):
        PatternMatchingEventHandler. __init__ (
            self,
            patterns=["*.py", "*.zip", "*.airflowignore"],
            ignore_directories =True
        )

    def on_created(self, event: FileCreatedEvent):
        logging.info( "Detected creation of %s" , event.src_path)
        handle_created_file(filepath=event.src_path)
```

Subclass watchdog EventHandler

Implementing an AirflowEventHandler

```
import logging
import time

from watchdog.events import PatternMatchingEventHandler, FileCreatedEvent
from watchdog.observers import Observer


def handle_created_file(filepath: str):
    logging.info("... Check if %s contains DAG ...", filepath)


class AirflowEventHandler(PatternMatchingEventHandler):
    def __init__(self):
        PatternMatchingEventHandler.__init__(
            self,
            patterns=["*.py", "*.zip", "*.airflowignore"],
            ignore_directories=True
        )

    def on_created(self, event: FileCreatedEvent):
        logging.info("Detected creation of %s", event.src_path)
        handle_created_file(filepath=event.src_path)
```

Patterns to watch in given path

Implementing an AirflowEventHandler

```python
import logging
import time

from watchdog.events import PatternMatchingEventHandler, FileCreatedEvent
from watchdog.observers import Observer


def handle_created_file(filepath: str):
    logging.info("... Check if %s contains DAG ...", filepath)


class AirflowEventHandler(PatternMatchingEventHandler):
    def __init__(self):
        PatternMatchingEventHandler.__init__(
            self,
            patterns=["*.py", "*.zip", "*.airflowignore"],
            ignore_directories=True
        )

    def on_created(self, event: FileCreatedEvent):
        logging.info("Detected creation of %s", event.src_path)
        handle_created_file(filepath=event.src_path)
```

Mapping on_created event to user-defined function

Implementing event-based DAG parsing with watchdog

- Implement AirflowEventHandler
- Watch the DAGs folder
- Make it trigger a "`handle_{created,moved,modified,deleted}_file()`" function

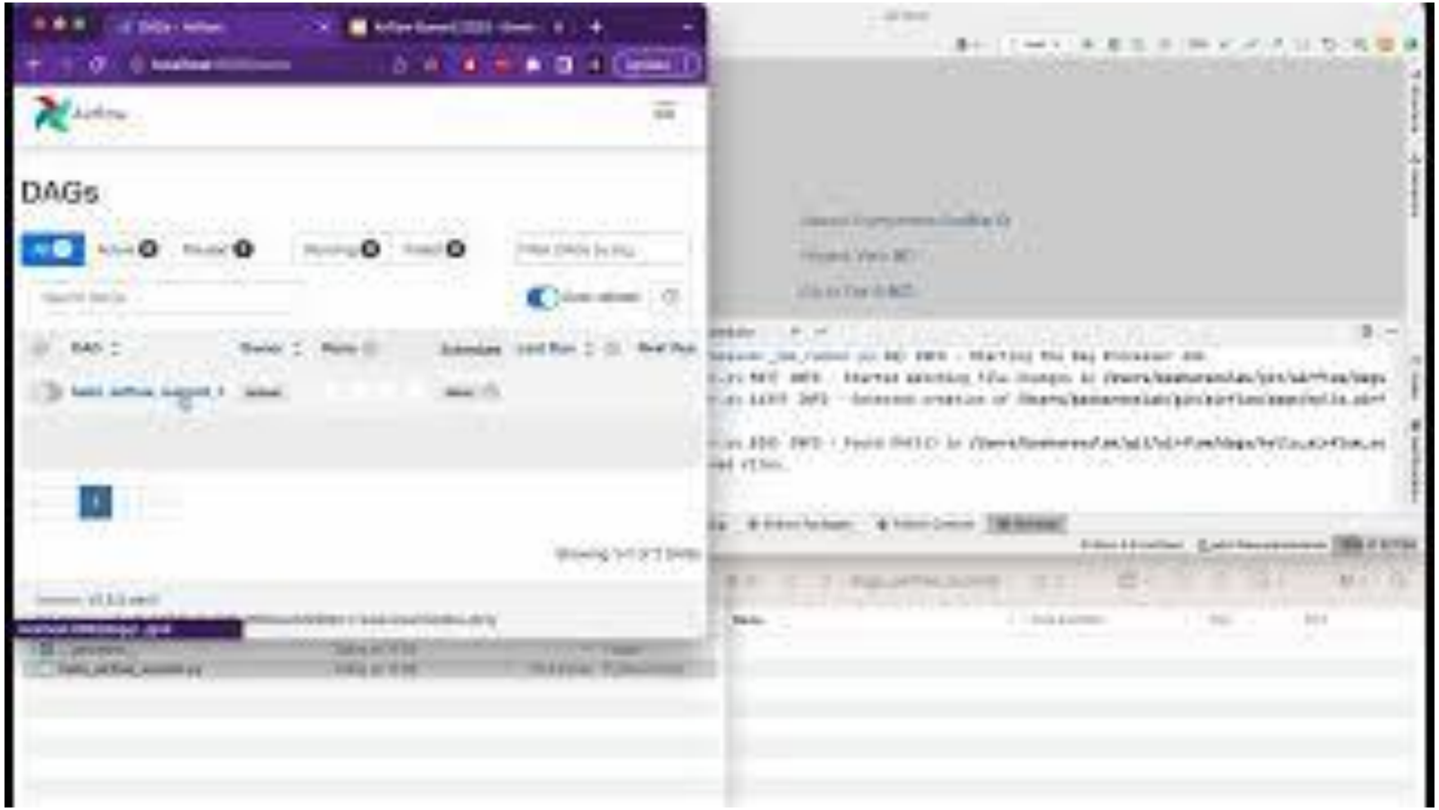
Biggest difference with current Airflow DAG parser:

- The current DAG parser implementation scans *ALL* files every 5 minutes, whereas with event-based parsing you only watch for changes
- Implemented new create/move/(re)process/delete methods to work on individual files

Demo

1. Create file
2. Modify file
3. Delete file

Demo



Waiting for new DAGs – Speedup

Before

Task name	Duration (seconds)	60	120	180	240	300	360
Wait for Airflow to start refresh_dag_dir()	300	🟦	🟦	🟦	🟦	🟦	
prepare_file_path_queue()	Negligible - 30						🟦
start_new_processes()	Negligible						🟦
Process file	Negligible						🟦
Refresh webserver	Wait for user to press F5						🟦

After

Task name	Duration (seconds)	60	120	180	240	300	360
Check file for DAG	Negligible	🟦					
start_new_processes()	Negligible	🟦					
Process file	Negligible	🟦					
Refresh webserver	Auto refresh period	🟦					

Speedup of 0-5 minutes for adding a new DAG file

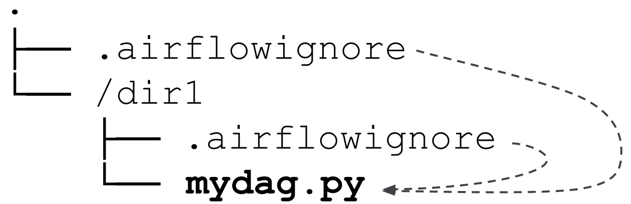
ASTRONOMER

Event-based parsing & .airflowignore

Say we added a new DAG file,
but the filepath is ignored by an .airflowignore pattern?

Challenge:

Watchdog triggers on creation of a new DAG file, but .airflowignore files are not modified, thus not triggering any event. However, the .airflowignore patterns do apply to the triggered files:



```
.
├── .airflowignore
└── /dir1
    ├── .airflowignore
    └── mydag.py
```

ASTRONOMER

Implementation change

All methods in current DAG parser implementation involve scanning complete directories, while event-based DAG parsing works on individual files.

1. Get all .airflowignore patterns
2. Filter filepaths
3. Scan files for DAGs

Handling .airflowignore patterns

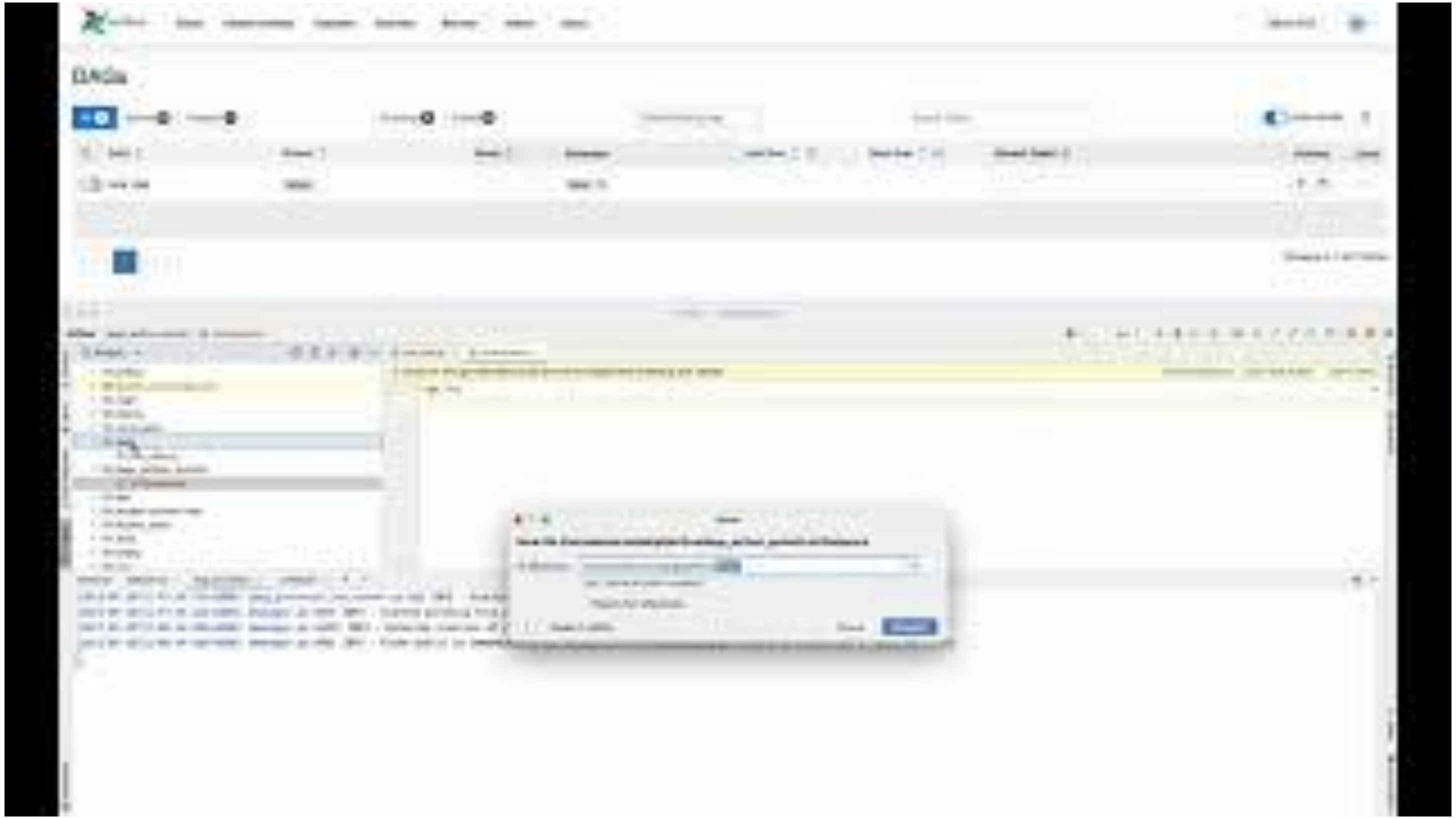
```
/.  (DAG folder root)
├── .airflowignore
├── /dir1
│   └── /nested_dir
│       ├── .airflowignore
│       ├── new_dag.py
│       └── dag_b.py
├── /dir2
│   ├── .airflowignore
│   └── dag_a.py
└── mydag.py
```

Instead of iterating ALL .airflowignore files, with event-based DAG parsing select only relevant .airflowignore files between the current folder & root.

So for **/dir1/nested_dir/new_dag.py:**

- **/.airflowignore**
- **/dir1/nested_dir/.airflowignore**
- /dir2/.airflowignore

.airflowignore demo



All covered scenarios

ON_CREATED

1. If .py/.zip extension -> handle created file
2. Elif .airflowignore extension -> handle created airflowignore file

ON_MOVED

3. If .py/.zip extension -> handle moved file
4. Elif .airflowignore extension -> handle moved airflowignore file

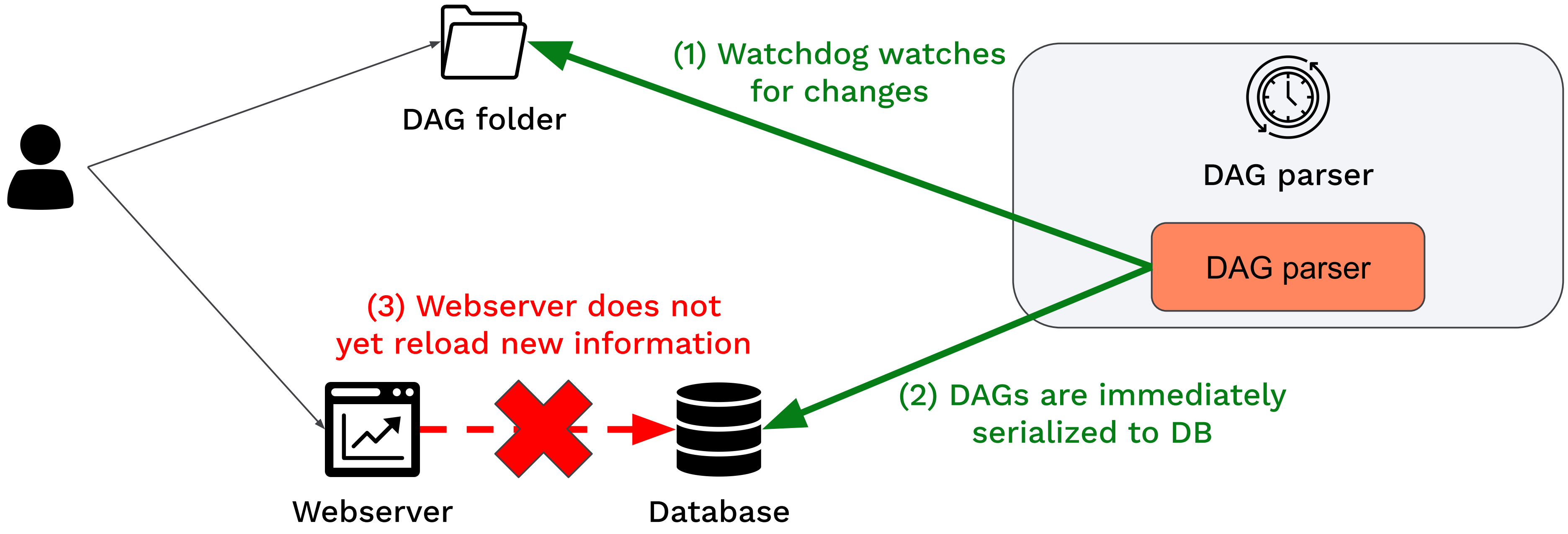
ON_DELETED

5. If .py/.zip extension -> handle deleted file
6. Elif .airflowignore extension -> handle deleted airflowignore file

ON_MODIFIED

7. If .py/.zip extension -> handle modified file
8. Elif .airflowignore extension -> handle modified airflowignore file

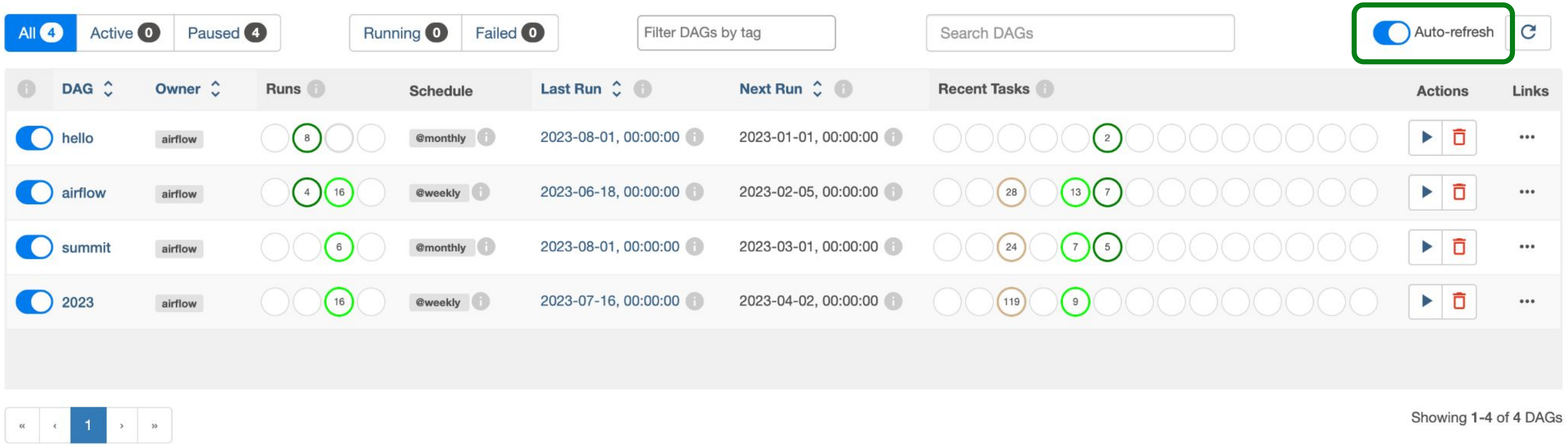
Refreshing the webserver



(1) Watchdog watches for changes
DAG folder
DAG parser
DAG parser
(3) Webserver does not yet reload new information
(2) DAGs are immediately serialized to DB
Webserver
Database

ASTRONOMER

Refreshing the webserver



All 4 | Active 0 | Paused 4 | Running 0 | Failed 0 | Filter DAGs by tag | Search DAGs | Auto-refresh

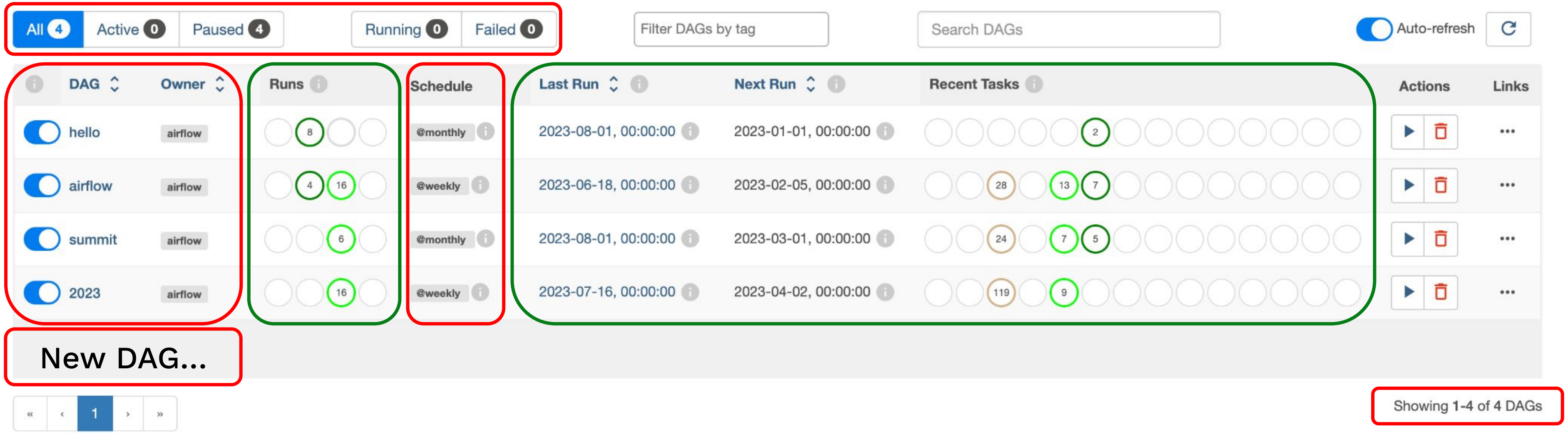
DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions	Links
hello	airflow	8	@monthly	2023-08-01, 00:00:00	2023-01-01, 00:00:00	2		...
airflow	airflow	4 16	@weekly	2023-06-18, 00:00:00	2023-02-05, 00:00:00	28 13 7		...
summit	airflow	6	@monthly	2023-08-01, 00:00:00	2023-03-01, 00:00:00	24 7 5		...
2023	airflow	16	@weekly	2023-07-16, 00:00:00	2023-04-02, 00:00:00	119 9		...

« ‹ 1 › »

Showing 1-4 of 4 DAGs

Auto-refresh does not refresh everything yet!

Refreshing the webserver



All 4 | Active 0 | Paused 4 | Running 0 | Failed 0

Filter DAGs by tag | Search DAGs | Auto-refresh

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions	Links
hello	airflow	8	@monthly	2023-08-01, 00:00:00	2023-01-01, 00:00:00	2		...
airflow	airflow	4 16	@weekly	2023-06-18, 00:00:00	2023-02-05, 00:00:00	28 13 7		...
summit	airflow	6	@monthly	2023-08-01, 00:00:00	2023-03-01, 00:00:00	24 7 5		...
2023	airflow	16	@weekly	2023-07-16, 00:00:00	2023-04-02, 00:00:00	119 9		...

New DAG...

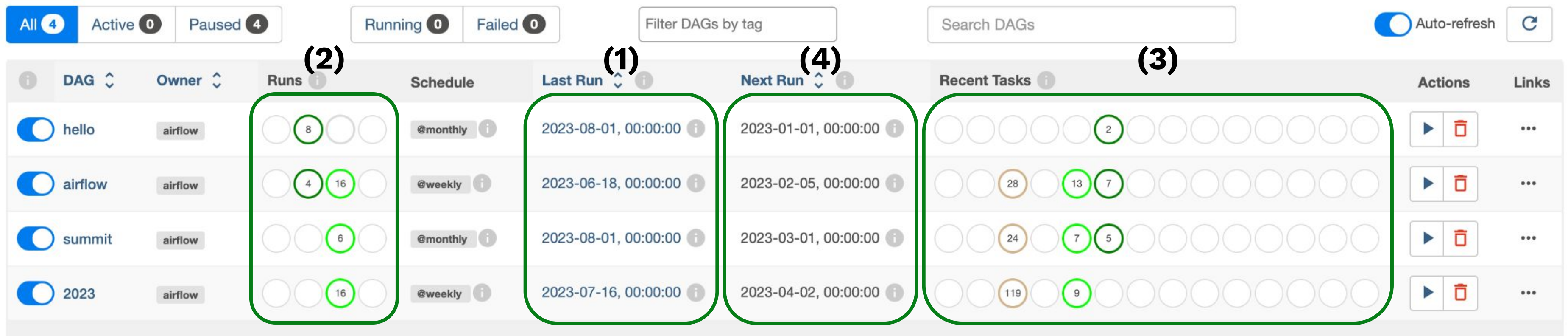
« ‹ 1 › »

Showing 1-4 of 4 DAGs

Refreshing the webserver

Current auto-refresh does 4 things. For each *active* (= not paused) DAG:

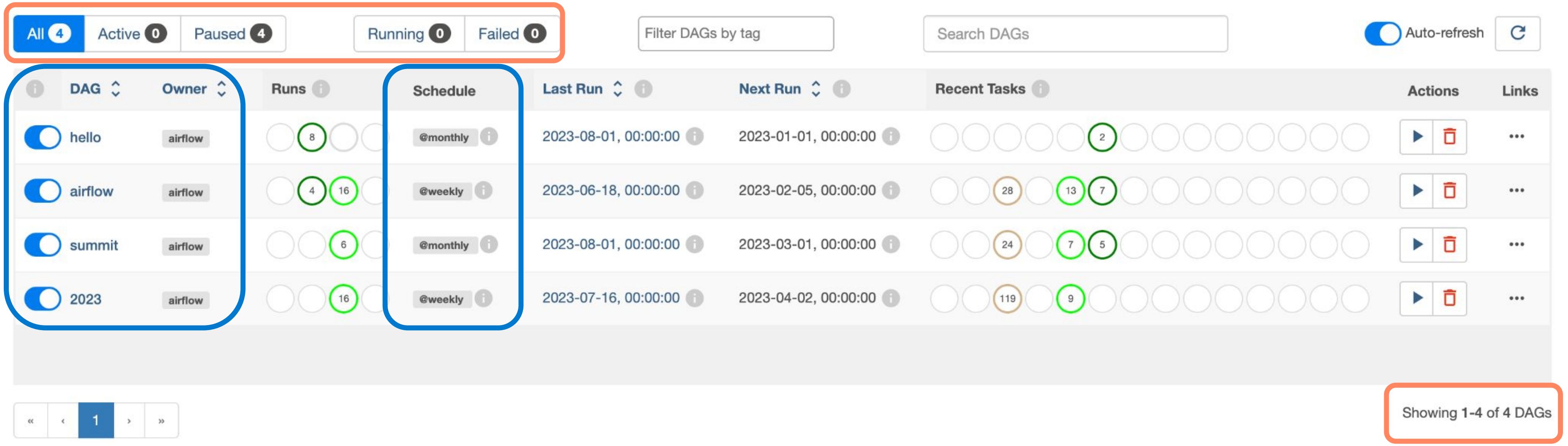
1. Update the execution date of the last DAG run
2. Refresh DAG run statistics
3. Refresh task instance statistics
4. Refresh “Next Run” information



DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions	Links
hello	airflow	8	@monthly	2023-08-01, 00:00:00	2023-01-01, 00:00:00	2		•••
airflow	airflow	4 16	@weekly	2023-06-18, 00:00:00	2023-02-05, 00:00:00	28 13 7		•••
summit	airflow	6	@monthly	2023-08-01, 00:00:00	2023-03-01, 00:00:00	24 7 5		•••
2023	airflow	16	@weekly	2023-07-16, 00:00:00	2023-04-02, 00:00:00	119 9		•••

Refreshing the webserver

Implemented two new endpoints:



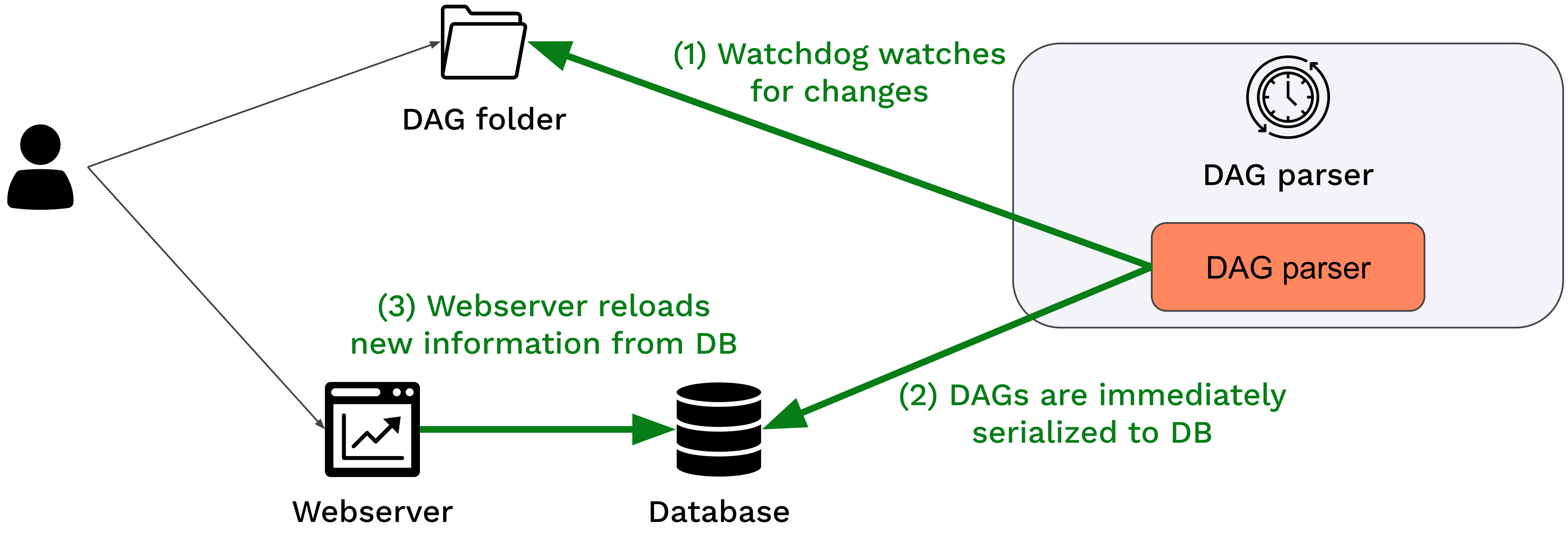
All 4 | Active 0 | Paused 4 | Running 0 | Failed 0 | Filter DAGs by tag | Search DAGs | Auto-refresh

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions	Links
hello	airflow	8	@monthly	2023-08-01, 00:00:00	2023-01-01, 00:00:00	2		•••
airflow	airflow	4 16	@weekly	2023-06-18, 00:00:00	2023-02-05, 00:00:00	28 13 7		•••
summit	airflow	6	@monthly	2023-08-01, 00:00:00	2023-03-01, 00:00:00	24 7 5		•••
2023	airflow	16	@weekly	2023-07-16, 00:00:00	2023-04-02, 00:00:00	119 9		•••

« ‹ 1 › »

Showing 1-4 of 4 DAGs

Refreshing the webserver


DAG folder
(1) Watchdog watches for changes
DAG parser
DAG parser
(3) Webserver reloads new information from DB
(2) DAGs are immediately serialized to DB
Webserver
Database

Future work

1. Get it into Airflow source code!
   - https://github.com/apache/airflow/pull/34487
2. Disentangle SLA processing logic from DAG processing
3. Remove delays in webserver by replacing polling with e.g. WebSockets or WebTransport

Known limitations

Some platforms (MacOS/BSD) require a raise in **ulimits** when the number of files in your DAGs folder is large (>256).

Known challenges

- Dynamic DAGs
  - Non-changing DAG code depending on external factors, e.g. list of S3 objects

- Operating systems trigger multiple events for certain single user activities. For example MacOS “copies” a file by first creating an empty file (generating ON_CREATE event) and then modifying the file to add all content (generating ON_MODIFIED event).
  - Implement debounce behaviour

Conclusion

- Current DAG parser implementation for periodically reparsing the complete DAGs folder is simpler, however this introduces a waiting time for the user
- Event-based DAG parsing enables immediate updates for the user, but implementation comes with complexities such as inter-file dependencies.

Thank you!

Questions?