Airflow 3 - Detailed Technical Explanations

Slide 5: Major Changes

This document provides in-depth technical explanations for each of the six major changes presented in slide 5 of the Airflow 3 migration presentation.

■ 1. Auth Manager Architecture

Overview of the change

Airflow 3 introduces a complete overhaul of the authentication and authorization system. The Flask-AppBuilder (FAB) system that managed authentication in Airflow 2.x is completely removed and replaced by a new modular architecture called Auth Manager.

Problems with Flask-AppBuilder (Airflow 2.x)

• **Tight coupling**: FAB was tightly coupled to the Airflow web application, making it difficult to customize or replace the authentication system.

• **OIDC limitations**: OIDC support in FAB was basic and required workarounds for complex enterprise scenarios.

• **Maintenance complexity**: FAB brought many unnecessary features for Airflow, increasing the security attack surface.

New Auth Manager architecture

Auth Manager is an abstract interface that allows plugging in different authentication systems. Instead of a monolithic system, Airflow 3 defines a contract (interface) that any Auth Manager implementation must respect.

Key components:

• **BaseAuthManager**: Abstract base class defining methods that any Auth Manager must implement (is_logged_in, get_user, check_authorization, etc.).

• **FabAuthManager**: Default implementation maintaining FAB compatibility to facilitate migration.

• **Custom Auth Managers**: Ability to create custom implementations for specific needs (like our Keycloak Auth Manager).

Improved native OIDC support

The new system offers better support for modern authentication protocols:

• **JWT Token Management**: Native handling of JWT tokens with validation, automatic refresh, and expiration management.

• **Claims Mapping**: Flexible mapping of OIDC claims to Airflow roles and permissions.

- **Multi-tenant**: Improved support for multi-tenant architectures.

Enhanced RBAC

Role-based access control (RBAC) is more granular and performant:

- **Granular permissions**: New resource-level permissions (DAG, Connection, Variable, Pool) with specific actions (read, edit, delete, create).
- **Performance**: Optimized permission caching, reducing DB queries.
- **Audit trail**: Better logging of authorization actions for compliance.

Our Keycloak implementation

For DataProduct BNPP, we developed a Custom Keycloak Auth Manager that:

- Extracts and validates JWT tokens from Keycloak
- Maps Keycloak groups to Airflow roles
- Handles automatic token refresh before expiration
- Integrates with our Zero Trust infrastructure (Illumio)

■ 2. Scheduler Performance Improvements

Critical role of the Scheduler

The Scheduler is the heart of Airflow. It is responsible for:

- Parsing DAG files to detect changes
- Creating DagRuns and TaskInstances according to schedules
- Determining which tasks are ready to execute
- Submitting tasks to executors

Optimized Parallel Parsing

Airflow 2.x - Problems:

- DAG parsing was done in multiprocessing with Python subprocesses
- Significant overhead from process fork/spawn
- No memory sharing between processes
- High startup delay for each subprocess

Airflow 3 - Improvements:

- **Process pooling**: Reuse of parsing processes between cycles, eliminating repeated creation/destruction costs.
- **Incremental parsing**: Only modified DAGs are re-parsed (detection via file mtime).
- **Shared memory structures**: Use of shared data structures to reduce memory duplication.
- **Async I/O**: Asynchronous DB read/write operations during parsing.

Performance impact

- **40-60% reduction** in parsing time for environments with 100+ DAGs
- **30% decrease** in scheduler CPU usage
- **Better scalability**: Efficiently supports 500+ DAGs on a single scheduler instance

Reduced latency

The latency between when a task becomes eligible and its actual start has been reduced through:

- **Optimized DB queries**: Reduced number of queries and more efficient index usage.
- **Batching**: TaskInstances are created and updated in batches rather than individually.
- **Lock reduction**: Less DB lock contention through better locking strategy.

Better memory management

- **Optimized garbage collection**: Better GC cycle for temporary objects
- **Memory pooling**: Reuse of memory allocations
- **Leak fixes**: Fixed several memory leaks present in Airflow 2.x

■ 3. Modern User Interface

Migration to React

Airflow's web interface has been completely rewritten using React, replacing the old system based on Flask templates with vanilla JavaScript.

Technical architecture

- **Frontend SPA**: Single Page Application with React 18
- **State Management**: Using React Query for cache and API query management
- **Component Library**: Reusable components with Chakra UI
- **Build System**: Vite for fast build times and HMR (Hot Module Replacement)

Navigation improvements

- **Client-side routing**: Instant navigation without page reload (React Router)
- **Smart breadcrumbs**: Contextual navigation based on hierarchy
- **Global search**: Unified search for DAGs, tasks, and runs with auto-completion
- **Favorites and history**: Quick access to frequently viewed DAGs

Increased performance

- **Virtual scrolling**: Display of large lists (thousands of task instances) without lag
- **Lazy loading**: Progressive data loading on demand
- **Code splitting**: Only necessary JavaScript code is downloaded
- **Asset optimization**: Images and assets optimized for the web

New DAG visualizations

- **Enhanced Graph View**: Performant SVG rendering with smooth zoom/pan and collision detection
- **Interactive Gantt Chart**: Temporal visualization with filters and drill-down
- **Calendar View**: Calendar view of runs with status indicators
- **Task Duration heatmap**: Visual identification of slow tasks

Responsive Design

The interface now adapts to different screen sizes (desktop, tablet, mobile) through responsive design based on CSS Grid and Flexbox.

■ 4. Enhanced REST API

New endpoints

Airflow 3 adds many new endpoints for operations that previously required direct SQL queries or workarounds:

- **Bulk operations**: /dags/~/dagRuns/bulk for batch operations
- **Task logs streaming**: /dags/{dag_id}/dagRuns/{run_id}/taskInstances/{task_id}/logs/stream
- **Health checks**: /health detailed endpoint with component status
- **Metrics**: /metrics for Prometheus-compatible metrics
- **Import errors**: /importErrors to debug DAG parsing issues

Improved OpenAPI documentation

- **OpenAPI 3.1**: Complete and up-to-date specification
- **Interactive examples**: Swagger UI with ability to test endpoints
- **Detailed schemas**: All data models documented with examples
- **Code generation**: SDK client generation in different languages

GraphQL Support (Experimental)

Airflow 3 introduces experimental GraphQL support allowing clients to query precisely the data they need:

- **Flexible queries**: Retrieve only necessary fields
- **Nested resources**: Get DAG + runs + tasks in a single query
- **Subscriptions**: Real-time updates via WebSockets

Optimized pagination

The API now uses cursor-based pagination instead of offset-based:

- **Performance**: No expensive OFFSET in database
- **Consistency**: Results remain consistent even when data is added/deleted
- **Scalability**: Efficiently supports millions of records

API versioning

The API now includes an explicit versioning system:

- **URL versioning**: /api/v1/, /api/v2/
- **Deprecation warnings**: Headers indicating obsolete endpoints
- **Backward compatibility**: Support for multiple versions in parallel

■ 5. DAG Versioning

Problem in Airflow 2.x

Airflow 2.x did not maintain a history of DAG versions. If a DAG was modified, it was impossible to know exactly which version of the code executed a past DagRun. This posed several problems:

- **Difficult debugging**: Impossible to reproduce a bug if the DAG has changed
- **Limited audit**: No traceability of changes
- **Complex rollback**: Required manual Git manipulations

Versioning architecture in Airflow 3

Airflow 3 introduces a native versioning system that captures and stores DAG versions:

Capture mechanism

- **DAG code hash**: Each time a DAG is parsed, a SHA-256 hash of the code is calculated
- **Storage in DB**: The hash and complete code are stored in a 'dag_version' table
- **Reference in DagRun**: Each DagRun points to the DAG version that created it
- **Automatic detection**: The system automatically detects code changes

Data structure

The dag_version table contains:

- dag_id: DAG identifier
- version_number: Incremental version number
- version_hash: SHA-256 hash of the code
- dag_code: Complete Python code of the DAG
- created_at: Creation timestamp
- created_by: User or system that created the version

Change history

The web interface now displays:

- **Version timeline**: Chronological list of all versions
- **Diff view**: Visual comparison between two versions (colored diff)
- **Blame/attribution**: Who made which change and when
- **Impact analysis**: Which DagRuns use which version

Simplified rollback

Rollback becomes a simple operation via the UI or API:

- **One-click UI**: 'Restore to this version' button in the interface

- **API endpoint**: POST /dags/{dag_id}/versions/{version_number}/restore
- **Safe rollback**: Validation before restoration (no running runs)
- **Rollback history**: Rollbacks are themselves versioned

Improved traceability

Versioning significantly improves audit and compliance:

- **Compliance**: Proof of which code processed which data
- **Reproduction**: Ability to re-execute a DagRun with its exact version
- **Change tracking**: Complete audit trail of pipeline modifications

Git Integration (Optional)

While versioning is native in Airflow, it can integrate with Git:

- **Git commit tracking**: Link between Airflow version and Git commit
- **Annotations**: Ability to add commit messages to versions
- **CI/CD integration**: CI/CD pipelines can tag versions

■ 6. Python 3.12 Support

Evolution of Python compatibility

Airflow 3 marks an important step in Python support modernization:

- **Airflow 2.x**: Python 3.7, 3.8, 3.9, 3.10, 3.11
- **Airflow 3.x**: Python 3.9, 3.10, 3.11, 3.12 (3.13 in preview)
- **Minimum required**: Python 3.9
- **Recommended**: Python 3.12 for new installations

Python 3.12 performance improvements

Python 3.12 brings significant performance gains that Airflow benefits from directly:

- **15-25% faster**: General performance improvement vs Python 3.11
- **PEP 709**: Comprehension inlining (faster list/dict comprehensions)
- **Improved GC**: More efficient garbage collector
- **Faster startup**: Python startup time reduced by 10-15%

New language features

Python 3.12 introduces features that DAGs can now use:

- **PEP 695 - Type Parameter Syntax**: Simplified syntax for generics

```
# Old (Python < 3.12) from typing import TypeVar, Generic T = TypeVar('T') class
Stack(Generic[T]): ... # New (Python 3.12+) class Stack[T]: ...
```

- **PEP 698 - Override decorator**: Validation of method overrides

```
from typing import override class CustomOperator(BaseOperator): @override def
execute(self, context): # Verified at runtime ...
```

- **PEP 701 - Enhanced f-strings**: More powerful f-strings with nested quotes

Improved security

- **CVE fixes**: Python 3.12 fixes several vulnerabilities present in 3.8/3.9
- **hashlib hardening**: More secure hash algorithms by default
- **SSL/TLS updates**: Support for latest TLS protocols

Modernized dependencies

With Python 3.12, Airflow can use more recent versions of its dependencies:

- **SQLAlchemy 2.0**: Better performance and modernized API
- **Pydantic 2.x**: Much faster data validation
- **Recent FastAPI**: If used for custom endpoints
- **Modern NumPy/Pandas**: Better performance for data processing

Impact on our DAGs

For DataProduct BNPP, migration to Python 3.12 offers:

- **Performance**: DAGs with heavy Python logic run faster
- **Maintenance**: More modern and easier to maintain code
- **Libraries**: Access to latest versions of Python libraries
- **Long-term support**: Python 3.12 supported until October 2028

Recommended migration

While Airflow 3 supports Python 3.9+, we recommend Python 3.12 for:

- New Airflow instances
- Environments where performance is critical
- Projects planned for 2+ years

Conclusion

These six major changes in Airflow 3 represent a significant evolution of the platform. Each improvement has been designed to address real limitations of Airflow 2.x and prepare the platform for future data orchestration needs. For DataProduct BNPP, these improvements translate into a more performant, secure, and maintainable infrastructure. Our investment in the migration will be quickly offset by gains in operational efficiency and developer experience.