Democratizing ML feature store framework at scale

Rafay Aleem & Victoria Varney
Rafay Aleem  
Faire  
Sr Data Engineer - ML platform

Victoria Varney  
Astronomer  
Sr Customer Success Manager
Today we are talking about...

Democratizing a machine learning feature store framework at Faire

How we have enabled everyone to contribute to a shared resource easily

Building a framework on top of Airflow by leveraging its low-level APIs

How Airflow is much more than just a workflow management tool
Search experience at Faire
Wholesale Food & Drink
Shop independent brands from around the world at wholesale prices.

FREEZE DRIED JOLLY BALLS (JOLLY RANCHERS)
MSRP: $0.95

1 oz Sea Salt Chips (case of 20 bags)
MSRP: $49.15

SAVORY PARTY CRACKER SEASONING - CLASSIC...
MSRP: $0.93

FREEZE DRIED RAINBOW BITES
MSRP: $0.23

SIMPLY MINTS - PEPPERMINT
MSRP: $0.08

Upscale Freeze
1 in 6 Snacks - Carolina...
Saw Good Inc
Simply Gum
Anatomy of a search

Query (t-shirts) product:query

Retrieval

Candidates (identifiers) p_pkjmad9rhxt: t shirts

Real-time rankers

Fetch features for candidates online feature-store

Model

Ranking

Search results
Anatomy of a search

Query (t-shirts)

Product index

Retrieval

Candidates (identifiers)

Real-time rankers

Fetch features for candidates

on feature-store

Model

Ranking

Search results
Enhancing the search experience

- **Unlock** the science behind search relevancy

- Search is **particularly important** in context of e-Commerce
Enhancing the search experience

- Features are float values with some underlying meaning such as:
  - num_countries_sold_in
  - avg_fulfillment_days_trailing_7_days

- Features are engineered and defined on the offline and eventually propagated to online feature store.
The need for democratizing the feature store

- Feature and model exploration
- Custom Airflow ETLs for feature engineering
- Model training
- Backfill for model training
The need for democratizing the feature store

- No **feature visibility** across teams

- No clear process to separate **online features** from offline

- Non-standardized **error-prone backfilling** with no clear notion of point-in-time joins

- Overhead **costs** on ElasticCache and Snowflake
The need for democratizing the feature store

The Problem

Consolidate all feature definitions into a single feature store with a complete feature registry

Complex & Error-prone backfills

Simplify and standardize feature backfilling

SQL + Custom ETLs

Plain SQL as feature definitions
Feature Store Framework

- All features are defined as SQL with their Python configurations

- Each feature can be configured separately with its own metadata

Examples
- specify if available online
- feature description
- author

- All configuration is fed into a queryable feature registry
Feature Store Framework

```python
# dataclass
class Feature:
    sql_feature_name: str
    feature_description: str = "No description has been specified"
    available_online: bool = False

# dataclass
class FeatureTask:
    sql_file: str
    features: list[Feature]
    entity: EntityDescription
    author: str = "Unknown author"
    is_static: bool = False

# backfill parameters
backfill_config: FeatureTaskBackfillConfig = None
```

High-level config dataclasses

Low-level APIs
Feature Store Framework

# This step creates individual tables for each invocation of MLFeatureSnowflakeOperator
# You can find those tables in `{schema_name}.features_{feature_group_name}`

```python
with TaskGroup(group_id='compute_offline_features') as compute_offline_features:
    for feature_entity in FEATURE_ENTITIES:
        feature_entity.get_dynamically_mapped_tasks(config)
```

# This is a dummy task used to make it easier to manage the DAG in the Airflow UI
compute_offline_features_dummy_task = EmptyOperator(
    task_id='compute_offline_features_dummy_task',
    doc='Dummy task to trigger offline features computation',
    trigger_rule='all_done',
)

# This pulls the daily feature used for dispersal and pivoting.
# Data is persisted to avoid duplicated computation and facilitate data checking.
update_daily_features = AmendTableOperator(
    task_id='update_daily_features',
    schema_name=config.get('schema'),
    table_name=config.get('features_daily_table_name'),
    create_sql='sql/create_features_daily_table.sql',
    sql='sql/update_features_daily_table.sql',
    fill_condition='ds = {{ ds }}',
    data_checks=[
        'checks/(check).sql'
        for check in [
            'update_daily_features_task_loads_data',
        ]
    ]
)
Adopting new Airflow features

```python
# We use partial and expand_kwsargs here to optimize dag parsing and delay task generation to runtime
return MLFeatureShowflakeOperator.partial(
    task_id="{}-{}-{}",
    start_date=None,
    end_date=None,
    is_backfill=True,
    expand_kwsargs=kwsargs
)
```

Use partial and expand_kwsargs to dynamically map existing task
Adopting new Airflow features

- Be proactive in adopting new versions of Airflow
- New features have greatly improved cluster performance
- Notable features:
  - Dynamic Task Mapping
  - Task Groups (easier visual dag management)
  - Deferrable operators for long running tasks such as AWS Batch jobs
Extending the framework for feature backfills

```python
FeatureTask(
    sql_file="brand_contact_book.sql",
    features=[
        Feature(
            sql_feature_name="number_of_contact_books_found",
            feature_description="Number of retailers with this brand in their address book",
            available_online=False,
            backfill_config=FeatureTaskBackfillConfig(
                start_date=datetime(2023, 8, 1),
                end_date=datetime(2023, 8, 30),
                namespace="brand_i_month_backfill",
            ),
        ),
    ],
    entity=BrandEntity(),
    author="rafae",
    is_static=False,
)
```

- Uses same SQL files
- Provides extra Jinja templated flag `{{ is_backfill }}`
- Entire process takes 3 lines of configuration!
Best practices for designing extensible frameworks

- Use mixins to extend operator capabilities

```python
class RebuildTableSnowflakeOperator(
    SnapshotMixin, TableOperatorMixin, ETLSnowflakeOperator
):
    # Extend operator capabilities using mixins

class SnapshotConfig:
    Frequency = Enum('Frequency', ['NEVER', 'DAILY', 'EVERY_RUN'])

store_daily_embeddings = RebuildTableSnowflakeOperator(
    task_id="store_daily_embeddings",
    sql=daily_online_embedding_sql(),
    schema_name=config.get("Schema"),
    table_name="daily_embeddings_v2",
    snapshot_frequency=SnapshotConfig.Frequency.EVERY_RUN,
)
```

- Allows adding new features to low-level APIs without breaking
Airflow beyond workflow management

Airflow goes far and beyond a workflow management tool.

It’s thoughtful design makes it very extensible and powerful.

It is very well suited for running mission critical workflows with tight SLA requirements.

It continues to be a very stable part of Faire’s infrastructure and continue to scale.
Key Takeaways

- Proactively consider onboarding to latest Airflow versions
- Don’t think of Airflow as just a workflow management tool
- Consider building shared frameworks instead of shared ETLs
- Thoughtful use of Airflow APIs and features goes a long way
Credits

- **Wayne Zhang** for his guidance on the offline feature store framework

- **Analytics Engineering** team for their feedback on table snapshot tooling

- **Core Data Infra** team for their constant support with Airflow and Snowflake

- **Machine Learning Platform** team for dealing with on-call issues and providing stakeholder support

- My wife’s constant support
Wrap Up & Questions

- Careers @ Faire: faire.com/careers/

- Where to find Rafay
  - LinkedIn: linkedin.com/in/mrafayaleem/
  - Airflow Slack: @Rafay Aleem
  - Twitter: @mrafayaleem
  - Email: contact@mrafayaleem.com
  - Newsletter: mrafayaleem.com/#/portal/

Thank You!