Bridging Real-Time and Batch: Declarative Feature Engineering with Apache Hamilton + Narwhals

Ryan Whitten

Director, ML Data Engineering

Best Egg

Organized by HOPSWORKS

Quick Intro

Me:

- 10 years in data & ML space
- Best Egg – 5 years
- 2 years – built internal ML Feature Platform
- Contributor & committer to Apache Hamilton
- Contributor to Narwhals (Ibis backend)

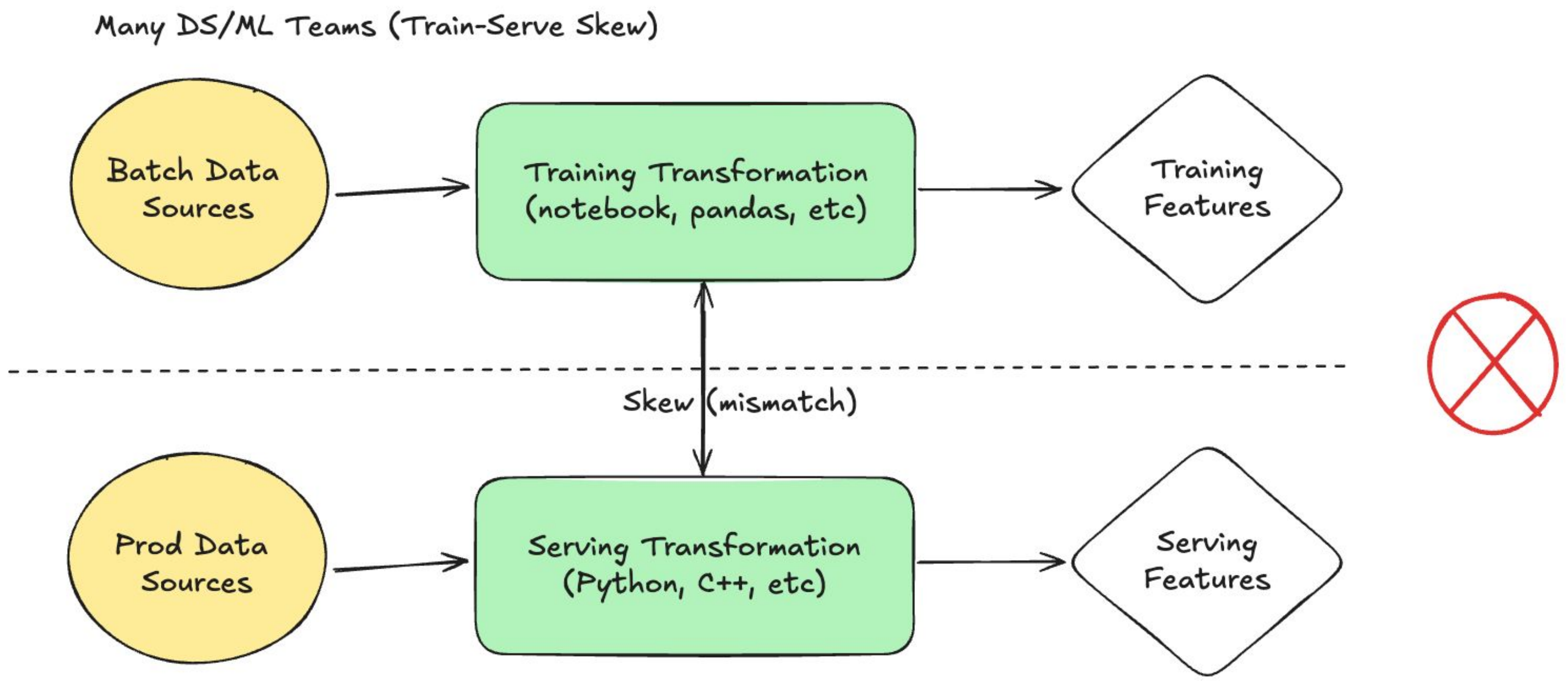
Best Egg

- Founded 2014
- Personal loans
- $21B+ funded
- 1.1M+ loans
- ML & features are core to underwriting

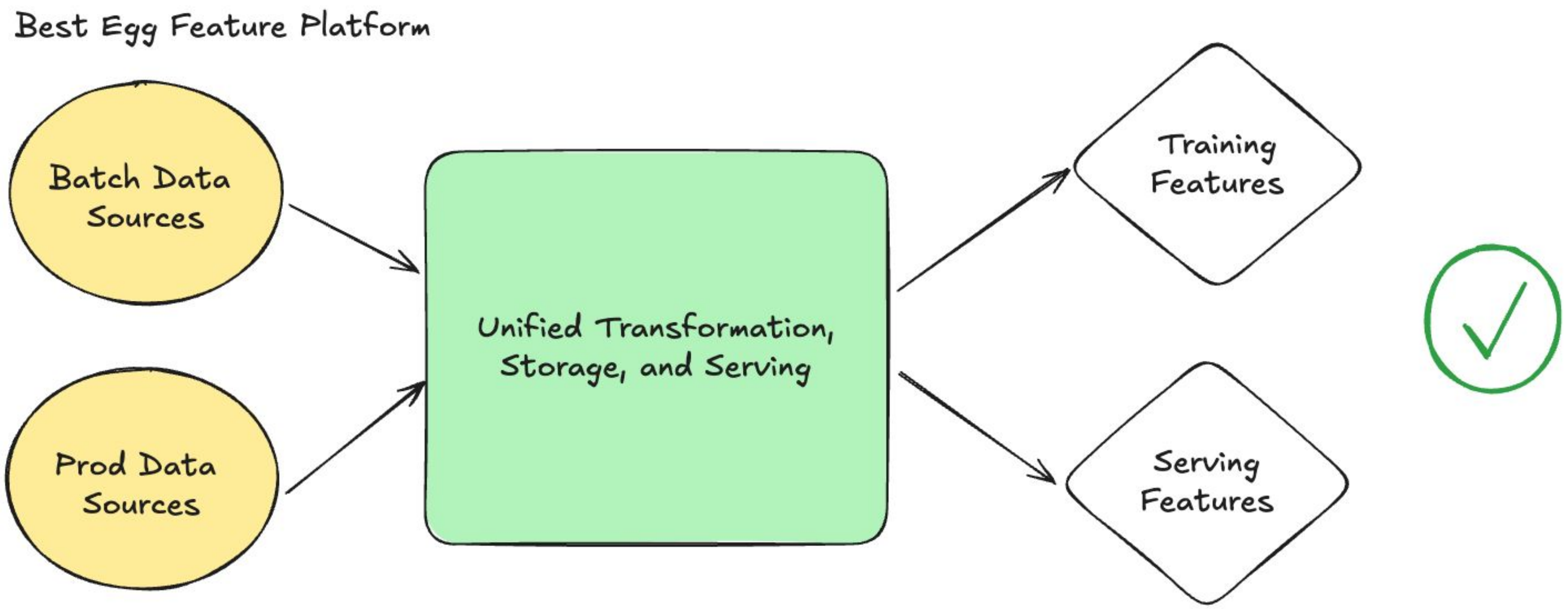
Backfilling Real-Time Features is Hard

- Real-time feature engineering looks simple ▯ until you need 5–10 years of history for training
- Log-and-wait ▯ delayed value from newly created features
- Python/pandas/polars ▯ memory & scale limits on billions of rows, especially when joins are involved
- Separate batch pipelines often rewrite logic ▯ train-serve skew

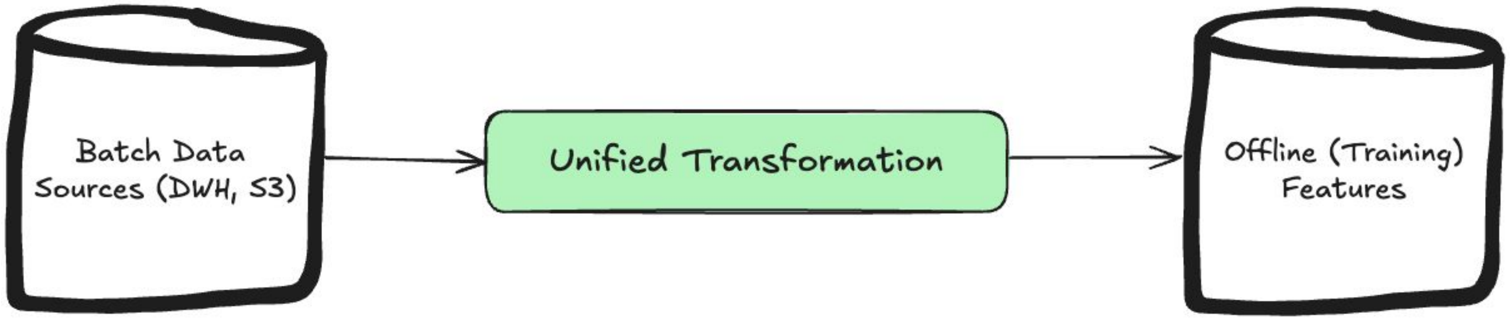
Separate pipelines is not the answer


Many DS/ML Teams (Train-Serve Skew)
Batch Data Sources
Training Transformation (notebook, pandas, etc)
Training Features
Skew (mismatch)
Prod Data Sources
Serving Transformation (Python, C++, etc)
Serving Features

Unify as much as possible, especially transformation


Best Egg Feature Platform
Batch Data Sources
Prod Data Sources
Unified Transformation, Storage, and Serving
Training Features
Serving Features

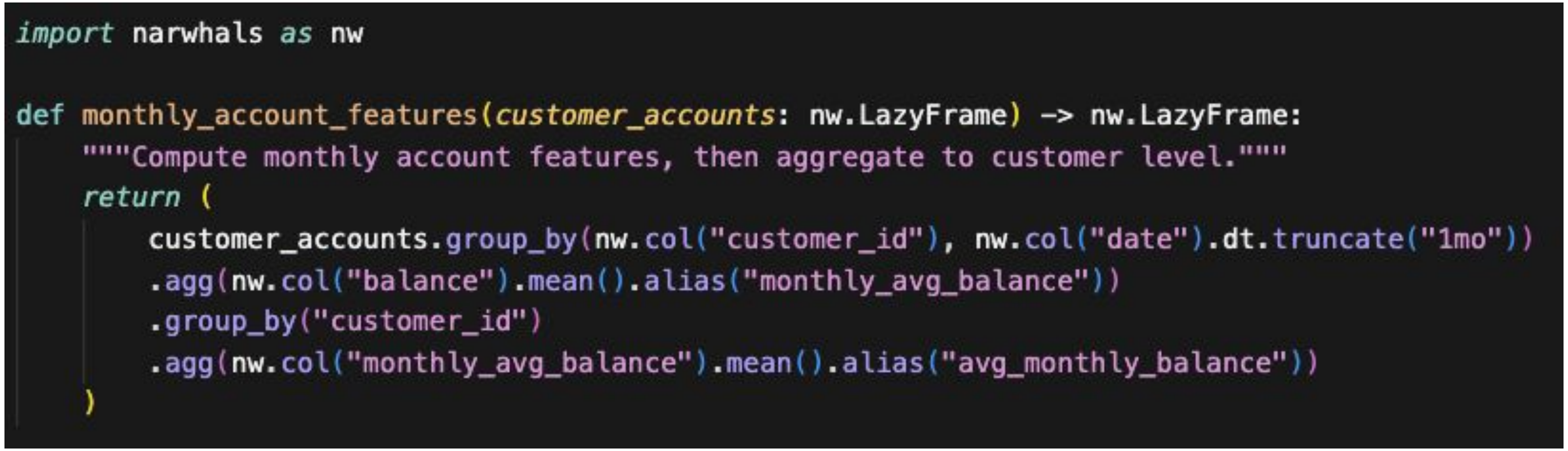
Change the IO for backfills, not transformation



Transformation needs to be a DSL, but a very flexible one...

Narwhals: One API, many DataFrames

- Pandas, Polars, Ibis ... fragmentation everywhere
- Narwhals = common **expression** layer using subset of the Polars API
- Write transformations once, run across different engines (and stay lazy)



```python
import narwhals as nw

def monthly_account_features(customer_accounts: nw.LazyFrame) -> nw.LazyFrame:
    """Compute monthly account features, then aggregate to customer level."""
    return (
        customer_accounts.group_by(nw.col("customer_id"), nw.col("date").dt.truncate("1mo"))
        .agg(nw.col("balance").mean().alias("monthly_avg_balance"))
        .group_by("customer_id")
        .agg(nw.col("monthly_avg_balance").mean().alias("avg_monthly_balance"))
    )
```

Narwhals is flexible enough

All core transformations can be easily run in both Polars and engines that support SQL:

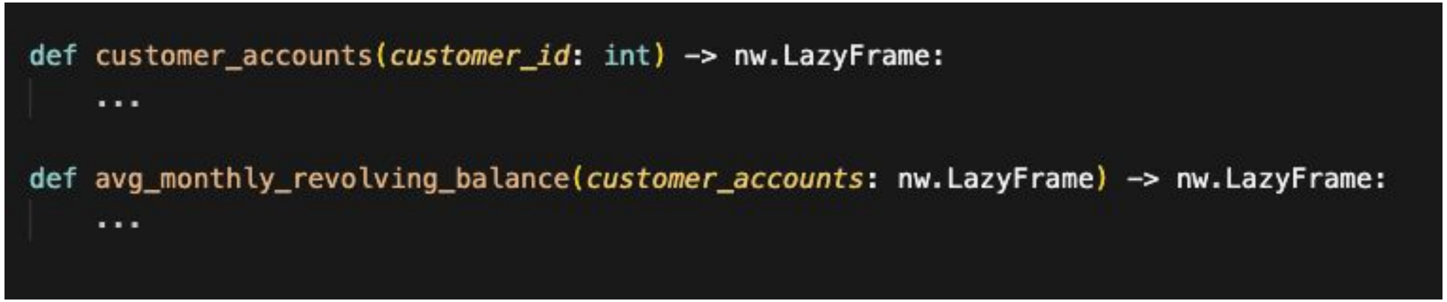
- Map (col → col): arithmetic, flags, conditions
- Filter: predicates, time ranges, null handling
- Join: left/inner/cross/semi/as-of/etc.
- Group by & aggregations: mean/sum/min/max/std/count/etc.
- Window functions: rolling/cumulative, lag/lead, range between (temporal)

Ibis is one Narwhals backend that can translate these expressions into ~20 different SQL engines, including:

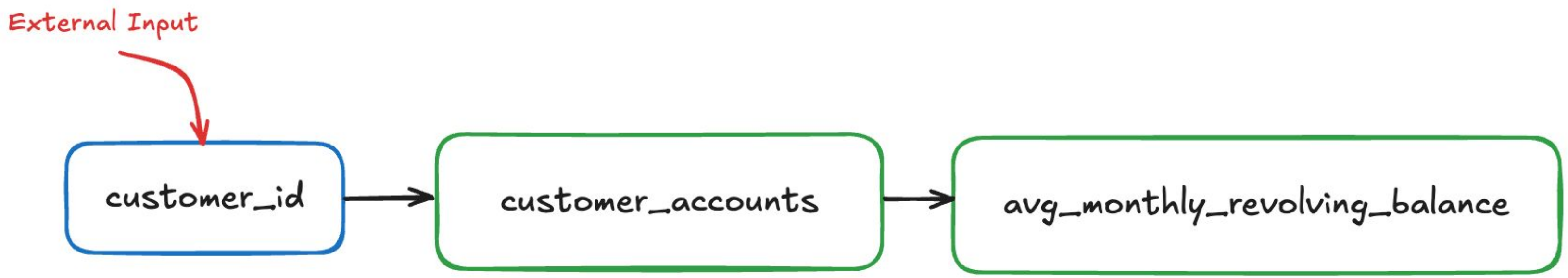
- Snowflake
- Databricks
- PySpark
- Flink
- Postgres
- Duckdb

Apache Hamilton: Declarative Feature DAGs

Lightweight Python DAGs → complex dependencies declared using input args

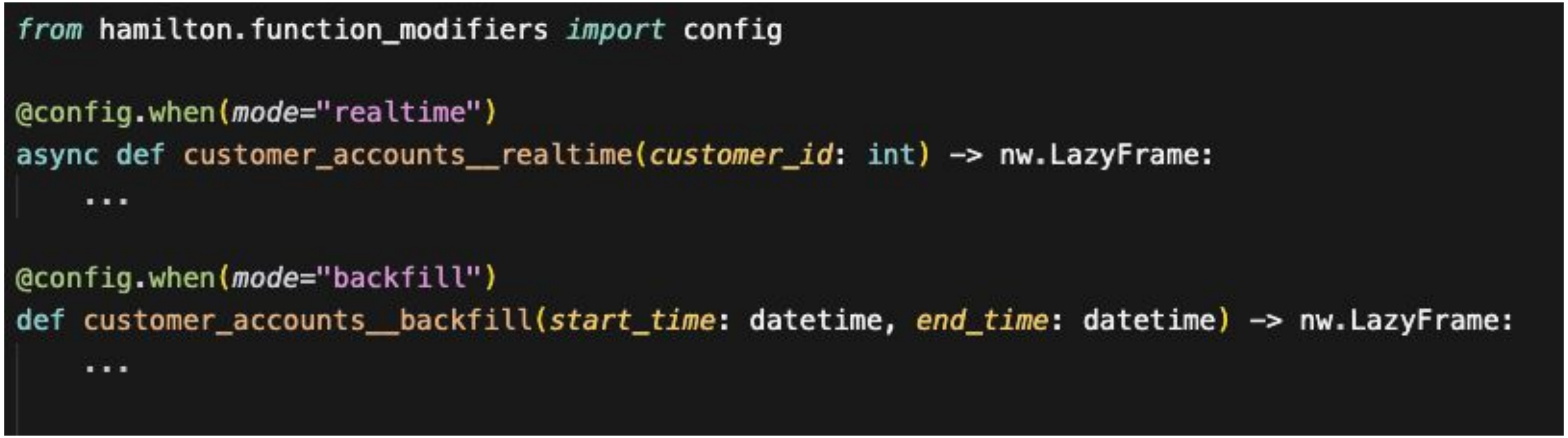


```python
def customer_accounts(customer_id: int) -> nw.LazyFrame:
    ...

def avg_monthly_revolving_balance(customer_accounts: nw.LazyFrame) -> nw.LazyFrame:
    ...
```



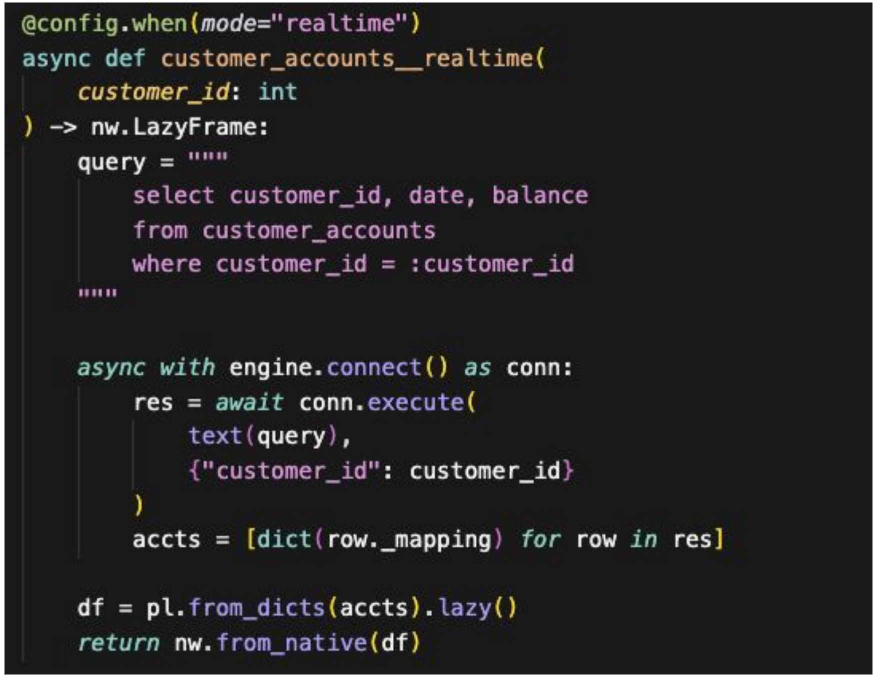
One Feature Definition → Two Execution Contexts

- @config.when → runtime switching of implementations depending on the run context
- Clear IO boundaries vs pure transforms — swap just the IO pieces for batch vs real-time

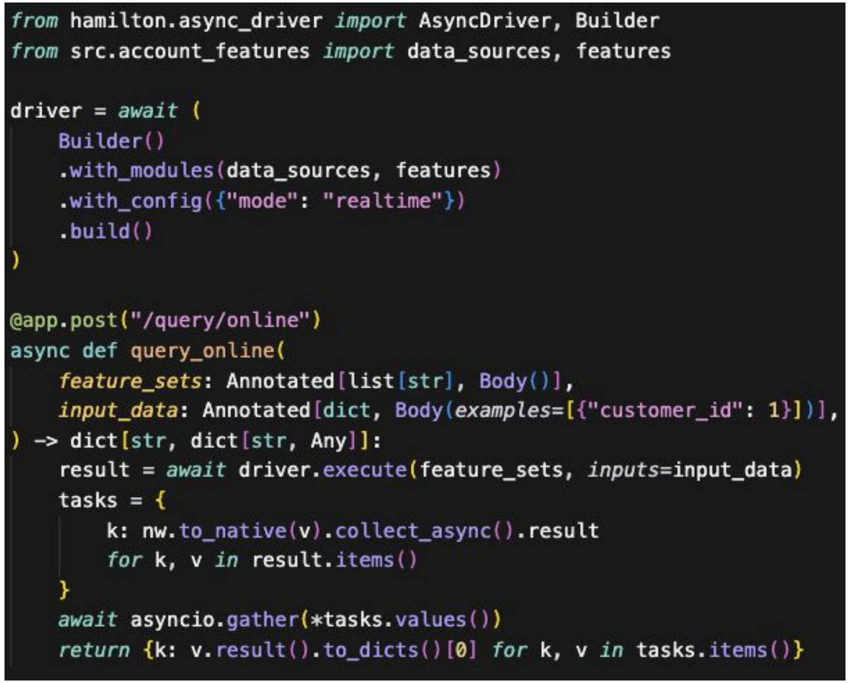


```python
from hamilton.function_modifiers import config

@config.when(mode="realtime")
async def customer_accounts__realtime(customer_id: int) -> nw.LazyFrame:
    ...

@config.when(mode="backfill")
def customer_accounts__backfill(start_time: datetime, end_time: datetime) -> nw.LazyFrame:
    ...
```

Real-Time Example (e.g., FastAPI)

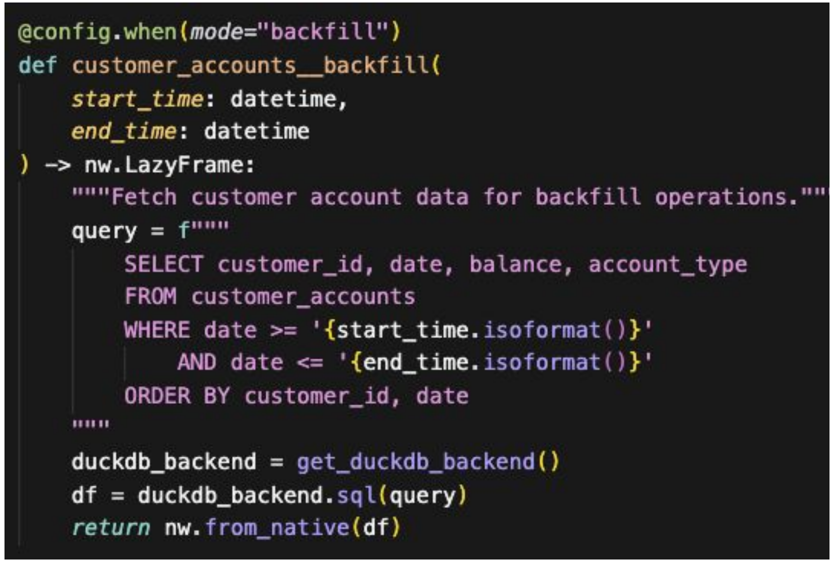


```python
@config.when(mode="realtime")
async def customer_accounts__realtime(
    customer_id: int
) -> nw.LazyFrame:
    query = """
        select customer_id, date, balance
        from customer_accounts
        where customer_id = :customer_id
    """

    async with engine.connect() as conn:
        res = await conn.execute(
            text(query),
            {"customer_id": customer_id}
        )
        accts = [dict(row._mapping) for row in res]

    df = pl.from_dicts(accts).lazy()
    return nw.from_native(df)
```

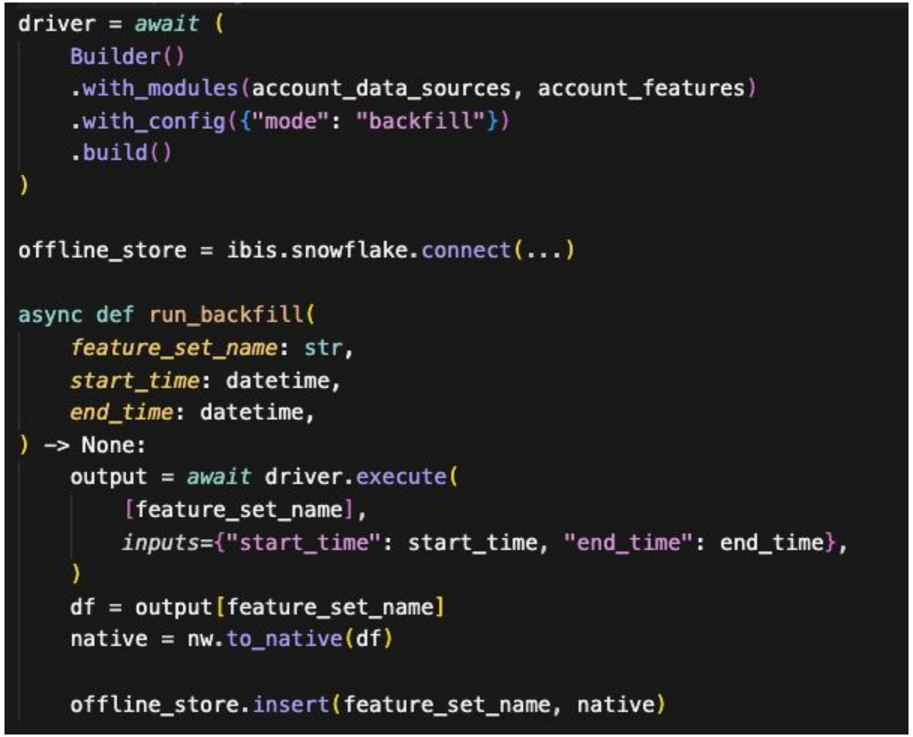


```python
from hamilton.async_driver import AsyncDriver, Builder
from src.account_features import data_sources, features

driver = await (
    Builder()
    .with_modules(data_sources, features)
    .with_config({"mode": "realtime"})
    .build()
)

@app.post("/query/online")
async def query_online(
    feature_sets: Annotated[list[str], Body()],
    input_data: Annotated[dict, Body(examples=[{"customer_id": 1}])],
) -> dict[str, dict[str, Any]]:
    result = await driver.execute(feature_sets, inputs=input_data)
    tasks = {
        k: nw.to_native(v).collect_async().result
        for k, v in result.items()
    }
    await asyncio.gather(*tasks.values())
    return {k: v.result().to_dicts()[0] for k, v in tasks.items()}
```

Backfill Example (e.g., K8s CronJob)

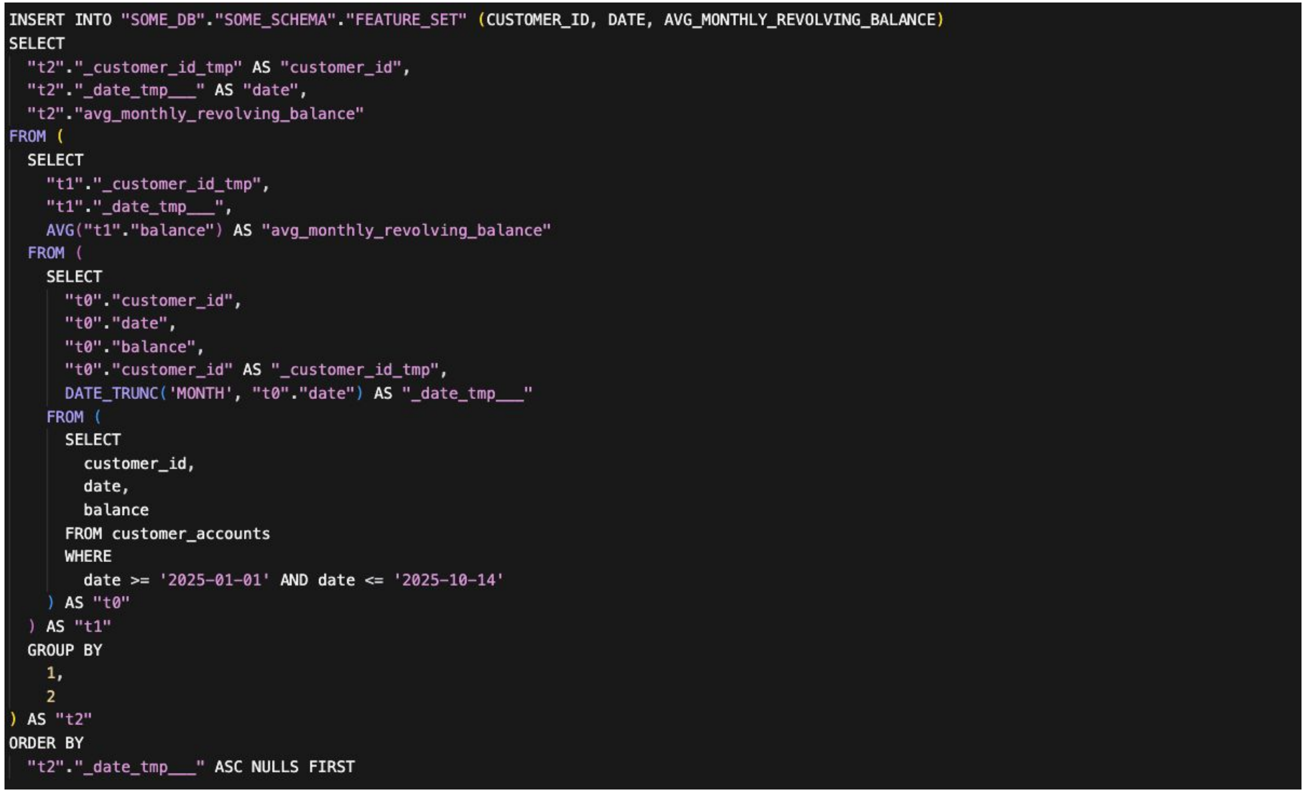


```python
@config.when(mode="backfill")
def customer_accounts__backfill(
    start_time: datetime,
    end_time: datetime
) -> nw.LazyFrame:
    """Fetch customer account data for backfill operations.""
    query = f"""
        SELECT customer_id, date, balance, account_type
        FROM customer_accounts
        WHERE date >= '{start_time.isoformat()}'
            AND date <= '{end_time.isoformat()}'
        ORDER BY customer_id, date
    """
    duckdb_backend = get_duckdb_backend()
    df = duckdb_backend.sql(query)
    return nw.from_native(df)
```



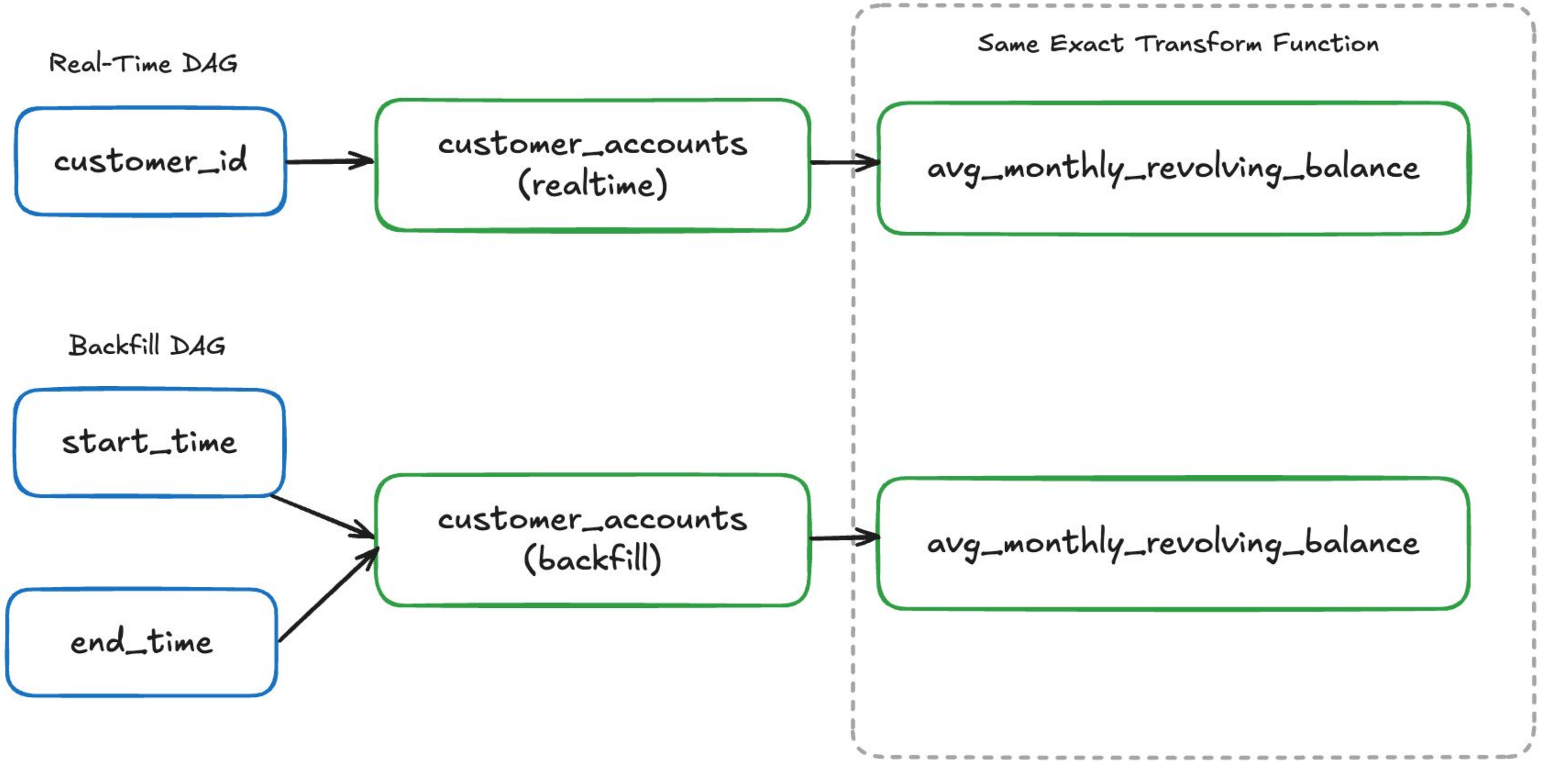
```python
driver = await (
    Builder()
    .with_modules(account_data_sources, account_features)
    .with_config({"mode": "backfill"})
    .build()
)

offline_store = ibis.snowflake.connect(...)

async def run_backfill(
    feature_set_name: str,
    start_time: datetime,
    end_time: datetime,
) -> None:
    output = await driver.execute(
        [feature_set_name],
        inputs={"start_time": start_time, "end_time": end_time},
    )
    df = output[feature_set_name]
    native = nw.to_native(df)

    offline_store.insert(feature_set_name, native)
```

Narwhals + Ibis = Dynamic SQL Pushed Down to DWH



```sql
INSERT INTO "SOME_DB"."SOME_SCHEMA"."FEATURE_SET" (CUSTOMER_ID, DATE, AVG_MONTHLY_REVOLVING_BALANCE)
SELECT
  "t2"."_customer_id_tmp" AS "customer_id",
  "t2"."_date_tmp___" AS "date",
  "t2"."avg_monthly_revolving_balance"
FROM (
  SELECT
    "t1"."_customer_id_tmp",
    "t1"."_date_tmp___",
    AVG("t1"."balance") AS "avg_monthly_revolving_balance"
  FROM (
    SELECT
      "t0"."customer_id",
      "t0"."date",
      "t0"."balance",
      "t0"."customer_id" AS "_customer_id_tmp",
      DATE_TRUNC('MONTH', "t0"."date") AS "_date_tmp___"
    FROM (
      SELECT
        customer_id,
        date,
        balance
      FROM customer_accounts
      WHERE
        date >= '2025-01-01' AND date <= '2025-10-14'
    ) AS "t0"
  ) AS "t1"
  GROUP BY
    1,
    2
) AS "t2"
ORDER BY
  "t2"."_date_tmp___" ASC NULLS FIRST
```

IO Changes, Features Don't


Real-Time DAG
customer_id
customer_accounts (realtime)
avg_monthly_revolving_balance
Backfill DAG
start_time
end_time
customer_accounts (backfill)
avg_monthly_revolving_balance
Same Exact Transform Function

Putting it all together

Narwhals Gives You:

- Powerful & flexible API for defining transformation (thanks, Polars!)
- Instant compatibility across most query engines (Ibis alone supports 20+ backends)
- Reusable expressions that avoid train-serve skew

Apache Hamilton Gives You:

- Powerful & flexible DAGs in pure Python
- Easy IO switching depending on run context (i.e., real-time or backfill, batch or stream, etc.)
- Dynamic selection of outputs at runtime
- Tons of other features (UI, caching, observability, etc.)

Lessons Learned

- Don’t backfill outside the warehouse if you can avoid it
- Declarative DAGs reduce code & pipeline duplication
- Having reusable transformations across query engines is the holy grail

Questions?

- https://github.com/apache/hamilton
- https://github.com/narwhals-dev/narwhals
- https://github.com/rwhitten577/feature-store-summit-2025

PS: I'm hiring! Reach out on Slack or LinkedIn for more info

Thank You