Driving Efficiency and Quality through a Feature Store at Delivery Hero

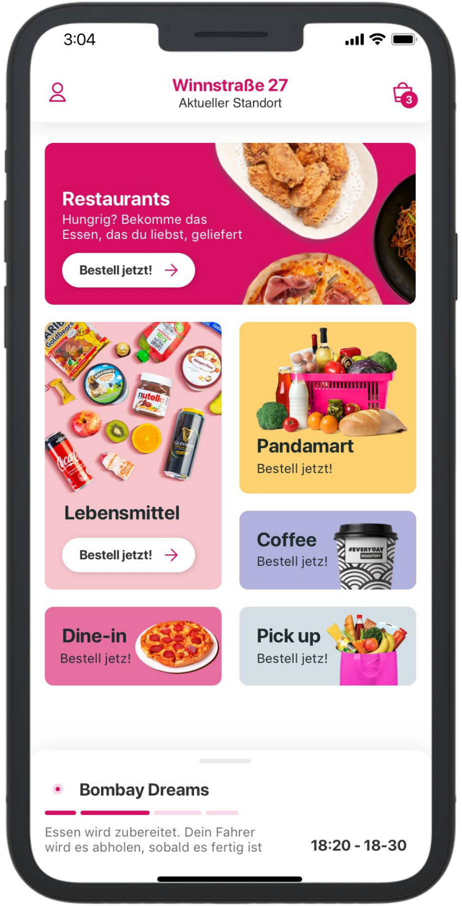
Breno Costa, Staff ML Engineer, Delivery Hero

FEATURE STORE SUMMIT 2023

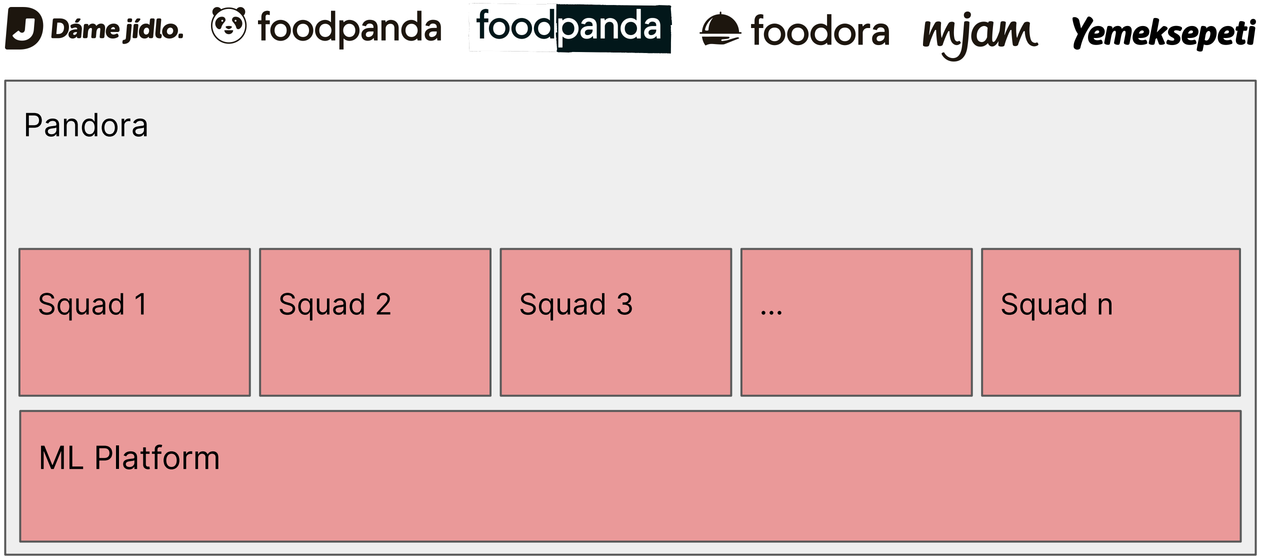
Organized by HOPSWORKS

Delivery Hero is present in over **70 countries across four continents**. We operate a wide range of local brands that are united behind our shared mission to always deliver an amazing experience - fast, easy, and to your door.

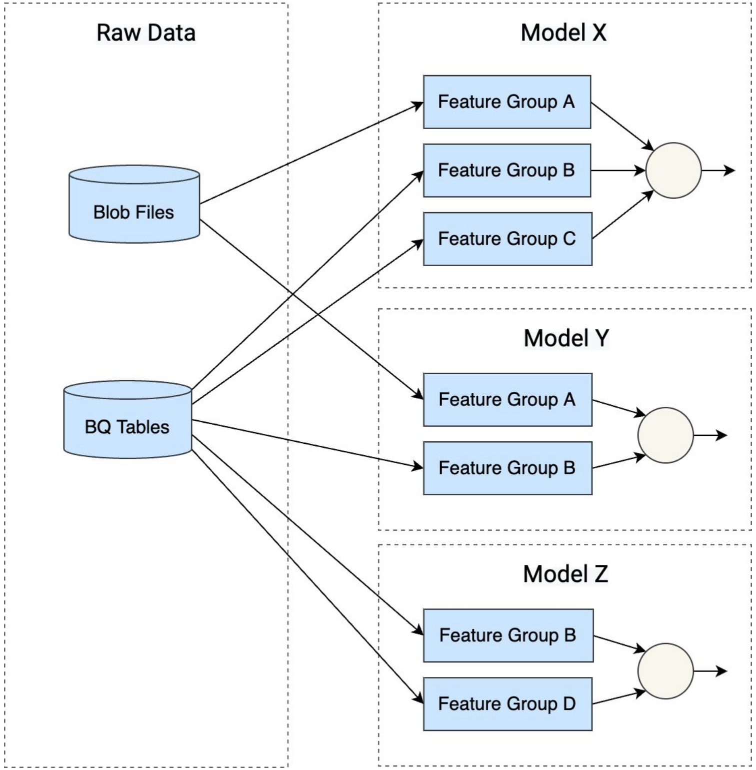
foodora
efood
Glovo
foodpanda
InstaShop
Yemeksepeti
PedidosYa
foody
배달의민족
talabat
HUNGER STATION



Pandora, the internal name for one of the Delivery Hero platforms, powers Foodpanda, Foodora, and a few other brands. Data Scientists from Pandora work on different problems (e.g. recommendation, ranking).



Context



Initially data scientists used to create features for ML models inside model repositories.

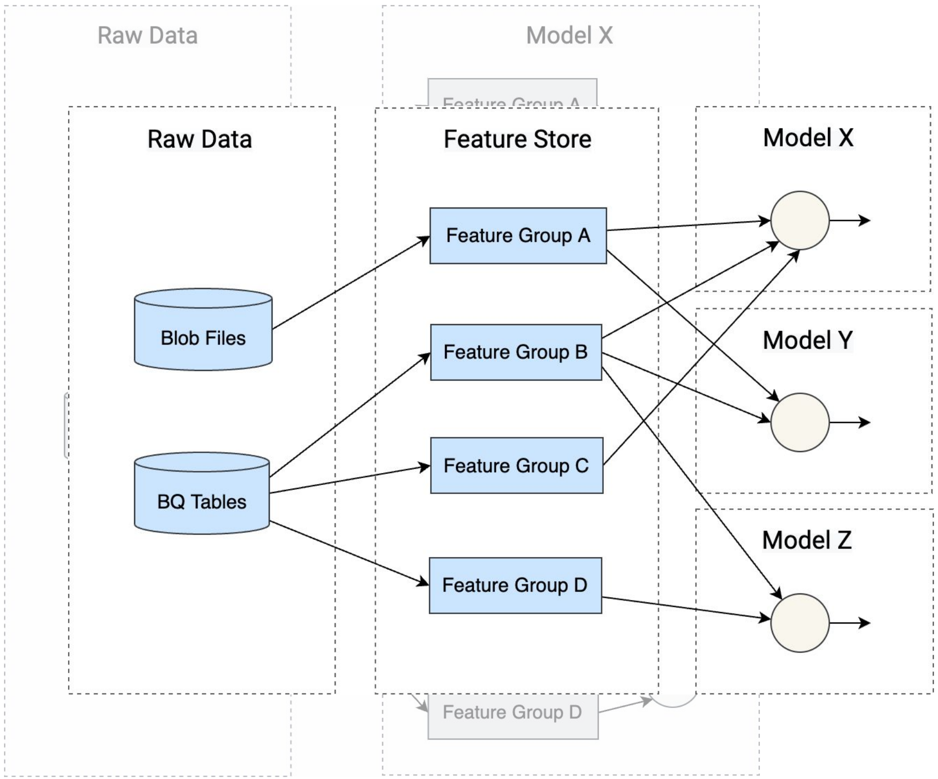
❌Features not easily reused across projects

❌Similar features created over and over again in different repositories

❌No standardization for feature building and data governance processes

That approach became inefficient as the number of teams and projects scaled up

Context



The Feature Store acts as a central component that enables data scientists and engineers to create and serve features

✅Increased feature re-use across projects

✅Standardized feature building and data governance processes

✅Reduced effort for creating new ML models

✅Improved feature consistency and quality

Workflow

Data scientists are primary users of the feature store and they have autonomy to create features, test them and push their changes to production. Empowered and productive data scientists!

Feature Repo

Data Scientists contribute to a feature repository which is a git repo shared with all teams.

Dev Tooling

Data Scientists can run and test new transformations using their development environments.

Code Review

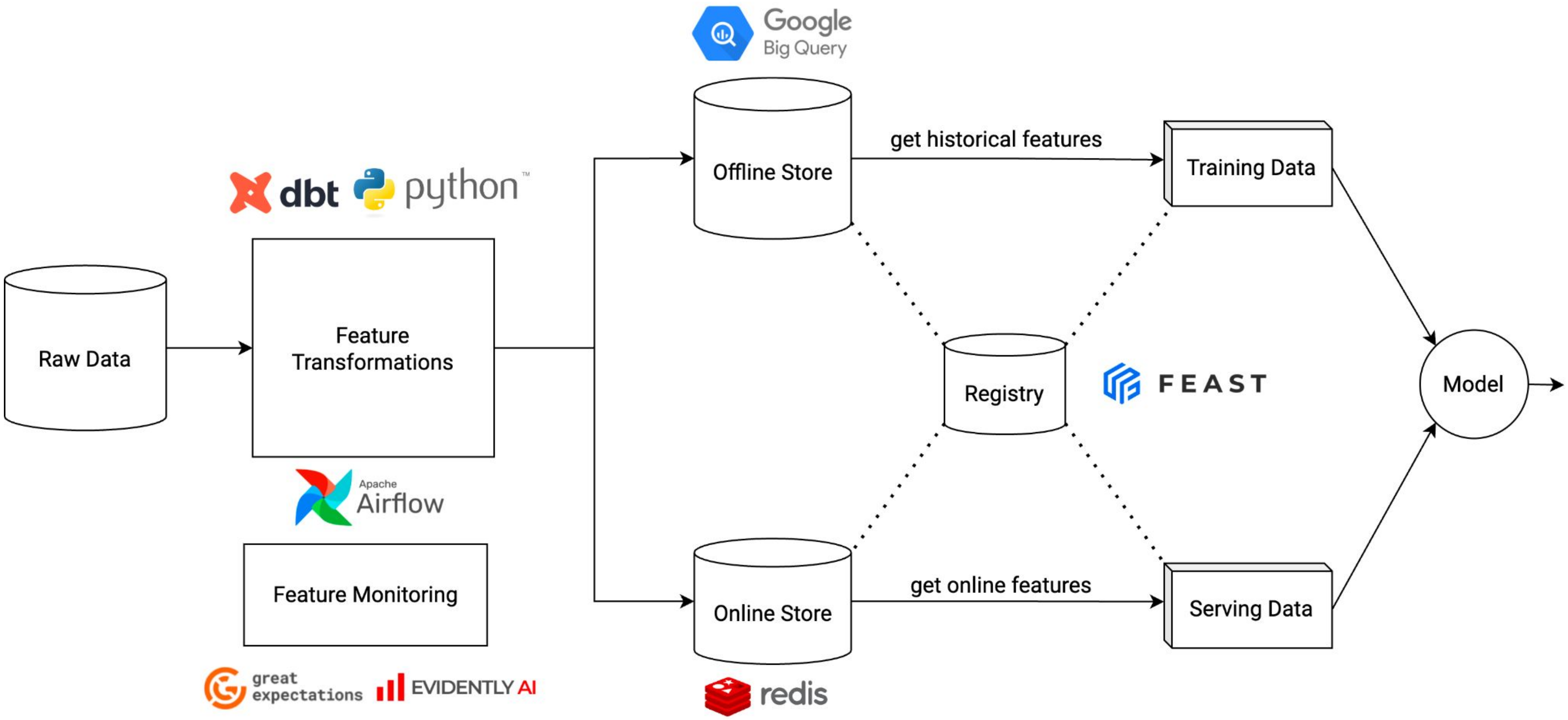
Data Scientists open pull requests to the feature repository and request reviews from their peers to guarantee code quality and consistency.

Production

When PRs are merged, the CI pipelines update remote configuration files and Airflow DAGs. Data Scientists don’t need to interact with Airflow.

High-Level Architecture

Feature Store consists of different components to generate, serve and monitor features



Feature Transformations - Python

- Data scientists can write feature transformations using Python language
- Python is ideal for more complex transformations that can't be managed in SQL
- More complex data structures such as embeddings can also be managed by using this kind of transformation
- Relies on Kubernetes for running Python transformations

Feature Transformations - Python

The folder structure for Python features is quite similar. They're also grouped into entities (e.g. vendors, customers, products).

```
features
  dbt
  python
    customers
    products
    sandbox
      sql
      __init__.py
      config.py
      sandbox_customer_age_features.py
      sandbox_geohash_vendors_features.py
    vendors
  __init__.py
```

```
sandbox_customer_age_features.py
features > python > sandbox > sandbox_customer_age_features.py > ...


def run(context: TransformationContext) -> pd.DataFrame:
    execution_ts = context.vars["exec_ts"]

    df = generate_initial_df(execution_ts)
    df["feature_timestamp_utc"] = execution_ts

    # apply transformations here and return df

    return df

```

example for illustration purposes only

For Python transformations, data scientists just need to write a python module with a function called run that receives context as argument and returns the transformed dataframe.

Feature Transformations - SQL

This component enables data scientists to write feature transformations using SQL

Heavily based on the open-source dbt which is a battled-tested software package from modern data stack

Relies on the BigQuery processing power to bring scalability to the feature generation

dbt is heavily used in many teams at Delivery Hero and it offers several benefits for our needs.

- Its SQL-based approach aligns with our existing data infrastructure and allows our data scientists to leverage their SQL skills to create feature transformations
- Its seamless integration with BigQuery ensures scalability, enabling us to handle large datasets effectively.
- Its proven track record provides us with confidence and its reliability and effectiveness.
- Its compatibility with various platforms gives us the flexibility to work with diverse data sources and adapt to future changes

Feature Transformations - Example

Features are grouped by entities (e.g. vendors, products, customers)

Data scientists that contribute to the same business context can reuse or get insights from existing code written by other data scientists

🔔 Each feature group is an independently managed dbt project. We can also create new projects using a different logical grouping (e.g. by project, team, domain)

- features
  - dbt
    - analytics
    - customers
    - homescreen
    - products
    - rlp
    - sandbox
      - analyses
      - macros
      - models / sandbox
        - orders_past_60_days.sql
        - orders_with_logistics_metrics_past_60_days.sql
        - orders_with_metrics.sql
        - schema.yml
        - vendor_popularity_features.sql
        - vendors_restaurant.sql

vendor_popularity_features.sql ×

features > dbt > sandbox > models > sandbox > vendor_popularity_features.sql

You, 2 minutes ago | 1 author (You)

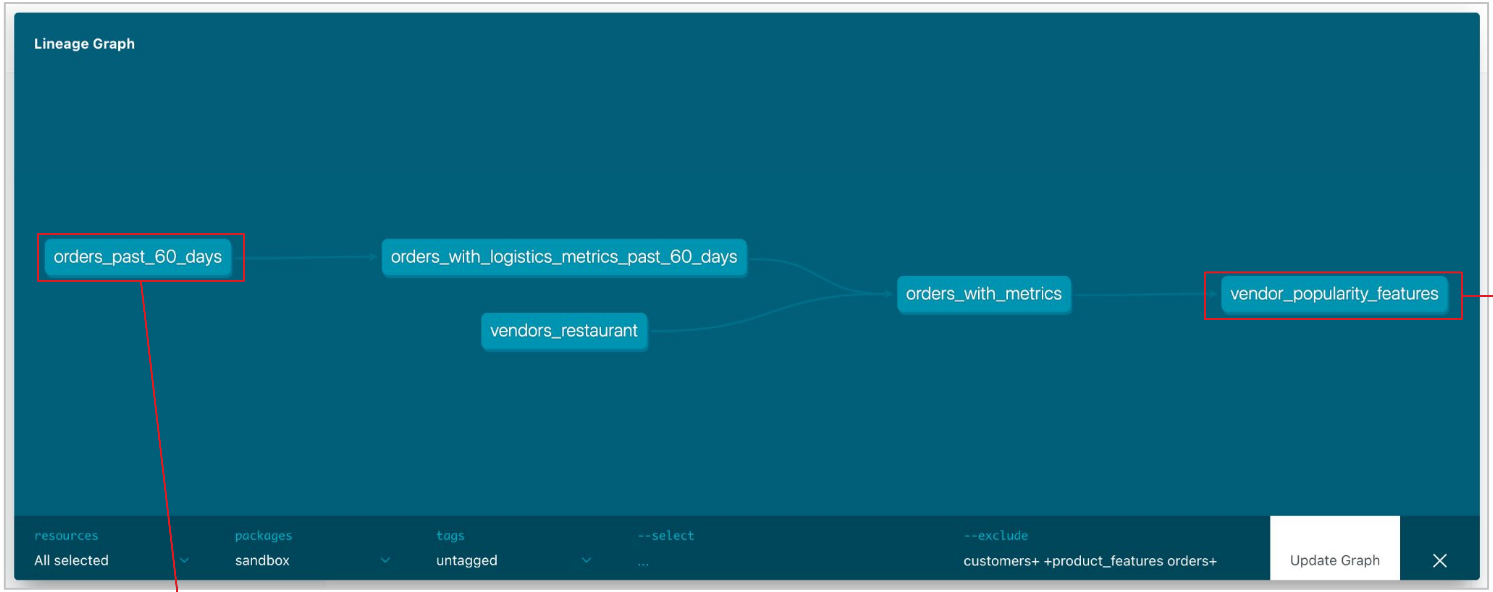
```sql
{{
  config(
    materialized = "incremental",
    incremental_strategy = "insert_overwrite",
    cluster_by = "country",
    partition_by = {
      "field": "feature_timestamp_utc",
      "data_type": "timestamp",
      "granularity": "day",
    },
  )
}}

SELECT
    {{ utils.exec_ts() }} AS feature_timestamp_utc,
    vendor_order_metrics.country,
    vendor_code,
    local_hour_from_day_type,
    COUNTIF(is_1d) AS popularity_1d,
    COUNTIF(is_3d) AS popularity_3d,
    COUNTIF(is_7d) AS popularity_7d,
    COUNTIF(is_15d) AS popularity_15d,
    COUNTIF(is_30d) AS popularity_30d,
    COUNTIF(is_60d) AS popularity_60d
FROM {{ ref('orders_with_metrics') }} AS vendor_order_metrics
GROUP BY
    vendor_order_metrics.country,
    vendor_code,
    local_hour_from_day_type
```

example for illustration purposes only

Feature Transformations - Example

DBT offers pre-built functionalities for executing SQL scripts, testing and documenting the code, and managing dependency graphs



Feature transformations at the end of the dependency graph usually generate features that are actually used by ML models

Base/Intermediate transformations that handle raw data can be reused to feed different feature transformations.
This pipeline decomposition reduces total computational cost.

Common Data Issues

⚠️ **Data producers may introduce breaking changes** to data sources over time causing some issues on ML model pipelines. E.g. deprecation of certain tables, dropped columns, out-of-domain values.

⚠️ **Data distribution shifts**. E.g. changes in business processes, unpredictable external events. Unforeseen events, such as sudden spikes or drops in data volume, can disrupt data distributions and subsequently affect the feature distribution.

⚠️ **Feature transformation bugs**. E.g: feature codes may have issues, and debugging those issues can be non-trivial. These bugs can lead to incorrect calculations, improper data manipulation or skewed representations of underlying patterns.

Feature Monitoring - Quality Checks

Data quality checks component is built on top of Great Expectations open-source package and allows data scientists to configure expectations for their data and validate the feature data produced by the Feature Store

```
monitoring
  dbt
    customers
    homescreen
    products
    rlp
    sandbox
      __init__.py
      customer_features.py
      customer_orders_features.py
      vendor_popularity_features.py
    vendors
```

Users just need to create a monitoring module and write expectations for their data using the GE

```python
def fetch_quality_checks(validator: DatasetValidator):
    validator.expect_column_values_to_not_be_null("country")
    validator.expect_column_values_to_not_be_null("vendor_code")
    validator.expect_column_values_to_not_be_null("feature_timestamp_utc")

    validator.expect_compound_columns_to_be_unique(["country", "vendor_code"])
    validator.expect_column_values_to_be_between("popularity_1d", 0, 120)
    # validator.expect_column_distinct_values_to_be_in_set("segment", ["A", "B", "C"]
```

Users have access to validation reports generated by data quality checks

great expectations

Home / Validations / feature_store_dbt_sandbox.vendor_popularity_features / feature_store_dbt_sandbox.ve

Expectation Validation Result

Evaluates whether a batch of data matches expectations.

Actions

Validation Filter:

Show All | Failed Only

Table of Contents

- Overview
- Table-Level Expectations
- country
- feature_timestamp_utc
- popularity_1d
- vendor_code

Overview

Expectation Suite: feature_store_dbt_sandbox.vendor_popularity_features

Data asset: feature_store_dbt_sandbox.vendor_popularity_features

Status: ✕ Failed

Statistics

Evaluated Expectations	5
Successful Expectations	4
Unsuccessful Expectations	1
Success Percent	80%

Show more info...

Table-Level Expectations

Status	Expectation
	Values for given compound columns must be unique together: country , 259302 unexpected values found. ≈98.67% of 262791 total rows.

Feature Monitoring - Drift Detection

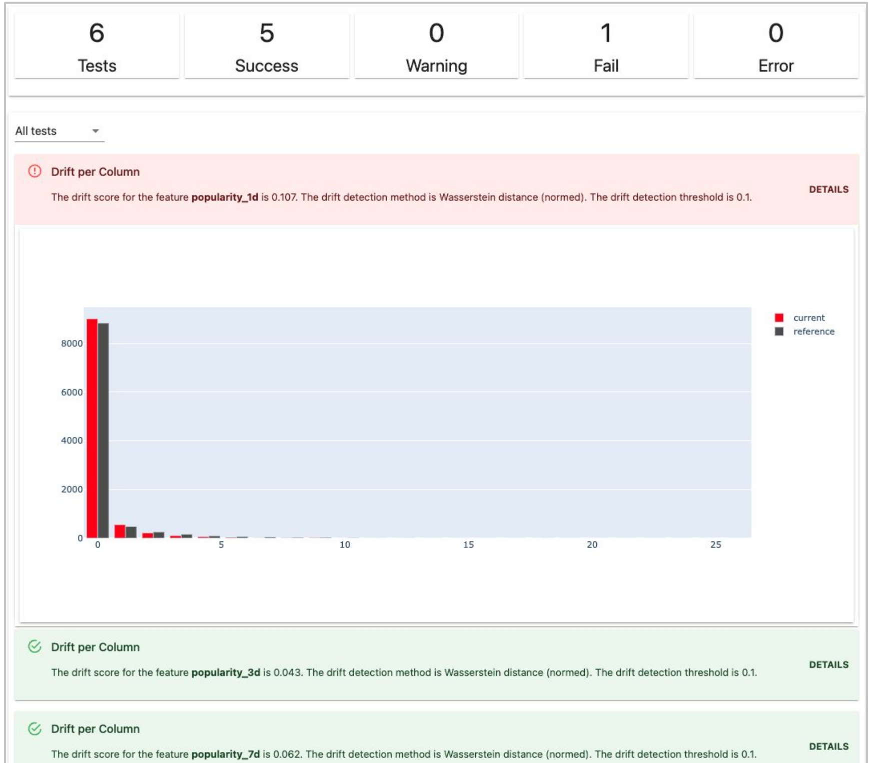
Data drift detection component is built on top of EvidentlyAI package and allows data scientists to detect sudden changes in feature data distribution

- monitoring
  - dbt
    - customers
    - homescreen
    - products
    - rlp
    - sandbox
      - __init__.py
      - customer_features.py
      - customer_orders_features.py
      - vendor_popularity_features.py

Users just need to create a monitoring module with drift config (algorithms, parameters)

```python
def fetch_drift_config():
    return DriftConfig(
        delta_reference_ts=timedelta(-1),
        sample_size=10000,
        columns=[
            {"name": "popularity_1d", "metric": "wasserstein", "threshold": 0.1},
            {"name": "popularity_3d", "metric": "wasserstein", "threshold": 0.1},
            {"name": "popularity_7d", "metric": "wasserstein", "threshold": 0.1},
            {"name": "popularity_15d", "metric": "wasserstein", "threshold": 0.1},
            {"name": "popularity_30d", "metric": "wasserstein", "threshold": 0.1},
            {"name": "popularity_60d", "metric": "wasserstein", "threshold": 0.1},
        ],
    )
```

Data drift reports show drift results for each feature configured by user



Feature Monitoring - Detected Issues

✅ **Unexpected duplication of keys in data source tables**. Unforeseen duplications were detected, likely due to systematic inconsistencies in data collection or aggregation processes.

✅ **Missing data due to ELT script changes**. Essential data was found to be missing as a consequence of changes introduced by data producers.

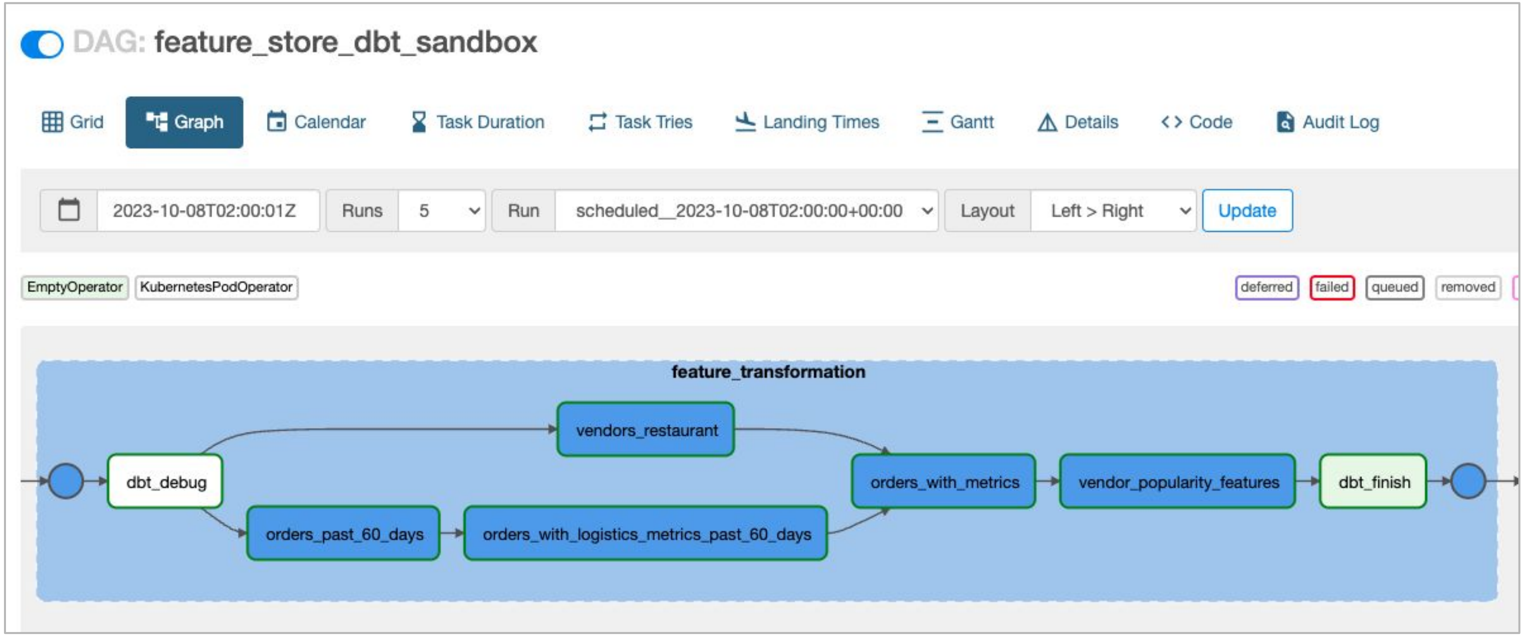
✅ **Country-specific event-triggered distribution shifts**. Significant shifts in feature data distribution were observed in specific countries owing to unpredictable events, causing a change in the original characteristics of the data.

✅ **Incorrect feature data**. Erroneous feature values may surface due to bugs in the feature transformation codes.

✅ **Out-of-domain values in feature data**. Lack of robust validation in data production permitted the introduction of out-of-domain values, resulting in a high-level of noise in data.

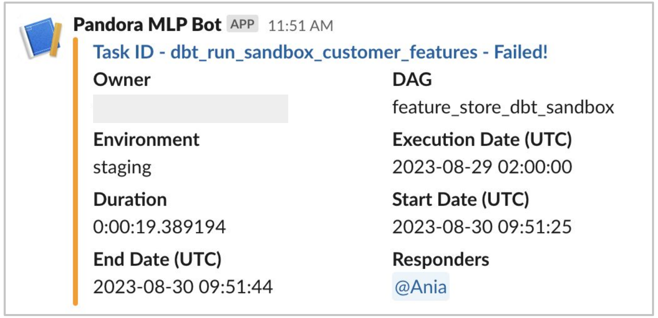
Feature Pipelines - Orchestration

Feature Pipelines are orchestrated by the Airflow. Dynamically generated DAGS without user intervention. Users can also backfill tables for a specified historical period using a configuration file.

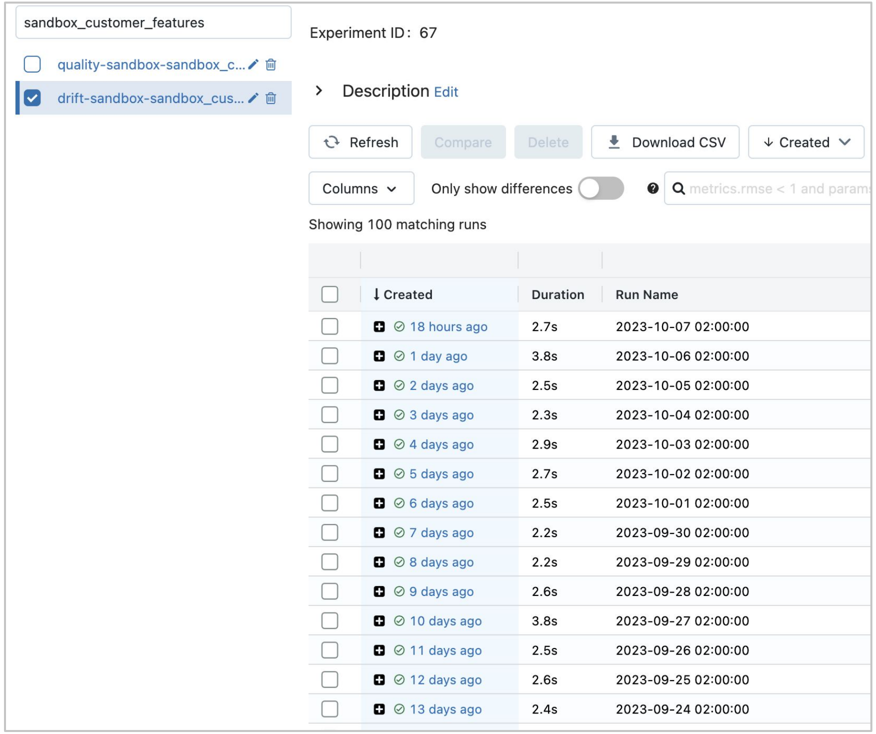


Feature Pipelines - Notifications

Users receive Slack notifications whenever a feature transformation is failed or a data quality issue is found



All reports are tracked into Metadata Tracking server Data Quality and Drift reports can be verified by users



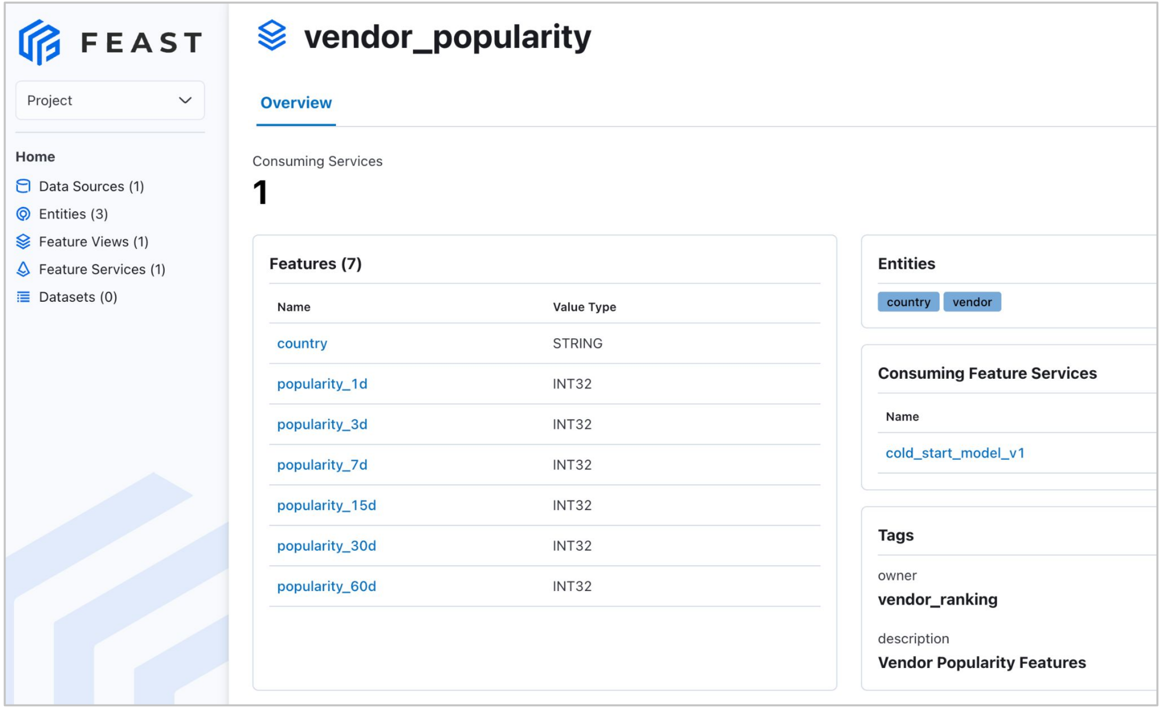
Feature Registry

Once the feature data is produced into BigQuery, data scientists can use Feast to register features into registry

```
vendor_popularity_ds = BigQuerySource(
    table=f"{dataset_id}.vendor_popularity_features",
    timestamp_field="feature_timestamp_utc",
)

vendor_popularity_fv = FeatureView(
    name="vendor_popularity",
    description="Vendor Popularity Features",
    owner="vendor_ranking",
    source=vendor_popularity_ds,
    entities=[country, vendor],
    schema=[
        Field(name="country", dtype=String),
        Field(name="vendor_code", dtype=String),
        Field(name="popularity_1d", dtype=Int32),
        Field(name="popularity_3d", dtype=Int32),
        Field(name="popularity_7d", dtype=Int32),
        Field(name="popularity_15d", dtype=Int32),
        Field(name="popularity_30d", dtype=Int32),
        Field(name="popularity_60d", dtype=Int32)
    ]
)
```

```
cold_start_model_v1 = FeatureService(
    name="cold_start_model_v1",
    description="Sandbox Cold Start Model",
    owner="vendor_ranking",
    features=[vendor_popularity_fv]
)
```



Feature views can be reused across different feature services. They can also be versioned (e.g. _v2, _v3).

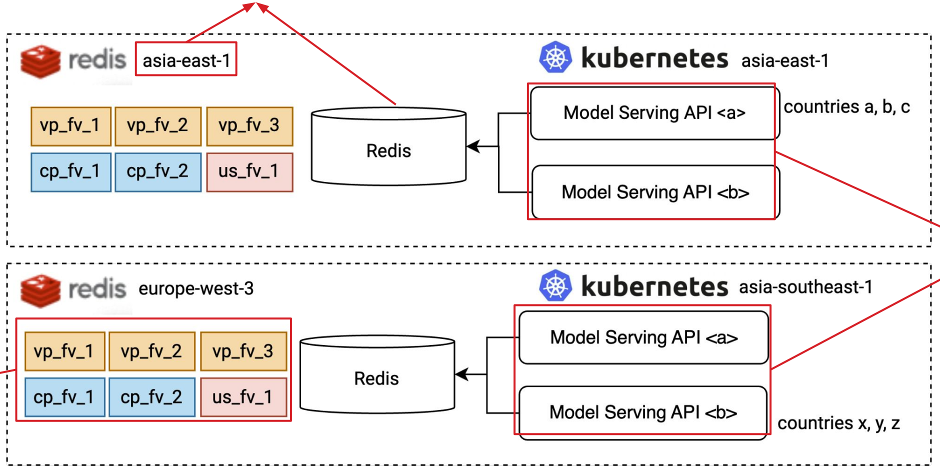
💡 Feast also supports other types of data sources (e.g. streaming)

Online Feature Store

Online stores serve features at low latency and Redis is a super fast in-memory data store.

Redis instances deployed in different regions to reduce network latency.

A subset of feature views are materialized in each Redis instance according to user configuration



Services deployed in different regions according to countries to reduce network latency

💡 Feast also supports other online stores (e.g. BigTable, DynamoDB, PostgreSQL, Rockset, Hazelcast)

Takeaways

- A centralized feature store promotes increased feature reusability across projects.
- By providing a shared feature repository, data scientists can leverage existing features, eliminating redundant creation of features and duplication of efforts.
- The feature store has also established standardized feature generation and quality processes.
- With a unified framework, data scientists can follow consistent practices for creating and validating features, resulting in more robust and reliable machine learning pipelines.
- All feature pipelines are automated and this also reduces the engineering efforts for data scientists and data engineers working on application teams.

Some Articles

- Leveraging the Feature Store for Fast-Tracking ML Model Development
- Personalization Journey @ Delivery Hero
- Personalization @ Delivery Hero: Understanding Customers
- Personalisation @ Delivery Hero: Ranking restaurants for new users
- Don't Worry, We Got You: Personalized Model
- Delivery Hero’s Double Feature at ACM RecSys 2023

Our tech blog 👇

https://tech.deliveryhero.com/

Thank you!

Do you have any questions?

breno.costa@deliveryhero.com
www.deliveryhero.com