Feathr Feature Store

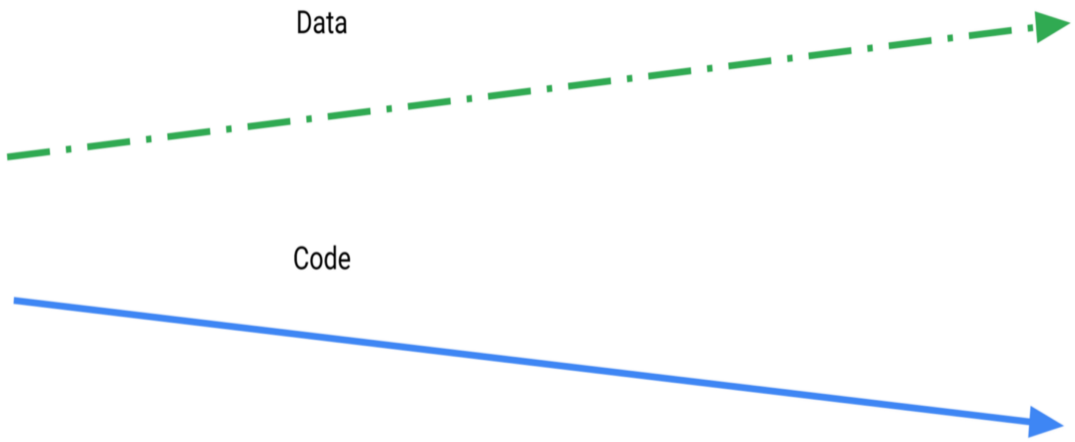
Feathr team

Agenda

- Feature Store Motivation
- Introducing Feathr and Feathr Highlights
- Architecture
- Demo
- Roadmap
- Q&A

ML Production Challenges

- ML = Code + Data
- Data is from real world
  - Never stop changing
  - You can’t control how it changes
- Impact of data and code lives in two separate planes



Feature Store: Solving Data Problems in AI

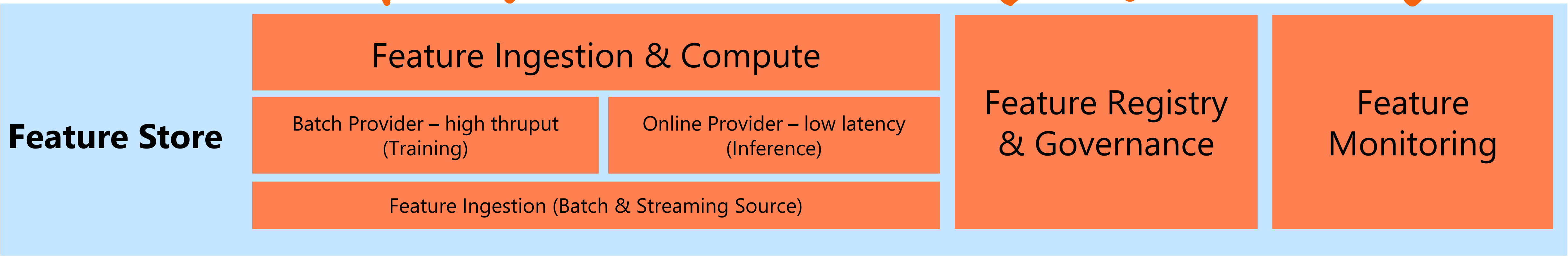
https://github.com/linkedin/feathr

Solving Challenges to Productizing Data for AI/ML

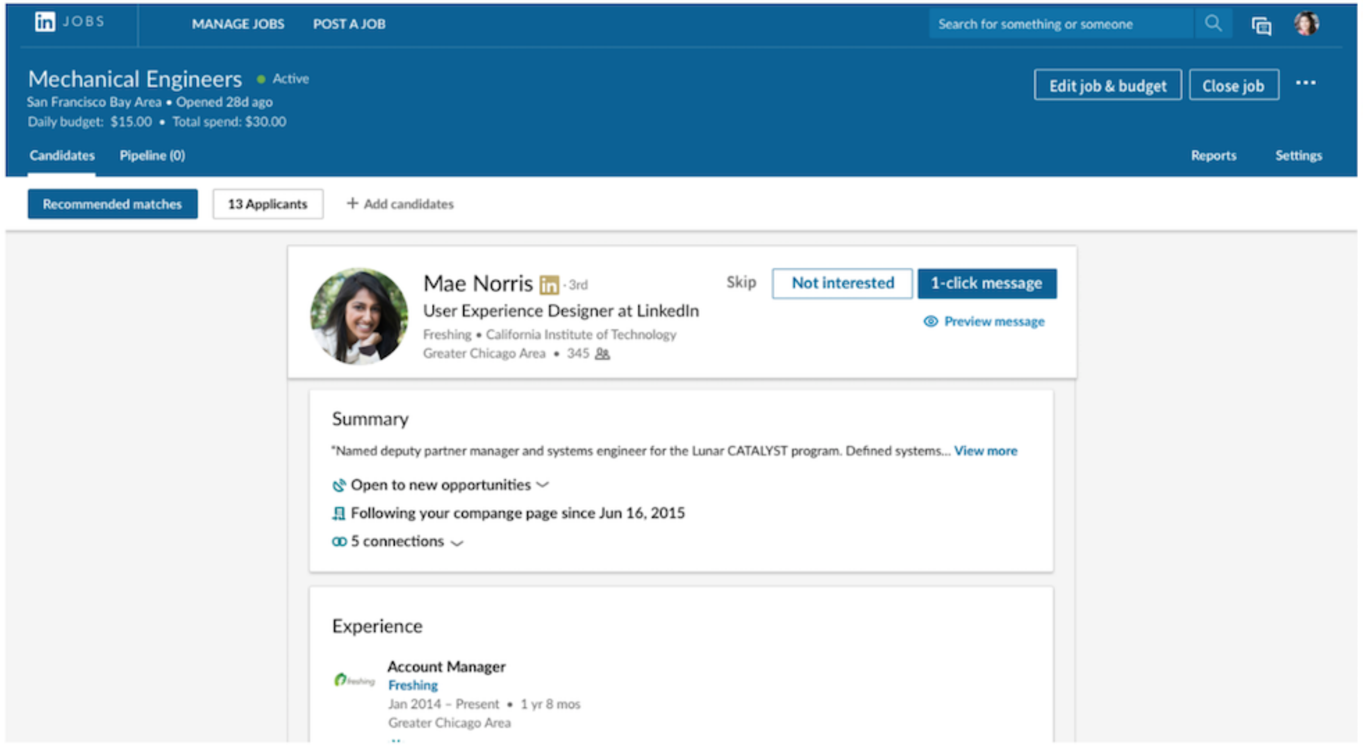
- Features aren't reused, and hard to be tracked & meet compliance standard
- Feature definitions are inconsistent across teams.
- Getting features into production is hard.
- Feature are inconsistent between training and serving, online and offline and cause model performance degrade
- Quality of features and data are changing overtime and need human in the loop

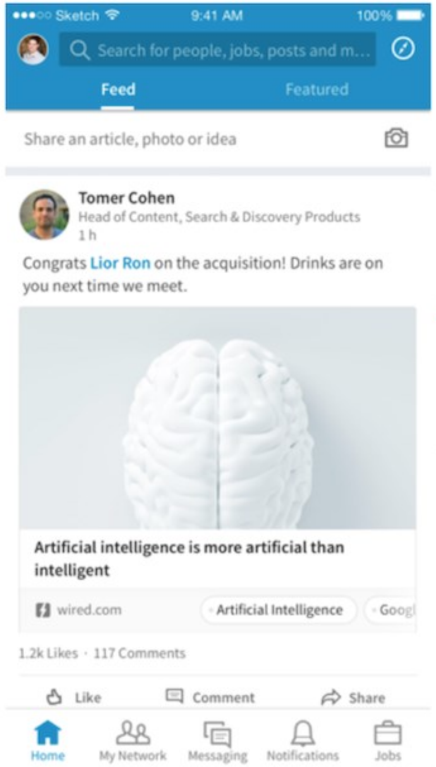
Feature Store solves the problem

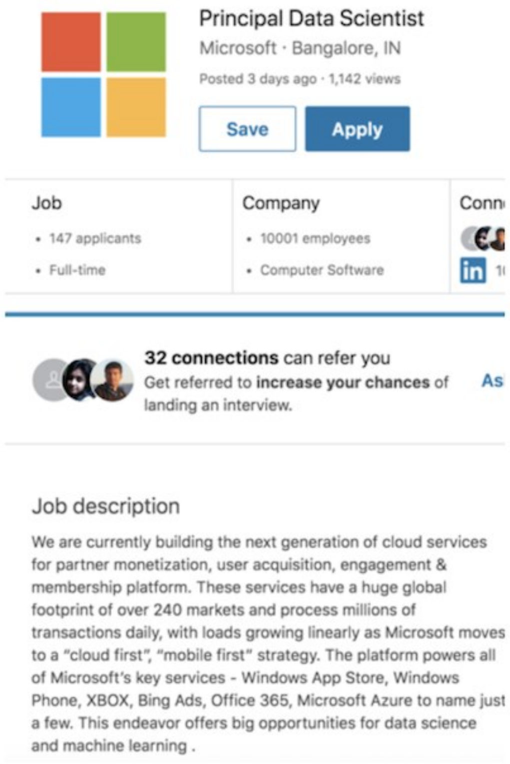
- Features aren't reused, and hard to be tracked & meet compliance standard
- Feature definitions are inconsistent across teams.
- Getting features into production is hard.
- Feature are inconsistent between training and serving, online and offline and cause model performance degrade
- Quality of features and data are changing overtime and need human in the loop



Introducing Feathr, a battle tested feature store built by LinkedIn







[Learning Hiring Preferences: The AI Behind LinkedIn Jobs]()
[Personalized Recommendations in LinkedIn Learning]()
[Helping members connect to opportunity through AI]()
[Near real-time features for near real-time personalization]()

Feathr – brief history within LinkedIn

- Widely used at LinkedIn:
  - Serving features for most ML applications at LinkedIn
  - Hundreds of training workflows
  - LinkedIn has been using Feathr in production for over 6 years and have a dedicated team improving it
- Usability & Flexibility
  - Large custom feature pipelines reduced from 1000s lines of code to 10s of lines of feature configuration
  - Feature compute & join performance increase compared to application-specific custom feature workflows
- Enabled feature sharing across teams, leading to biz metrics gains
- Open sourced & announced in Apr 2022

Feathr Highlights

Rich UDF Support

- Highly customizable UDFs with native PySpark and Spark SQL to lower learning curve for data scientists

```
def udf_count_distinct(df: DataFrame):
    from pyspark.sql.functions import when, countDistinct, regexp_replace
    df.withColumn('address',
     when(df.address.endswith('Rd'),regexp_replace(df.address,'Rd','Road')) \
    .when(df.address.endswith('St'),regexp_replace(df.address,'St','Street')) \
    .when(df.address.endswith('Ave'),regexp_replace(df.address,'Ave','Avenue')) \
    .otherwise(df.address)) \
    .show(truncate=False)
    df = df.withColumn("trip_distance", df.trip_distance.cast('double'))
    df.select(countDistinct("DOLocationID", "PULocationID"))
    return df

request_anchor = FeatureAnchor(name="request_features",
                               source=batch_source,
                               features=features,
                               preprocessing=udf_count_distinct)
```

Rich Support for Point-in-time Joins and Aggregations

- High performant built-in operators designed for feature store
  - Including point in time joins, time-aware sliding window aggregation, look up features, all with point-in-time correctness
- Feature ideation to production reduced from weeks to hours with built-in operators

```
agg_features = [Feature(name="f_location_avg_fare",
                        key=location_id,
                        feature_type=FLOAT,
                        transform=WindowAggTransformation(agg_expr="cast_float(fare_amount)",
                                                          agg_func="AVG",
                                                          window="90d")),
                Feature(name="f_location_max_fare",
                        key=location_id,
                        feature_type=FLOAT,
                        transform=WindowAggTransformation(agg_expr="cast_float(fare_amount)",
                                                          agg_func="MAX",
                                                          window="90d"))
                ]
feature_query = FeatureQuery(
    feature_list=["f_location_avg_fare", "f_trip_time_rounded", "f_is_long_trip_distance"], key=location_id)
settings = ObservationSettings(
    observation_path="abfss://feathrazuretest3fs@feathrazuretest3storage.dfs.core.windows.net/demo_data/green_tripdata_2020-04.csv",
    event_timestamp_column="lpep_dropoff_datetime",
    timestamp_format="yyyy-MM-dd HH:mm:ss")
client.get_offline_features(observation_settings=settings,
                            feature_query=feature_query,
                            output_path="abfss://feathrazuretest3fs@feathrazuretest3storage.dfs.core.windows.net/demo_data/output.avro")
```

Derived Features - Encourage Feature Reuse

- Allow defining features on other features to encourage feature reuse:

```
f_trip_distance = Feature(name="f_trip_distance",
                          feature_type=FLOAT, transform="trip_distance")
f_trip_time_duration = Feature(name="f_trip_time_duration",
                               feature_type=INT32,
                               transform="(to_unix_timestamp(lpep_dropoff_datetime) - to_unix_timestamp(lpep_pickup_datetime))/60")

f_trip_time_distance = DerivedFeature(name="f_trip_time_distance",
                                      feature_type=FLOAT,
                                      input_features=[
                                          f_trip_distance, f_trip_time_duration],
                                      transform="f_trip_distance * f_trip_time_duration")
```

Read multiple sources at once, including Streaming Source

- Allow ingesting data from multiple sources at once
- Ingesting data from Streaming sources to make sure features are fresh

```
file_source = HdfsSource(name="nycTaxiBatchSource",
                         path="wasbs://public@azurefeathrstorage.blob.core.windows.net/sample_data/green_tripdata_2020-04.csv",
                         event_timestamp_column="lpep_dropoff_datetime",
                         timestamp_format="yyyy-MM-dd HH:mm:ss")

snowflake_source = HdfsSource(name="snowflakeSampleBatchSource",
                         path="jdbc:snowflake://dqllago-ol19457.snowflakecomputing.com/?user=feathrintegration&sfWarehouse=COMPUTE_WH&dbtable=CALL_CENTER&
                         sfDatabase=SNOWFLAKE_SAMPLE_DATA&sfSchema=TPCDS_SF10TCL",
                         )

stream_source = KafKaSource(name="kafkaStreamingSource",
                         kafkaConfig=KafkaConfig(brokers=["feathrazureci.servicebus.windows.net:9093"],
                                                 topics=["feathrcieventhub"],
                                                 schema=schema)
                         )
```

Type system designed for ML

- Rich type system including support for embeddings for advanced ML/DL scenarios
  - Built-in support for embeddings (such as user activity, content, etc.) and those embeddings can be reused across an organization
  - Reduce time to deliver complex embedding features and boost model performance
  - Support common ML types, such as categorical/categorical set, etc.

```
# Another example to compute embedding similarity
user_embedding = Feature(name="user_embedding", feature_type=FLOAT_VECTOR, key=user_key)
item_embedding = Feature(name="item_embedding", feature_type=FLOAT_VECTOR, key=item_key)


user_item_similarity = DerivedFeature(name="user_item_similarity",
                                      feature_type=FLOAT,
                                      key=[user_key, item_key],
                                      input_features=[user_embedding, item_embedding],
                                      transform="cosine_similarity(user_embedding, item_embedding)")
```

Type system designed for ML

- Rich type system including support for embeddings for advanced ML/DL scenarios
  - Built-in support for embeddings (such as user activity, content, etc.) and those embeddings can be reused across an organization
  - Reduce time to deliver complex embedding features and boost model performance
  - Support common ML types, such as categorical/categorical set, etc.

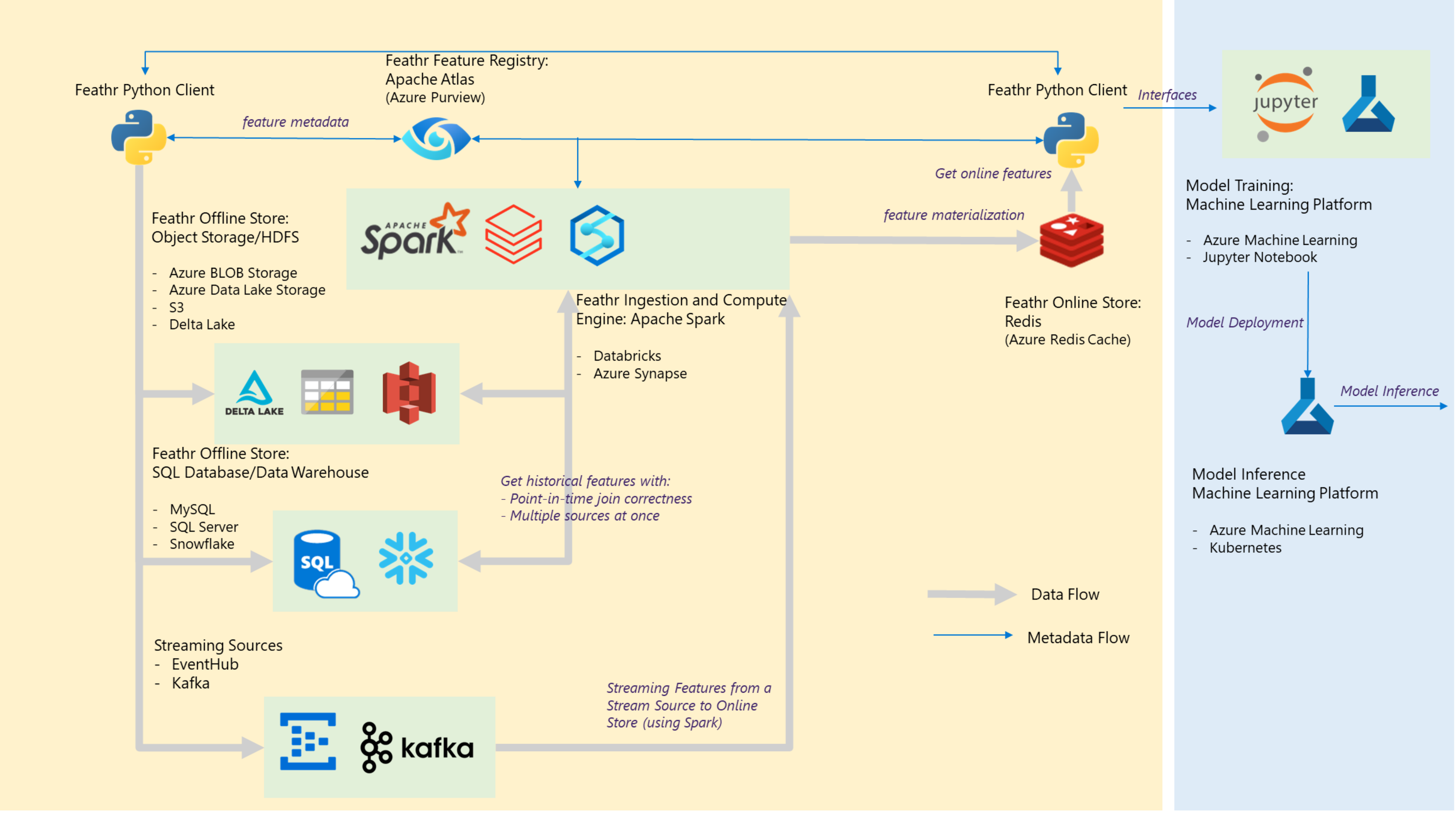
```
# Another example to compute embedding similarity
user_embedding = Feature(name="user_embedding", feature_type=FLOAT_VECTOR, key=user_key)
item_embedding = Feature(name="item_embedding", feature_type=FLOAT_VECTOR, key=item_key)


user_item_similarity = DerivedFeature(name="user_item_similarity",
                                      feature_type=FLOAT,
                                      key=[user_key, item_key],
                                      input_features=[user_embedding, item_embedding],
                                      transform="cosine_similarity(user_embedding, item_embedding)")
```

Scalability and Compliance

- Scalability
  - Capable of processing tens of billions of rows and PB scale data
  - Native optimizations like bloom filters, join plan optimizer, salted join
  - Cloud-friendly scalable architecture
- Compliance
  - Went thru CCPA/GDPR reviews and users can build CCPA/GDPR compliant pipelines using Feathr

Feathr on Azure



Feathr Python Client
Feathr Feature Registry:
Apache Atlas
(Azure Purview)
feature metadata
Feathr Python Client
Interfaces
jupyter
Get online features
feature materialization
Model Training:
Machine Learning Platform
- Azure Machine Learning
- Jupyter Notebook
Feathr Offline Store:
Object Storage/HDFS
- Azure BLOB Storage
- Azure Data Lake Storage
- S3
- Delta Lake
APACHE Spark
Feathr Ingestion and Compute Engine: Apache Spark
- Databricks
- Azure Synapse
Feathr Online Store:
Redis
(Azure Redis Cache)
Model Deployment
Model Inference
DELTA LAKE
Feathr Offline Store:
SQL Database/Data Warehouse
- MySQL
- SQL Server
- Snowflake
SQL
Get historical features with:
- Point-in-time join correctness
- Multiple sources at once
Model Inference
Machine Learning Platform
- Azure Machine Learning
- Kubernetes
Data Flow
Metadata Flow
Streaming Sources
- EventHub
- Kafka
kafka
Streaming Features from a Stream Source to Online Store (using Spark)

Feathr Integration with Cloud Services

Feathr component	Cloud Integrations
Offline store – Object Store	Azure Blob Storage, Azure ADLS Gen2, AWS S3
Offline store – SQL	Azure SQL DB, Azure Synapse Dedicated SQL Pools, Azure SQL in VM, Snowflake
Streaming Source	Kafka, EventHub
Online store	Azure Cache for Redis
Feature Registry	Azure Purview
Compute Engine	Azure Synapse Spark Pools, Databricks
Machine Learning Platform	Azure Machine Learning, Jupyter Notebook
File Format	Parquet, ORC, Avro, Delta Lake

https://github.com/linkedin/feathr#cloud-integrations

Demo

Feature Store Abstraction

- “Put a feature in” (Producer)
  - Register a feature based on a pre-computed feature data set
  - Register a feature based on a **raw data set**
    - Sliding time windows
    - Aggregations
    - Transformations
    - Lookups/joins
  - Register a feature based on other feature(s)
- “Get some features out” (Consumer)
  - Join features to training data labels/observations
  - For training, compute historical values of features (**point-in-time correctness**)
  - Efficiently precompute, store, and serve features for **online inference**

Demo

- https://github.com/linkedin/feathr/blob/main/feathr_project/feathrcli/data/feathr_user_workspace/nyc_driver_demo.ipynb

Resources

More Resources

- Project homepage:
  - https://linkedin.github.io/feathr/
- For more details on getting started, please refer to:
  - https://github.com/linkedin/feathr/blob/main/feathr_project/feathrcli/data/feathr_user_workspace/nyc_driver_demo.ipynb
- Source code:
  - https://github.com/linkedin/feathr
- The Feathr team can also be reached in the Feathr Community (Anyone is welcome!)
  - https://feathrai.slack.com/

Roadmap: https://github.com/linkedin/feathr/milestones

- Short term:
  - Online Feature Transformation
  - Feature Monitoring

  - More input/output sources
  - Improved Feature Registry experience

- Mid term & Long term
  - Feature Versioning
  - RBAC support
  - Auto featurization tool integration