

Use Apache Beam to build Machine Learning Feature System at Affirm

Hao Xu



Use Apache Beam To Build Machine Learning Feature System At Affirm

- Hao Xu



01

ABOUT ME

Earnest -> Fast -> Affirm -> JP Morgan & Chase

TABLE OF CONTENTS

01

BACKGROUND

- MLFS
- Stream Platform

03

SOLUTION

- Unified transformation
- OOTB APIs

02

PAIN POINTS

- Slowness
- Learning curves

04

OUTCOME

- Performance
- Dev Velocity

Background

- BNPL
- Machine learning feature store
- Streaming and Batch Compute Platform

The Story of BNPL

Your 3 payments of \$50.00



Total of payments \$150.00 ▾

\$50.00 is due next month



Set up automatic payments (optional)

You'll pay \$50.00 on each due date.

Complete your order

Pick a payment plan

\$233.00/monthly

3 Months

APR	Interest	Total
0.00%	\$0.00	\$500.00

\$120.00/monthly

6 Months

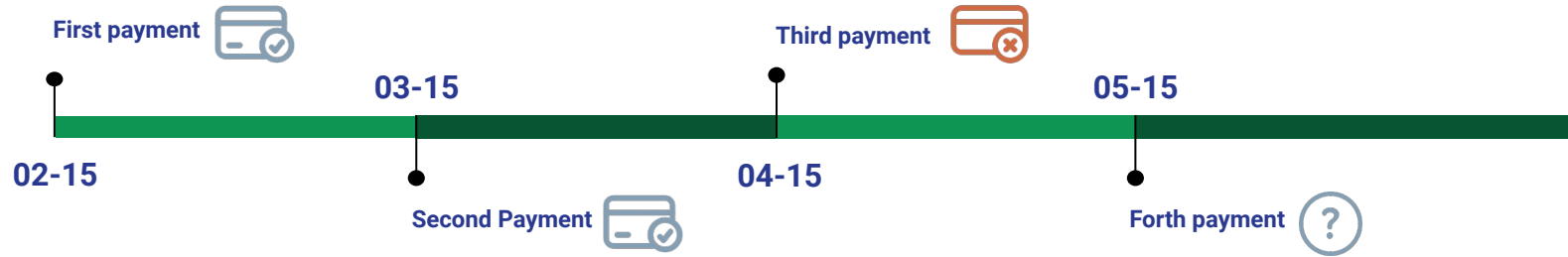
APR	Interest	Total
15.01%	\$22.66	\$522.66

\$62.00/monthly

12 Months

APR	Interest	Total
15.01%	\$42.40	\$542.40

The Story of BNPL

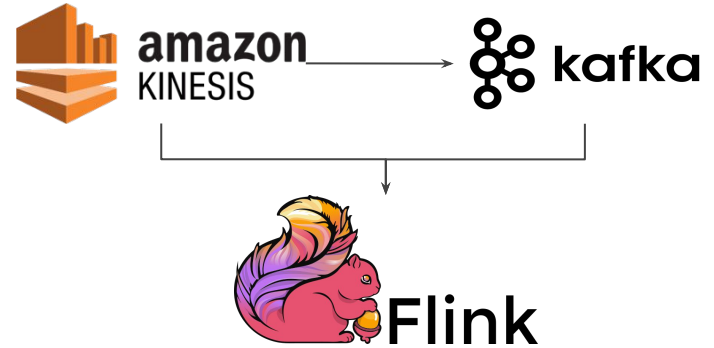
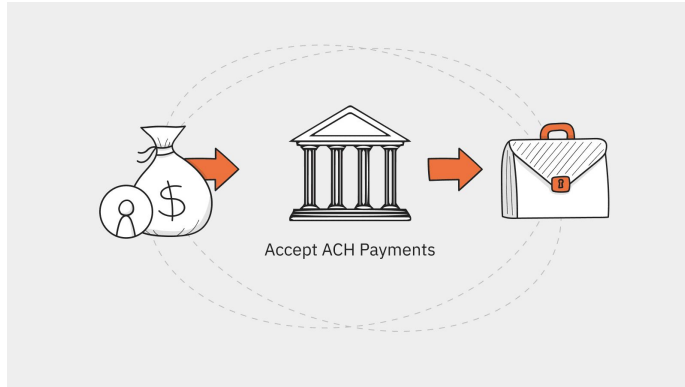


If a user failed the third payment, is it likely that they will also fail the fourth one?

Has the user failed to make a loan payment, and if so, have we identified the issue? Should we approve another loan for them?

Payment flow

The payment data was processed in batches, resulting in a delay of a couple of days. Utilizing stream data can help prevent such delays in the future.



Feature Store



Figure 1. A feature store is the interface between feature engineering and model development.

Pain Points

The background features abstract, organic shapes in shades of blue and dark blue. A thin, light blue line meanders across the top and right sides. In the bottom right corner, there are two dark blue shapes, each containing three parallel white diagonal lines.

Pain Points



Development Velocity

Slow backfilling of stream features.
Excessive code required to define a feature.



Variety

Inability to join two streams from Kinesis together, which is typically required for stateful processing.



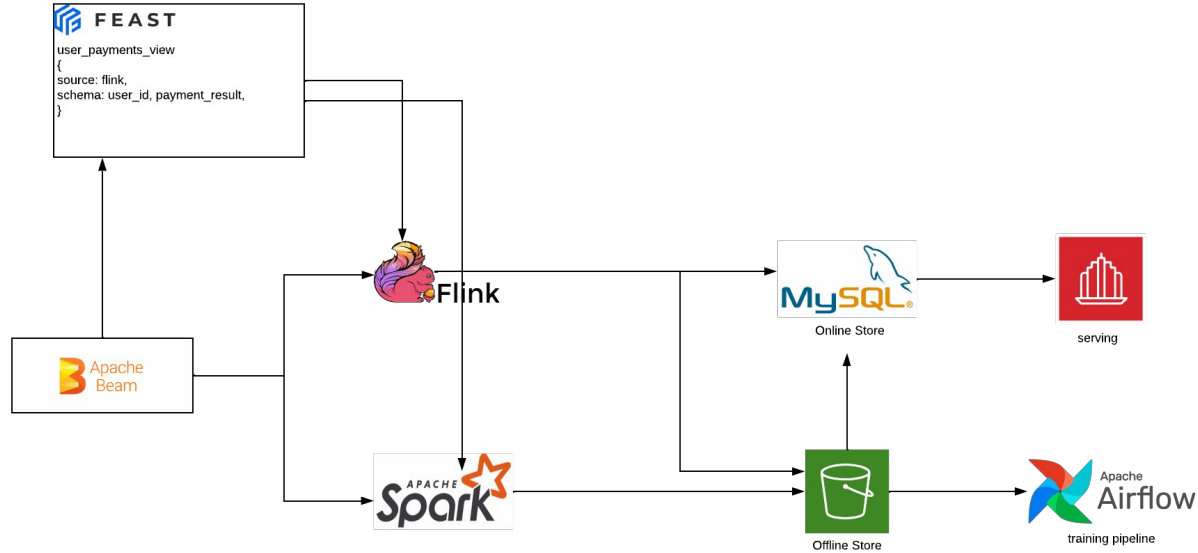
Visibility

Lack of registry to quickly lookup data sources, features and metadata.

The background features abstract, organic shapes in shades of blue and dark blue. A thin, light blue line meanders across the white space. In the top right and bottom right corners, there are larger, solid-colored shapes in a darker blue, each containing three parallel white diagonal lines.

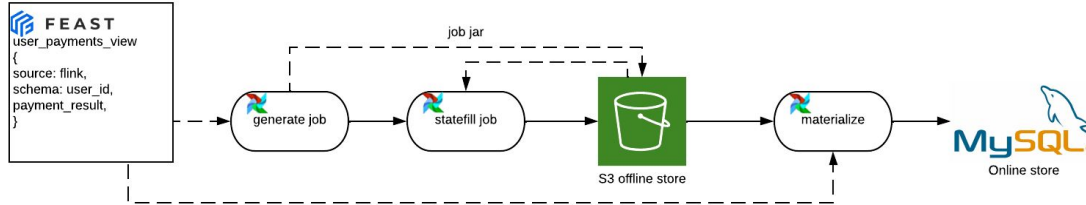
Solution

MLFS Architecture

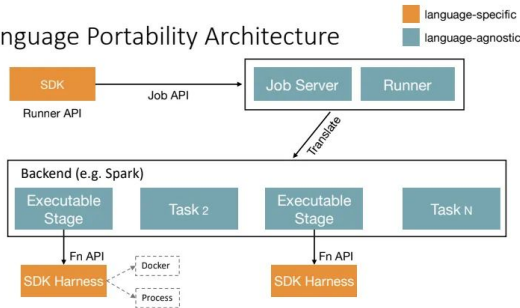


Complex of Backfilling

Backfilling is the process to backfill a feature data to the historical point in time



Language Portability Architecture



```
spec:
  containers:
  - name: spark-kubernetes-executor
    volumeMounts:
    - name: beam-data
      mountPath: /opt/apache/beam/
  initContainers:
  - name: init-beam
    image: apache/beam_python3.7_sdk
    command:
    - cp
    - /opt/apache/beam/boot
    - /init-container/data/boot
  volumeMounts:
  - name: beam-data
    mountPath: /init-container/data
  volumes:
  - name: beam-data
    emptyDir: {}
```

Unified Transformation Interface

```
class UnifiedTransformer(Transformer[beam.PCollection, beam.PCollection]):

    @property
    def window(self) -> beam.WindowInto:
        return self._window

    @property
    def event_transform(self) -> beam.PTransform:
        return self._event_transform

    @property
    def aggregator(self) -> beam.PTransform:
        return self._aggregator

    def run(self, inputs: beam.PCollection) -> beam.PCollection:
        if self.feast_context.runner == Runner.flink:
            if self.window:
                inputs = inputs | self.window
            return (
                inputs
                | self.event_transform.with_output_types(Tuple)
                | self.aggregator.with_output_types(Tuple)
            )
        elif self.feast_context.runner == Runner.spark:
            return (
                inputs
                | self.event_transform.with_output_types(Tuple)
                | self.aggregator.with_output_types(Tuple)
            )
        else:
            raise ValueError("Unsupported runner: {}".format(self.feast_context.runner))
```

Unified Transformation Interface

```
@stream_feature_view(  
    entities=[entity_registry['user_ari']],  
    ttl=timedelta(days=0),  
    schema=[  
        Field(name="user_ari", dtype=String),  
        Field(name="timestamp", dtype=UnixTimestamp),  
        Field(name="latest_payment_fail", dtype=UnixTimestamp),  
        Field(name="latest_payment_fail_ach_nsf", dtype=UnixTimestamp),  
    ],  
    online=True,  
    source=user_payment_fails_stream_source,  
    timestamp_field="timestamp",  
    tags={},  
    mode="flink",  
)  
def user_last_payment_fail(feast_context: FeastContext, inputs: PCollection) -> PCollection:  
    transformer = UnifiedTransformer(  
        feast_context=feast_context,  
        aggregator=LatestFeatureAggregator(feast_context, 'timestamp'),  
        event_transform=extract_payment_fail_data,  
    )  
    return transformer.run(inputs)
```


The background features abstract, organic shapes in shades of blue and dark blue. A thin, light blue line meanders across the top and right sides. In the bottom right corner, there are two dark blue shapes, each containing three parallel white diagonal lines. The word "Outcome" is centered in a bold, blue, sans-serif font.

Outcome

Performance boost

80%

Backfilling time

Backfilling time improved by 80%

60%

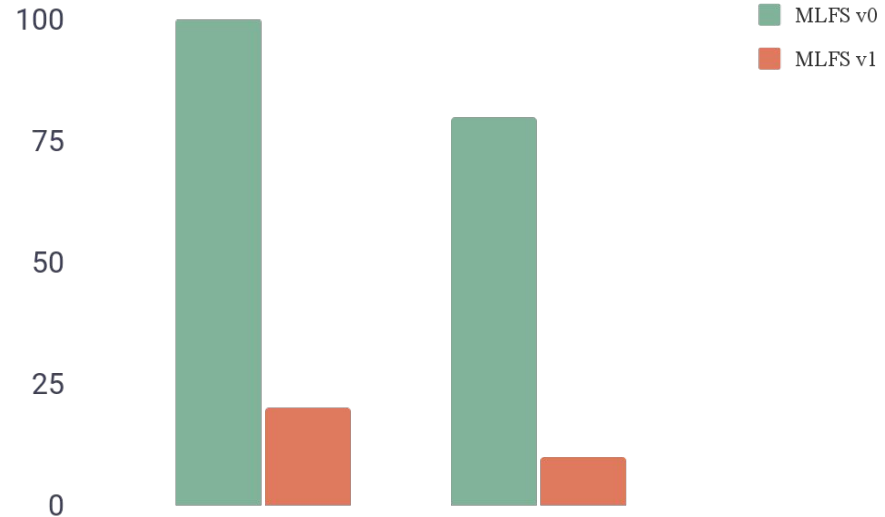
Code lines

Reduced 100+ lines to 20+ lines

40%

Registry

200+ data sources
100+ features



The time spent to backfill features for feature *time_since_user_checkout* and *tiem_since_user_last_payment_failure*

Future improvement

1. OOTB transformation interface
2. Transformation framework
3. Improvement on Beam Spark Runner