DOORDASH

Fabricator : A Declarative Feature Platform

Kunal Shah

Agenda

Feature Engineering at Doordash

Reimagining an ideal Feature Platform

Fabricator : Overview

Architecture deep dives

Results and Learnings

Agenda

Feature Engineering at Doordash

Reimagining an ideal Feature Platform

Fabricator : Overview

Architecture deep dives

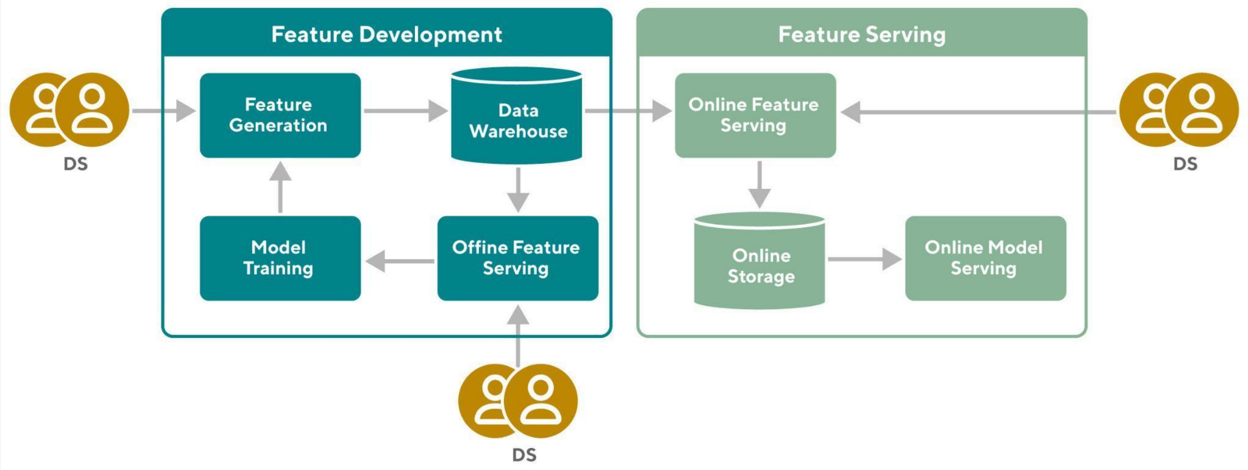
Results and Learnings

Looking back : A year ago

400	100B	60
Unique Features	**Daily Feature Values**	**Number of jobs**

How did our legacy systems look?

- Efficient feature store
- ETL framework with a robust warehouse
- Manual steps for everything else



Pain points

Fragmentation hampers velocity

Data Scientists have to interface with many loosely coupled systems

Infrastructure evolution is low

Improving best practices and integrations takes way too long

No control plane

Maintaining features requires more than just code

Agenda

Feature Engineering at Doordash

Reimagining an ideal Feature Platform

Fabricator : Overview

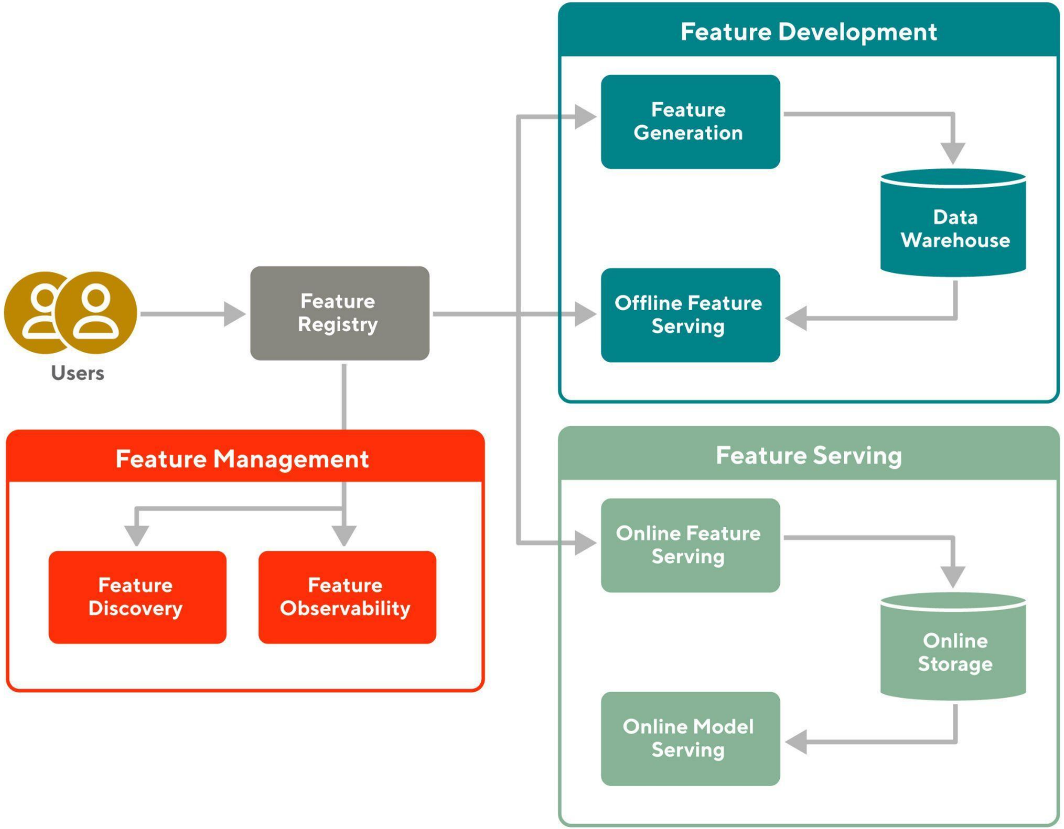
Architecture deep dives

Results and Learnings

What does an ideal platform look like?

- Single entrypoint
- Semantic feature representation
- Simplified abstractions
- High iteration velocity
- Automable feature lifecycle management

Architecture of an ideal platform



Agenda

Feature Engineering at Doordash

Reimagining an ideal Feature Platform

Fabricator : Overview

Architecture deep dives

Results and Learnings

Fabricator Vision

Enable Data Scientists to **declaratively** define efficient **end-to-end** feature pipelines and **automate** the operational lifecycle of features.

Three core components

Centralized Declarative Registry

Provide an **entrypoint** that allows ML practitioners to define **E2E feature semantics** in simple **abstractions.**

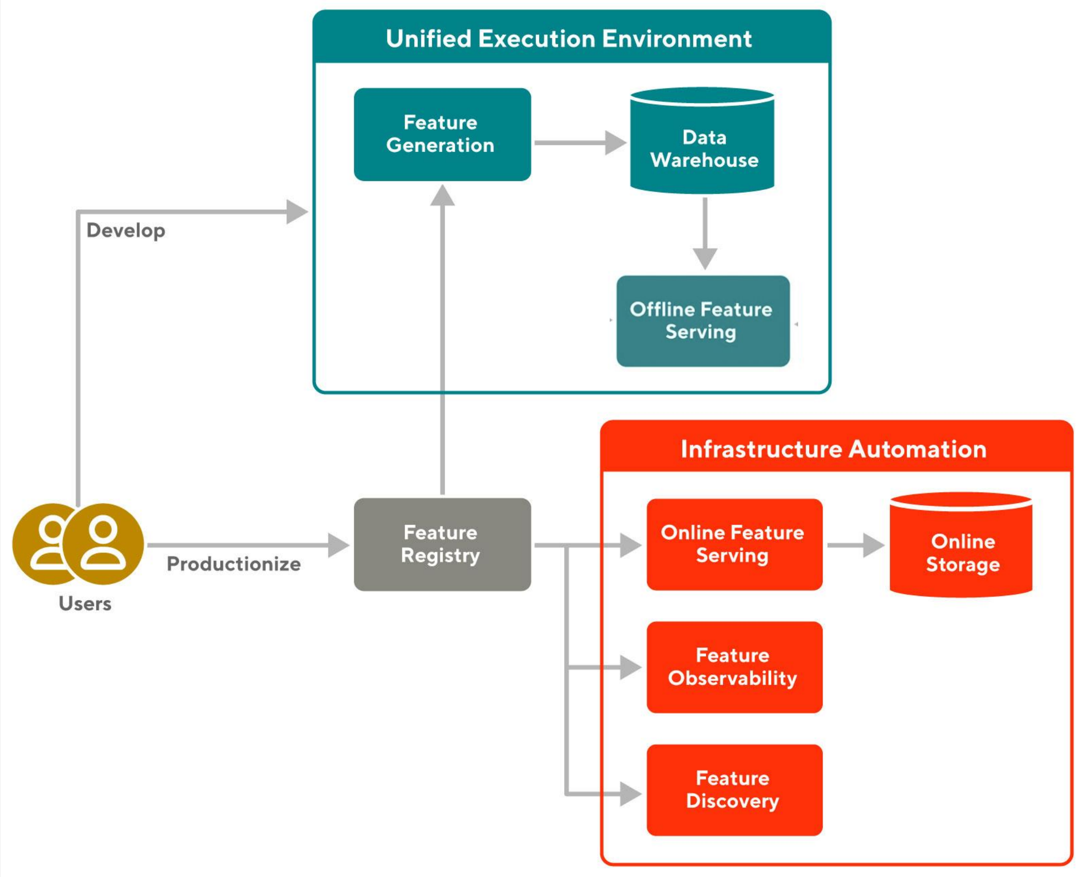
Unified Execution Environment

Provide an execution environment with **simple APIs** for **high iteration velocity**.

Infrastructure Automation

Provide an **automated** integration for all other **downstream operations**.

Fabricator : Architecture



Agenda

Feature Engineering at Doordash

Reimagining an ideal Feature Platform

Fabricator : Overview

Architecture deep dives

Results and Learnings

Feature Registry

- Simple **YAML** definitions for feature semantics
- **Protobuf** backed schema for YAML objects
- **DB backed service** for global access for definitions
- **Continuously deployed** for every change

Feature Semantics

An E2E pipeline requires only a few YAML definitions.

- Source
- Sink
- Feature

```yaml
source:
  name: consumer_engagement_features
  storage_spec:
    type: DELTA_LAKE
    table_name: dimension_consumer_engagement_features
  compute_spec:
    type: SPARK
    spark_spec:
      file: ...
      resource_overrides: ...
  trigger_spec:
    upstreams: ...

sink:
  name: search-redis
  type: REDIS
  redis_spec:
    cluster_node: ...

feature:
  name: caf_consumer_clicks
  entities:
    - consumer_id
  source: consumer_engagement_features
  materialize_spec:
    sink: search-redis
```

Feature Semantics

An E2E pipeline requires only a few YAML definitions.

- **Source**
- Sink
- Feature

```yaml
source:
  name: consumer_engagement_features
  storage_spec:
    type: DELTA_LAKE
    table_name: dimension_consumer_engagement_features
  compute_spec:
    type: SPARK
    spark_spec:
      file: ...
      resource_overrides: ...
  trigger_spec:
    upstreams: ...

sink:
  name: search-redis
  type: REDIS
  redis_spec:
    cluster_node: ...

feature:
  name: caf_consumer_clicks
  entities:
    - consumer_id
  source: consumer_engagement_features
  materialize_spec:
    sink: search-redis
```

Feature Semantics

An E2E pipeline requires only a few YAML definitions.

- **Source**
    - **Generation**
- Sink
- Feature

```yaml
source:
  name: consumer_engagement_features
  storage_spec:
    type: DELTA_LAKE
    table_name: dimension_consumer_engagement_features
  compute_spec:
    type: SPARK
    spark_spec:
      file: ...
      resource_overrides: ...
  trigger_spec:
    upstreams: ...

sink:
  name: search-redis
  type: REDIS
  redis_spec:
    cluster_node: ...

feature:
  name: caf_consumer_clicks
  entities:
    - consumer_id
  source: consumer_engagement_features
  materialize_spec:
    sink: search-redis
```

Feature Semantics

An E2E pipeline requires only a few YAML definitions.

- **Source**
    - **Generation**
- Sink
- Feature

```yaml
source:
  name: consumer_engagement_features
  storage_spec:
    type: DELTA_LAKE
    table_name: dimension_consumer_engagement_features
  compute_spec:
    type: SNOWFLAKE_SQL
    snowflake_spec:
      sqls:
        - >-
          SELECT ...
        - >-
          SELECT ...
  trigger_spec:
    upstreams: ...
  metadata_spec:
    user: dsml-search

sink:
  name: search-redis
  type: REDIS
  redis_spec:
    cluster_node: ...

feature:
  name: caf_consumer_clicks
  entities:
    - consumer_id
  source: consumer_engagement_features
  materialize_spec:
    sink: search-redis
```

Feature Semantics

An E2E pipeline requires only a few YAML definitions.

- **Source**
  - **Generation**
- Sink
- Feature

```yaml
source:
  name: consumer_engagement_features
  storage_spec:
    type: DELTA_LAKE
    table_name: dimension_consumer_engagement_features
  type: REALTIME
  compute_spec:
    type: RIVIERA
    riviera_spec:
      kafka_sources:
        - cluster: default
          topic: delivery_lifecycle_events
      sql: >-
        SELECT ...
  trigger_spec:
    upstreams: ...
  metadata_spec:
    user: dsml-search

sink:
  name: search-redis
  type: REDIS
  redis_spec:
    cluster_node: ...

feature:
  name: caf_consumer_clicks
  entities:
    - consumer_id
  source: consumer_engagement_features
  materialize_spec:
    sink: search-redis
```

Feature Semantics

An E2E pipeline requires only a few YAML definitions.

- **Source**
  - **Generation**
  - **Storage**
- Sink
- Feature

```yaml
source:
  name: consumer_engagement_features
  storage_spec:
    type: DELTA_LAKE
    table_name: dimension_consumer_engagement_features
  compute_spec:
    type: SPARK
    spark_spec:
      file: ...
      resource_overrides: ...
  trigger_spec:
    upstreams: ...

sink:
  name: search-redis
  type: REDIS
  redis_spec:
    cluster_node: ...

feature:
  name: caf_consumer_clicks
  entities:
    - consumer_id
  source: consumer_engagement_features
  materialize_spec:
    sink: search-redis
```

Feature Semantics

An E2E pipeline requires only a few YAML definitions.

- **Source**
  - **Generation**
  - **Storage**
  - **Orchestration**
- Sink
- Feature

```yaml
source:
  name: consumer_engagement_features
  storage_spec:
    type: DELTA_LAKE
    table_name: dimension_consumer_engagement_features
  compute_spec:
    type: SPARK
    spark_spec:
      file: ...
      resource_overrides: ...
  trigger_spec:
    upstreams: ...

sink:
  name: search-redis
  type: REDIS
  redis_spec:
    cluster_node: ...

feature:
  name: caf_consumer_clicks
  entities:
    - consumer_id
  source: consumer_engagement_features
  materialize_spec:
    sink: search-redis
```

Feature Semantics

An E2E pipeline requires only a few YAML definitions.

- Source
- **Sink**
- Feature

```yaml
source:
  name: consumer_engagement_features
  storage_spec:
    type: DELTA_LAKE
    table_name: dimension_consumer_engagement_features
  compute_spec:
    type: SPARK
    spark_spec:
      file: ...
      resource_overrides: ...
  trigger_spec:
    upstreams: ...

sink:
  name: search-redis
  type: REDIS
  redis_spec:
    cluster_node: ...

feature:
  name: caf_consumer_clicks
  entities:
    - consumer_id
  source: consumer_engagement_features
  materialize_spec:
    sink: search-redis
```

Feature Semantics

An E2E pipeline requires only a few YAML definitions.

- Source
- Sink
- **Feature**
  - **Entities**
  - **Serving**

```yaml
source:
  name: consumer_engagement_features
  storage_spec:
    type: DELTA_LAKE
    table_name: dimension_consumer_engagement_features
  compute_spec:
    type: SPARK
    spark_spec:
      file: ...
      resource_overrides: ...
  trigger_spec:
    upstreams: ...

sink:
  name: search-redis
  type: REDIS
  redis_spec:
    cluster_node: ...

feature:
  name: caf_consumer_clicks
  entities:
    - consumer_id
  source: consumer_engagement_features
  materialize_spec:
    sink: search-redis
```

Benefits of the design

Evolution is easy

PB based backend makes our definitions robust to extension

Support for infrastructure flexibility

New storage and compute paradigms can be adopted without significant shifts

Global availability

Every downstream has immediate access to definitions

Unified Execution Environment

- Library suite that **bridges** registry and infrastructure
- Enables **contextual executions** of registry definitions
- Provides **black box optimizations**

Contextual Executions

Pythonic wrappers around YAML definitions designed to “execute” the YAMLs efficiently

```python
class FeatureContext(BaseContext):
    def __init__(
        self,
        features,
        entities,
        table_name,
        storage_type="datalake",
        env="staging",
    ):
```

```python
class SparkFeatureUpload:
    def __init__(self, context: FeatureContext):
        self.context = context
        self.df = None
```

```python
context = FeatureContext.from_source("consumer_engagement_features")
job = SparkFeatureUpload(context)
job.run()
```

Contextual Executions

Context objects wrap registry objects within a Python wrapper.

```
class FeatureContext(BaseContext):
    def __init__(
        self,
        features,
        entities,
        table_name,
        storage_type="datalake",
        env="staging",
    ):
```

```
class SparkFeatureUpload:
    def __init__(self, context: FeatureContext):
        self.context = context
        self.df = None
```

```
context = FeatureContext.from_source("consumer_engagement_features")
job = SparkFeatureUpload(context)
job.run()
```

Contextual Executions

Upload jobs provide an optimized and efficient process to execute a Context

```python
class FeatureContext(BaseContext):
    def __init__(
        self,
        features,
        entities,
        table_name,
        storage_type="datalake",
        env="staging",
    ):
```

```python
class SparkFeatureUpload:
    def __init__(self, context: FeatureContext):
        self.context = context
        self.df = None
```

```python
context = FeatureContext.from_source("consumer_engagement_features")
job = SparkFeatureUpload(context)
job.run()
```

Contextual Executions

User code is a highly condensed expression of the registry definitions

```
class FeatureContext(BaseContext):
    def __init__(
        self,
        features,
        entities,
        table_name,
        storage_type="datalake",
        env="staging",
    ):
```

```
class SparkFeatureUpload:
    def __init__(self, context: FeatureContext):
        self.context = context
        self.df = None
```

```
context = FeatureContext.from_source("consumer_engagement_features")
job = SparkFeatureUpload(context)
job.run()
```

Benefits of the design

Most jobs are no-code

Unless you need customizations, same code executes multiple YAMLs

High fidelity testing

Notebook clusters mimic production job setup.

Efficient execution

Users don't have to optimize for different storage or compute choices

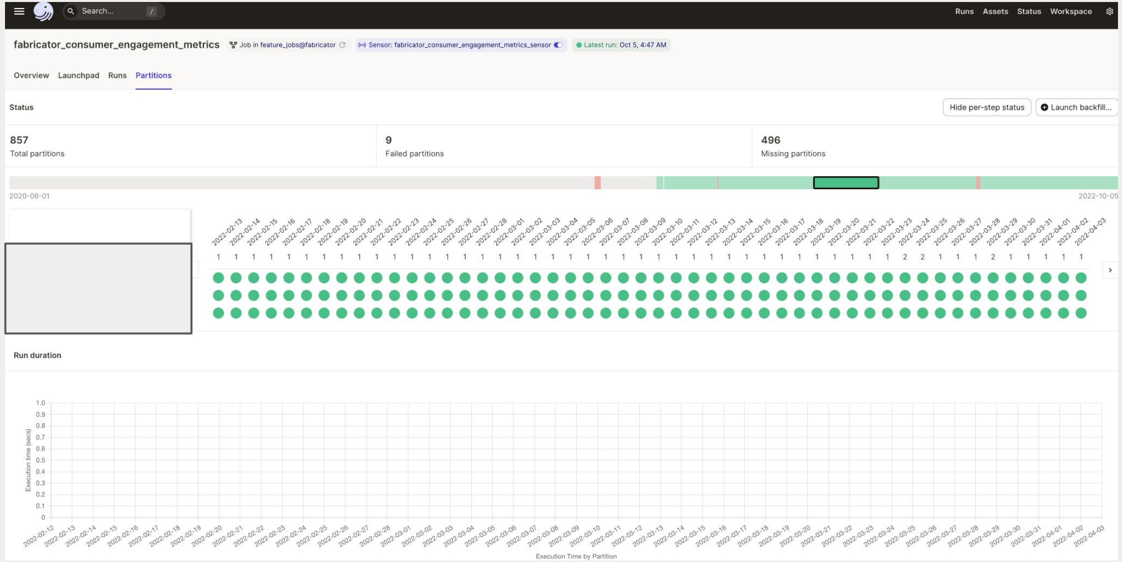
Infrastructure Automation

A central registry and a unified library suite enable us to provide every **downstream integration** to a feature definition for free

Orchestration

- Automated DAG construction
- Flexible choice for orchestrator
- Date partitioning
- Scalable and parallelized backfilling.

```yaml
source:
  name: consumer_engagement_features
  storage_spec:
    type: DELTA_LAKE
    table_name: dimension_consumer_engagement_features
  compute_spec:
    type: SPARK
    spark_spec:
      file: ...
      resource_overrides: ...
  trigger_spec:
    upstreams: ...
```



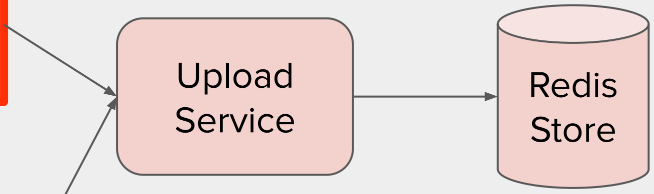
Online Serving

Automate materialization of features to our scalable feature store

```yaml
source:
  name: consumer_engagement_features
  storage_spec:
    type: DELTA_LAKE
    table_name: dimension_consumer_engagement_features
  compute_spec:
    type: SPARK
    spark_spec:
      file: ...
      resource_overrides: ...
  trigger_spec:
    upstreams: ...

sink:
  name: search-redis
  type: REDIS
  redis_spec:
    cluster_node: ...

feature:
  name: caf_consumer_clicks
  entities:
    - consumer_id
  source: consumer_engagement_features
  materialize_spec:
    sink: search-redis
```

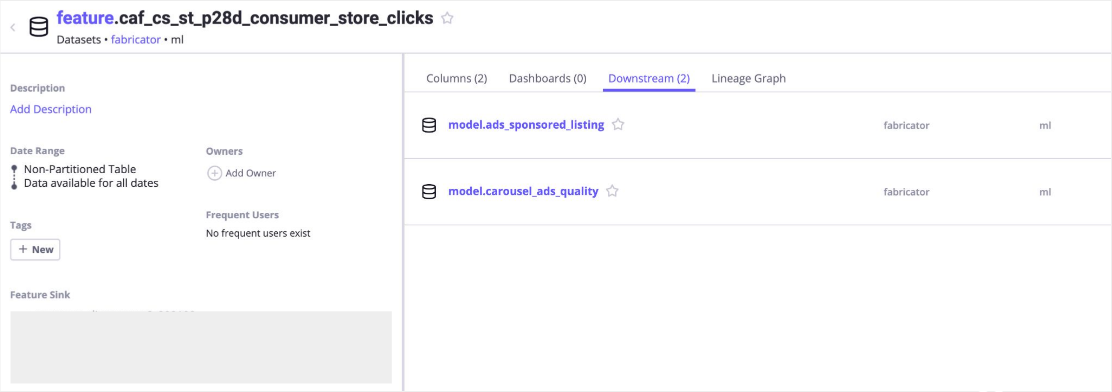


Feature Discovery

Automate registry synchronization with Amundsen

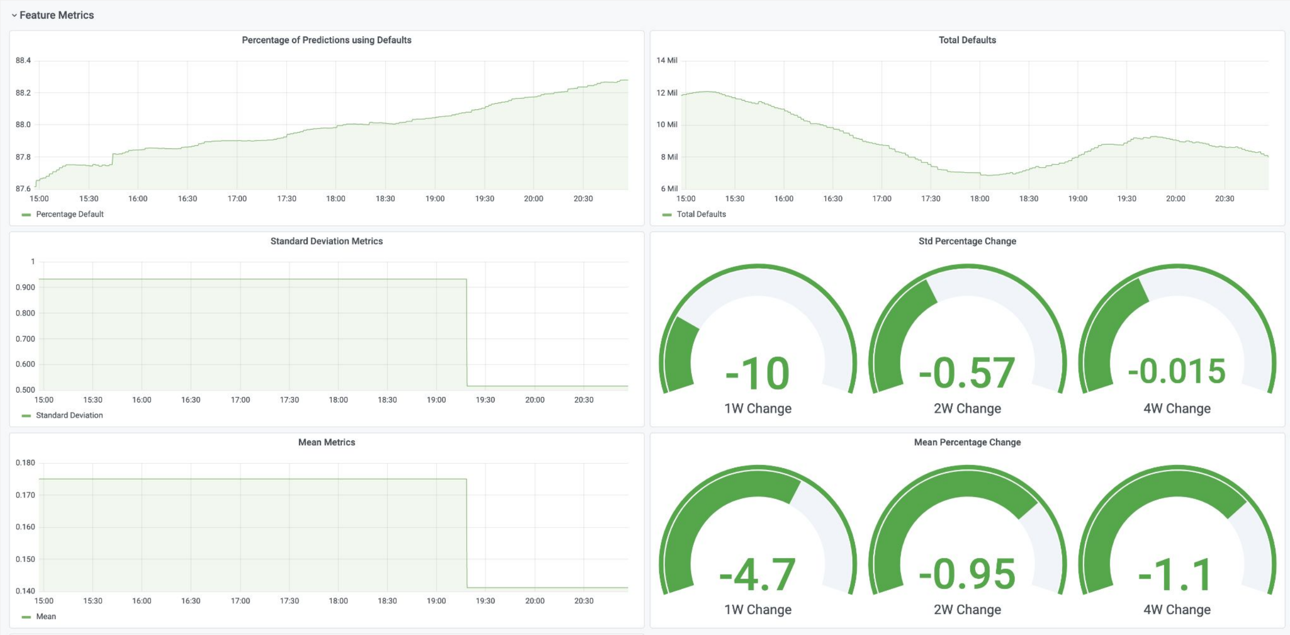
- Registry enables metadata extractors
- Company wide integration enable lineage tracking

```yaml
source:
  name: consumer_engagement_features
  storage_spec:
    type: DELTA_LAKE
    table_name: dimension_consumer_engagement_features
  compute_spec:
    type: SPARK
    spark_spec:
      file: ...
      resource_overrides: ...
  trigger_spec:
    upstreams: ...
  metadata_spec:
    user: dsml-search
```



Feature Observability

Automate feature observability with Chronosphere



Agenda

Feature Engineering at Doordash

Reimagining an ideal Feature Platform

Fabricator : Overview

Architecture deep dives

Results and Learnings

Results

>2K	1T	500
Unique Features	**Daily Feature Values**	**Number of jobs**

Learnings

Build products, not systems

Adoption was slower when users interfaced with systems, rather than a single product

Make it easy to do the right thing

Simplify the most prolific patterns, and leave room for customization

Reliability lags behind growth

Adoption is not without risks, and can come at the cost of robustness.

Thank you!