DATA+AI SUMMIT
BY databricks

SOON For Near Real-time Data

How Coinbase built and optimized SOON (Spark cOntinuOus iNgestion), a streaming ingestion framework

Databricks
2023

Chen Guo
Staff Software Engineer, Data Platform & Service Team

Agenda

Problems to Solve

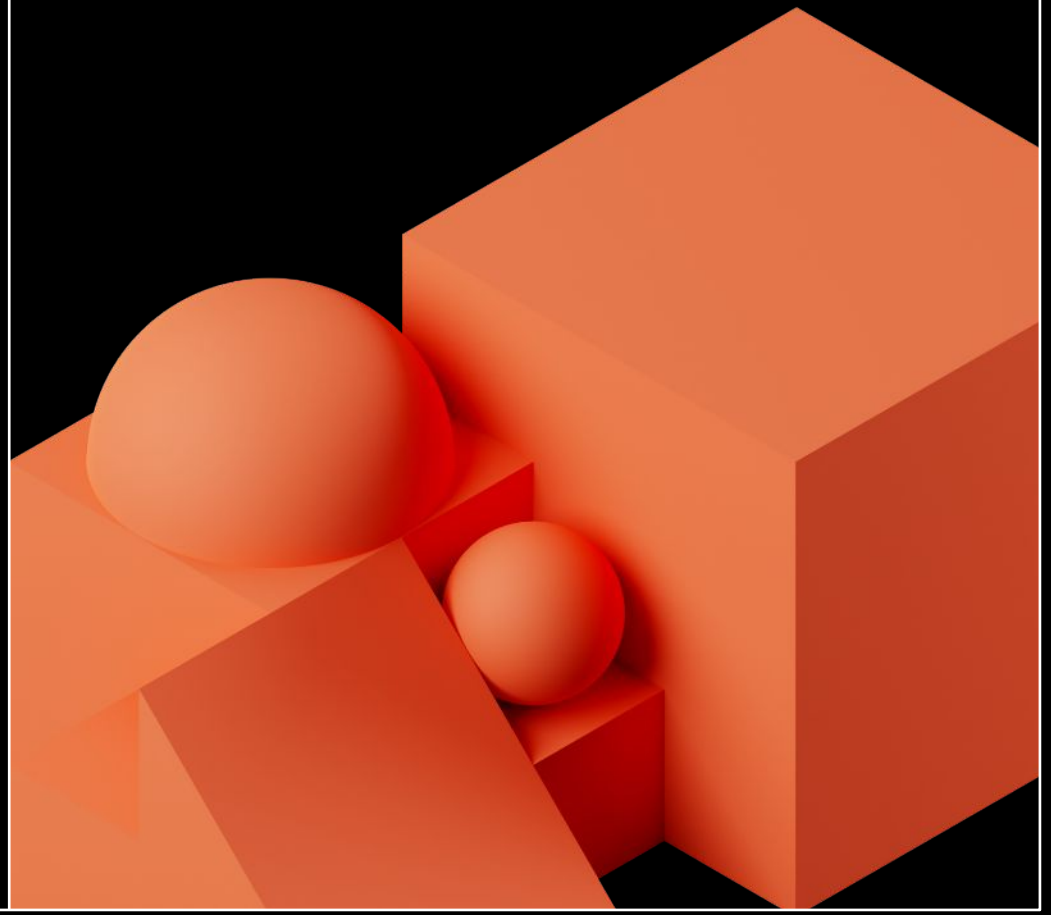
SOON Ecosystem

SOON Basics

Optimizations In Depth

Incremental Load to Other Systems

Problems to Solve



Problems to Solve

High Data Replication Latency

- Why is it important
  - Ad hoc incident analysis, customer supports
  - Dashboards or products that require near real-time metrics
  - Production monitoring, anomaly detection, etc...
- Many tables are built/replicated in full on a daily basis
  - Incrementally built tables are faster, but may not have perfect data quality
  - Up-to-date data in OLTP stores
  - Data in multiple places
  - Replicate different varieties of tables to a centralized place in near real-time

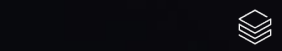
Problems to Solve

Diverse architectures/designs for ingesting different sources, like databases and Kafka

- Multiple codebases to maintain
  - Can be written in different programming languages
  - High operational overhead
  - Long learning curve for new team members
- Inconsistent user experiences
  - Different wikis and onboarding guides to read
  - Different configuration domain-specific languages (DSLs) to learn

SOON (Spark cOntinuOus iNgestion)

A unified, configuration-driven streaming ingestion framework from Kafka to Delta Lake implemented using Spark Structured Stream APIs



Problems to Solve

Our Solution – SOON

Unify DB replication and Kafka ingestion

- Table Replication based on Change Data Capture(CDC) events
- Normal non-CDC events

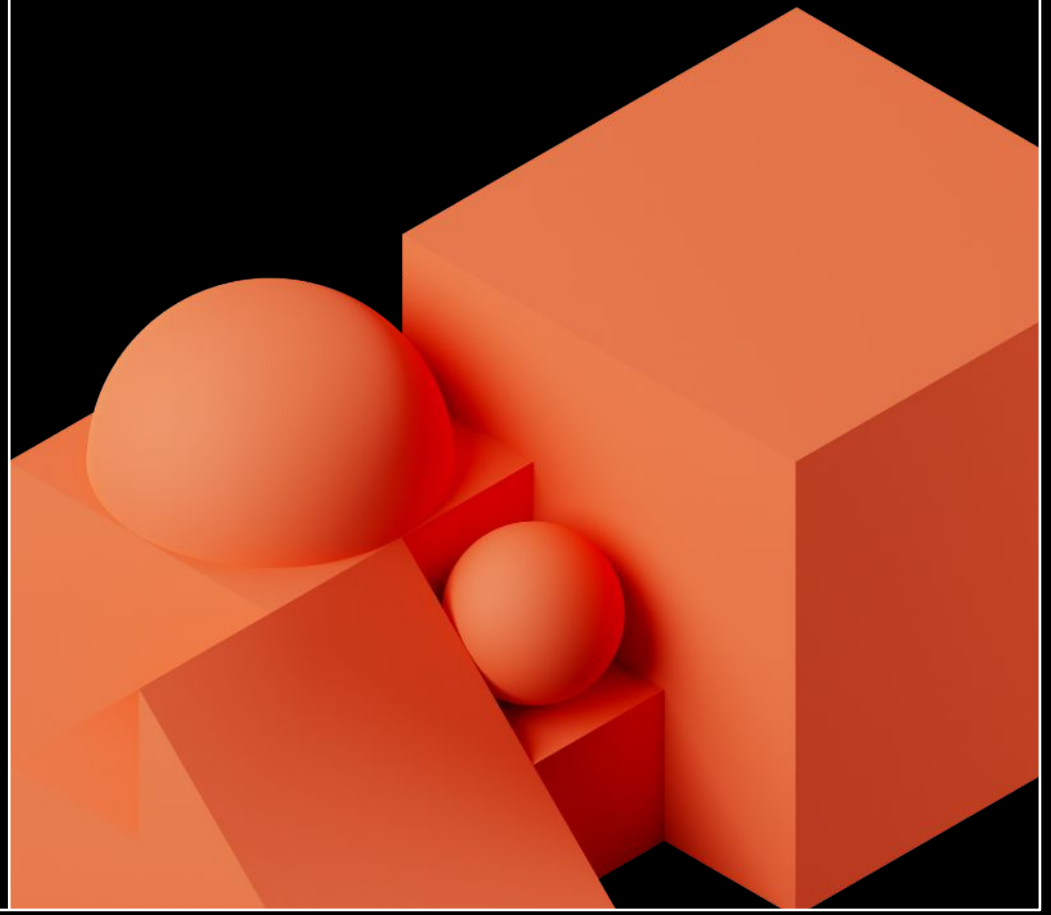
One generic framework for all scenarios

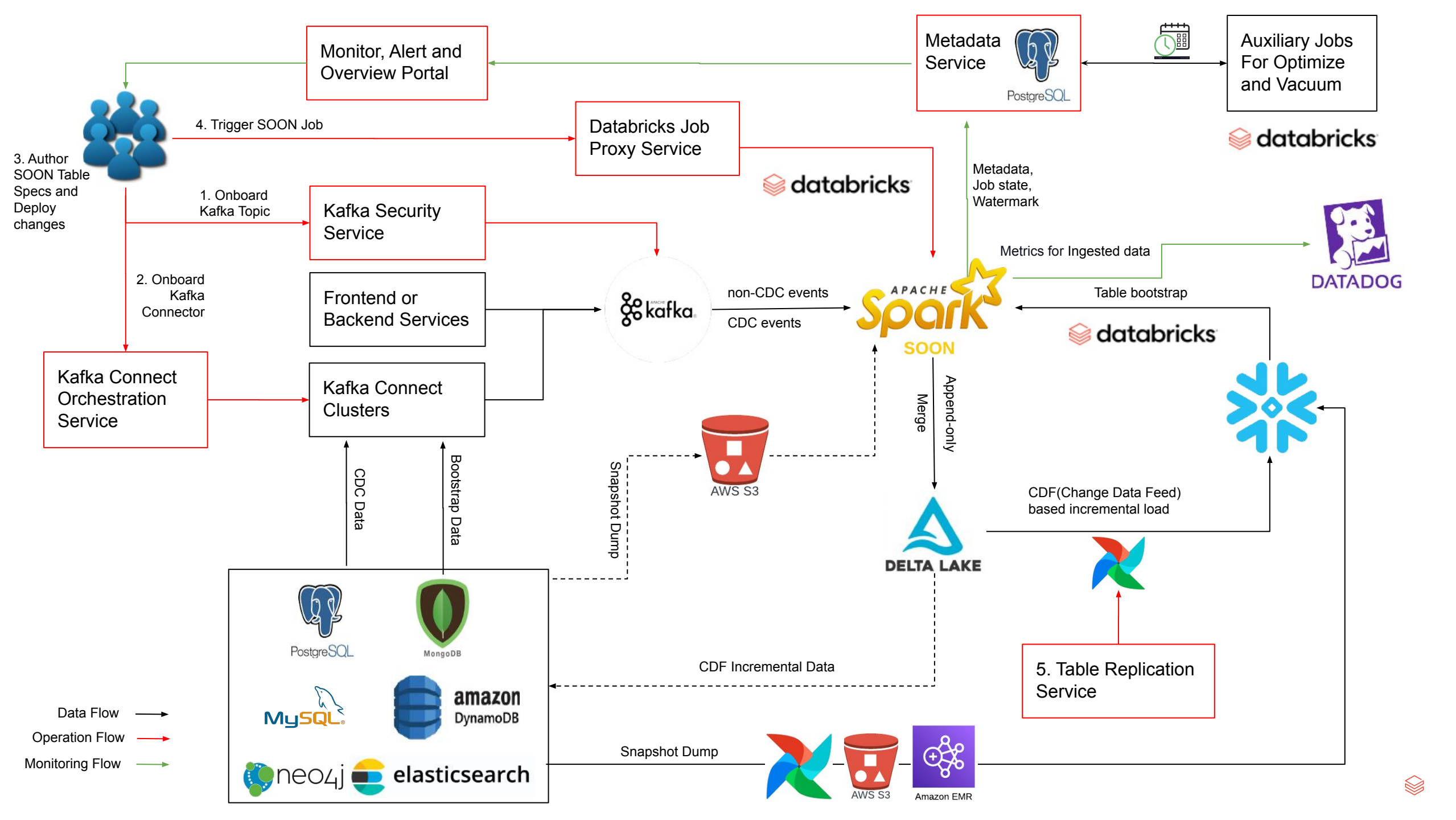
- Support append-only and merge-updates for both CDC and non-CDC

One unified onboarding experience for the users

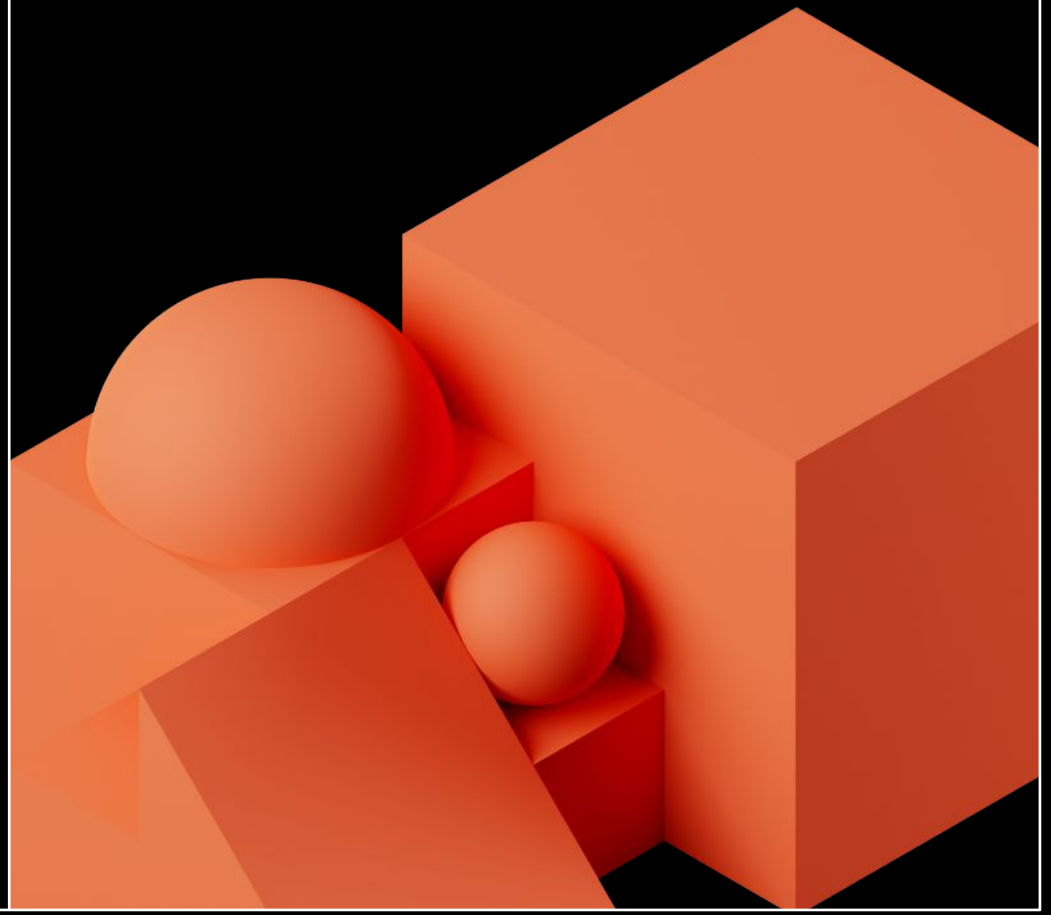
One framework to develop and maintain for the team

SOON Ecosystem




Monitor, Alert and Overview Portal
Metadata Service
PostgreSQL
Auxiliary Jobs For Optimize and Vacuum
databricks
3. Author SOON Table Specs and Deploy changes
4. Trigger SOON Job
Databricks Job Proxy Service
databricks
Metadata, Job state, Watermark
1. Onboard Kafka Topic
Kafka Security Service
Metrics for Ingested data
DATADOG
2. Onboard Kafka Connector
Frontend or Backend Services
kafka
non-CDC events
CDC events
APACHE Spark
SOON
Table bootstrap
databricks
Kafka Connect Orchestration Service
Kafka Connect Clusters
Merge
Append-only
AWS S3
Snapshot Dump
CDC Data
Bootstrap Data
CDF(Change Data Feed) based incremental load
DELTA LAKE
PostgreSQL
MongoDB
MySQL
amazon DynamoDB
neo4j
elasticsearch
CDF Incremental Data
5. Table Replication Service
Snapshot Dump
AWS S3
Amazon EMR
Data Flow
Operation Flow
Monitoring Flow

SOON Basics



SOON Basics

Job Specification DSL

SOON translates a job specification file into a Spark streaming job

- HOCON format
- Support connection to multiple Kafka clusters
- Derived columns support for map-side transformations
- Multiple job specifications can be triggered in the same cluster

```
{
  kafka: {
    cluster: cluster-name
    topic: kafka-topic-name
    …
  }
  outputTable: {
    dev: schema_dev.table_name
    prod: schema_prod.table_name
  }
  rawColumns: [
    {columnName: column1, type: Long}
    {columnName: column2, type: String, isExcluded: true}
  ]
  derivedColumns: [
    {
      columnName: column2_derived, type: String,
      defExpression: "udf(column2)",
      comment: "some comments here"
    }
  ]
  partitionColumns: [
      {columnName: "column2", generatedPartitionType:
"date_hour"}
  ]
  features: [ … ]
  …
}
```

SOON Basics

Append-only Jobs

NO merge keys defined in the DSL

Supported Features

- Supported formats: JSON, Protobuf, any customized binary format
- Physical or generated columns as partition columns
- Backfills

```
xxxColumns: [
  {columnName: c1, type: Long, isPrimaryKey: false}
  {columnName: c2, type: Timestamp, isPrimaryKey: false}
]
```

```
partitionColumns: [
    {columnName: timestamp, generatedPartitionType: date}
    {
      columnName: name, generatedPartitionType: substring,
      substringParams: {
        pos: 0,
        len: 6
      }
    }
  ]
```

SOON Basics

Merge Jobs

Job specification is translated into "MERGE" queries

- Many optimizations to improve merge performance
- CDC-events Merge
  - Standard CDC event schema
  - Only support JSON for now
- non-CDC-events Merge
  - Can be any format
  - Deletes are not supported

```
sourceTable: {
  dev: schema_dev.table_name
  prod: schema_prod.table_name
}

xxxColumns: [
  {columnName: id, type: Long, isPrimaryKey: true}
  {columnName: col, type: Timestamp}
]
```

SOON Basics

CDC Merge Jobs

SOON standard CDC event schema

- Unify all databases' raw CDC schema
- Kafka Connect Single Message Transforms(SMT) to transform raw CDC events into the standard schema

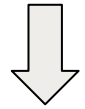
Last Change Selection (LCS)

- Find last change for each merge key
- Window function based on CDC operation time and offsets

Generate and run merge queries

```
outputTable: {
  dev: schema_dev.table_name
  prod: schema_prod.table_name
}

xxxColumns: [
  {columnName: id, type: Long, isPrimaryKey: true}
  {columnName: col, type: Timestamp}
]
```



```
MERGE INTO schema_prod.table_name t
  USING stream_events s
  ON t.id = s.id
  WHEN MATCHED
       and s.operation = "DELETE"
    THEN DELETE
  WHEN MATCHED
       and s.operation != "DELETE"
    THEN UPDATE SET *
  WHEN NOT MATCHED
       and s.operation != "DELETE"
    THEN INSERT *;
```

SOON Basics

SOON Standard CDC Event Schema

oc: operation code

- “I” for “Insert” event
- “D” for “Delete” event
- “U” for “Update” event

ns: namespace

- Composed of <database or schema name>. <table name>
- For differentiating CDC source tables for a shared Kafka CDC topic for multiple tables

An update event example:

```
{
   "oc": "U",
   "ns": "schema.table",
   "ot": 1651743495123,
   "pk": {
      "id1": "123",
      "id2": "abc"
    },
   "va": {
      "field1": "value1",
      "field2": "value2"
    }
}
```

A delete event example:

```
{
   "oc": "D",
   "ns": "schema.table",
   "ot": 1651743495123,
   "pk": {
      "id1": "123",
      "id2": "abc"
    }
}
```

SOON Basics

SOON Standard CDC Event Schema

ot: operation time

- The epoch timestamp when the DB operation happens that creates the CDC event

pk: primary key

- Columns composing the composite primary keys or unique index for a row
- “pk” values are from a post-image after the change. It doesn’t have to come from an immediate post-image after the change

An update event example:

```
{
   "oc": "U",
   "ns": "schema.table",
   "ot": 1651743495123,
   "pk": {
      "id1": "123",
      "id2": "abc"
    },
   "va": {
      "field1": "value1",
      "field2": "value2"
    }
}
```

A delete event example:

```
{
   "oc": "D",
   "ns": "schema.table",
   "ot": 1651743495123,
   "pk": {
      "id1": "123",
      "id2": "abc"
    }
}
```

SOON Basics

SOON Standard CDC Event Schema

va: value

- The rest of the columns for a row excluding the “pk” fields
- Same as the “pk” field, where values are from the same post-image after the change

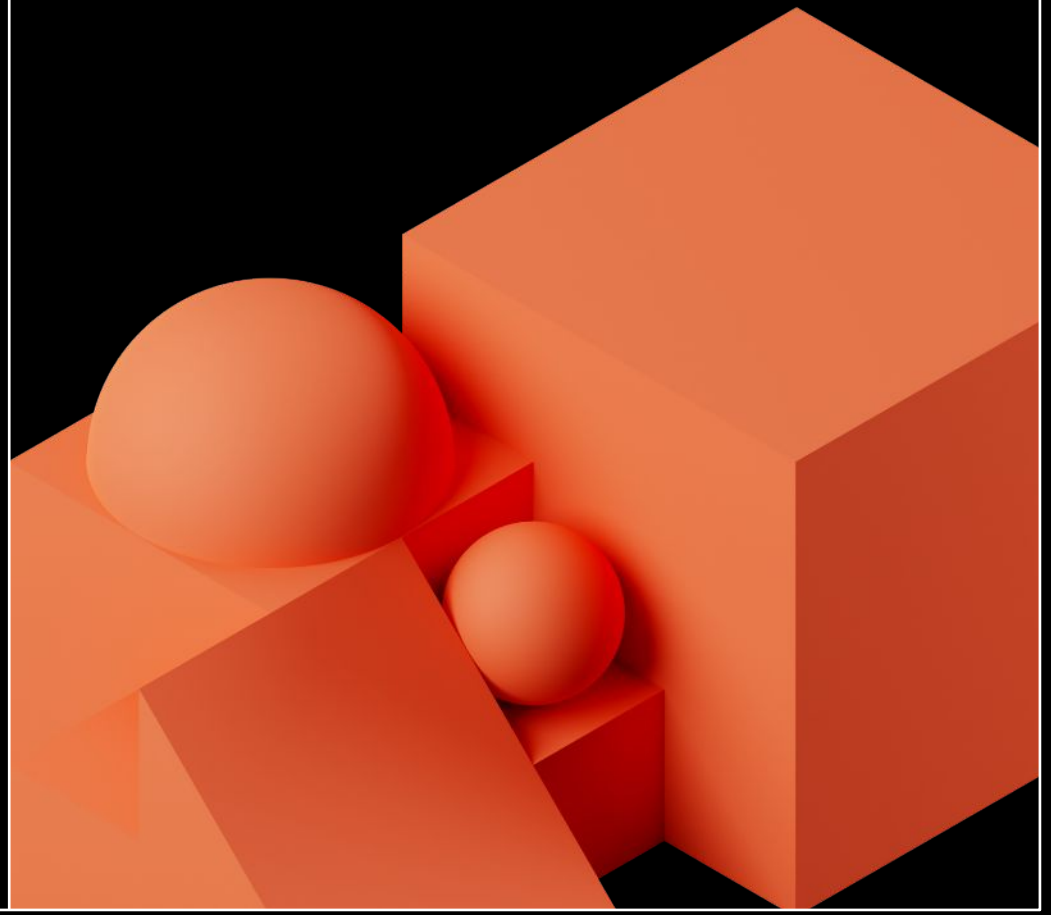
An update event example:

```
{
   "oc": "U",
   "ns": "schema.table",
   "ot": 1651743495123,
   "pk": {
      "id1": "123",
      "id2": "abc"
    },
   "va": {
      "field1": "value1",
      "field2": "value2"
    }
}
```

A delete event example:

```
{
   "oc": "D",
   "ns": "schema.table",
   "ot": 1651743495123,
   "pk": {
      "id1": "123",
      "id2": "abc"
    }
}
```

Optimizations In Depth



Optimizations In Depth

Optimizations For Merge

Reduce amount of data to read from S3 for the target table

- Min-max range merge optimization
- KMeans range merge optimization

Reduce amount of data to rewrite the target table

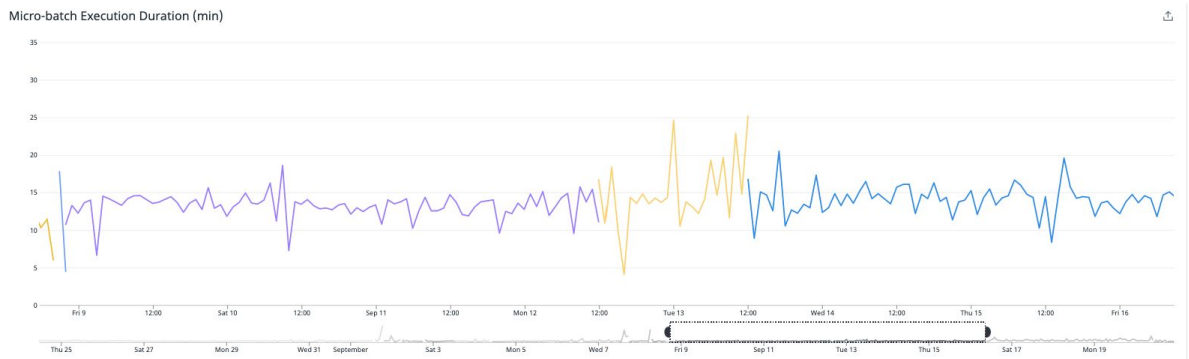
- No-update merge for deduplication (merge non-CDC)

Reduce unnecessary joins with “insert-only” merge

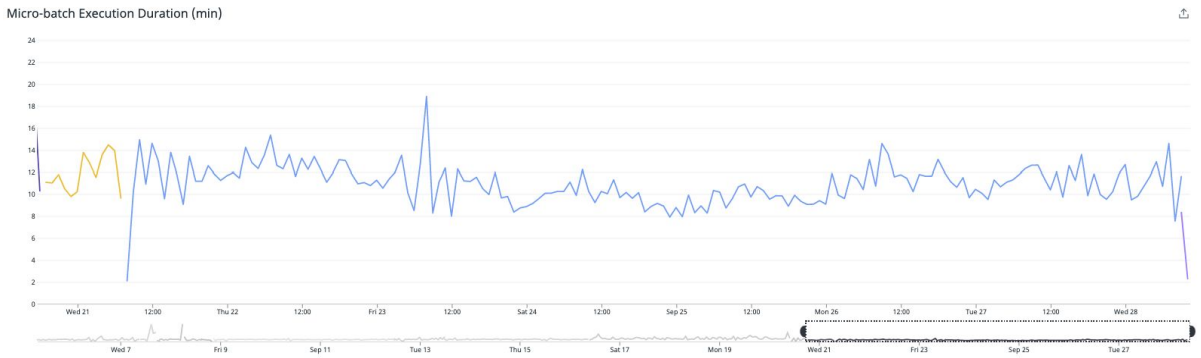
- Merge-with-Insert optimization (merge CDC)

Optimizations In Depth

Optimizations For Merge – MinMax Range Merge Optimization

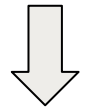


Micro-batch execution time for merge with no optimizations



Micro-batch execution time for merge with MinMax Range Optimization. Performance increases by ~ 20%.

```
xxxColumns: [
  {columnName: id, type: String, isPrimaryKey: true,
enableMinMaxMergeOptimization: true}
]
```



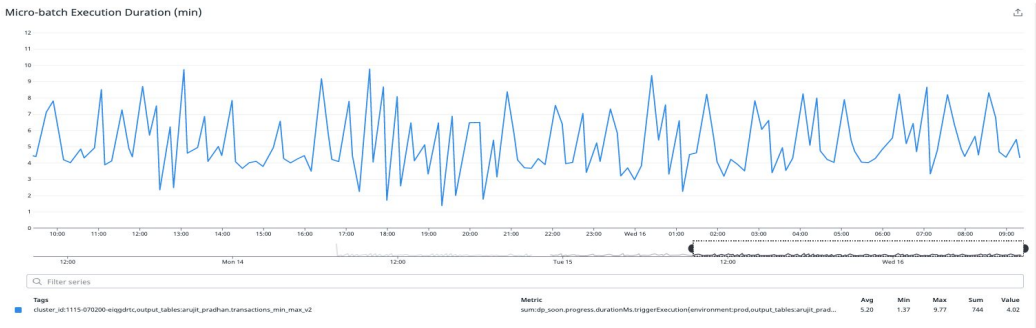
```
MERGE INTO output_db.output_table t
  USING stream_events s
  ON t.id = s.id
    AND t.id >= <MIN_ID> AND t.id <= <MAX_ID>
  WHEN MATCHED
       and s.operation = "DELETE"
    THEN DELETE
  WHEN MATCHED
       and s.operation != "DELETE"
    THEN UPDATE SET *
  WHEN NOT MATCHED
       and s.operation != "DELETE"
    THEN INSERT *;
```

Optimizations In Depth

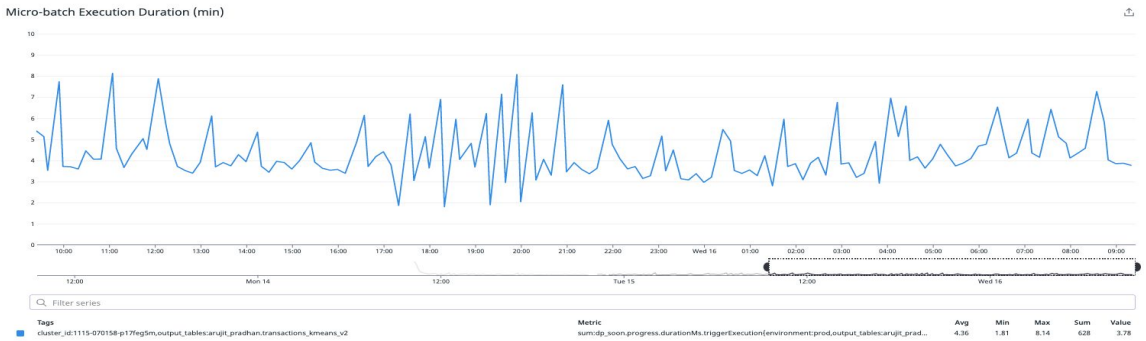
Optimizations For Merge – KMeans Range Merge Optimization

prediction	clusterLeftBound	clusterRightBound
1	2017-12-22 11:21:25.729	2017-12-23 12:18:38.518
2	2020-11-21 07:41:17.044	2020-11-21 07:42:40.168
3	2022-02-11 03:34:20.062	2022-04-28 03:08:29.359
4	2022-05-01 17:33:13.116	2022-06-21 18:46:21.223
5	2022-07-14 02:02:19.538	2022-08-06 08:13:23.976
8	2022-08-16 18:37:01.086	2022-08-31 02:58:22.255
6	2022-09-02 12:44:48.633	2022-09-09 23:20:30.712
7	2022-09-11 14:24:03.561	2022-09-14 03:27:16.964
9	2022-09-14 11:45:27.835	2022-09-16 14:08:28.391
0	2022-09-16 15:42:20.047	2022-09-18 01:29:58.508

A few updates in history can make the MinMax range very wide



Average micro-batch execution time for merge with MinMax Range Optimization: ~5.3 mins

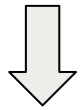


Average micro-batch execution time for merge with KMeans Range Optimization: ~4.3 mins. Additional ~15-20% performance gain over MinMax Range Optimization

Optimizations In Depth

Optimizations For Merge – KMeans Range Merge Optimization

```
xxxColumns: [
  {columnName: id, type: String, isPrimaryKey: true}
  {columnName: created_at, type: Timestamp, kMeansMergeOptimizationType: Timestamp}
]
```



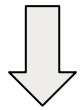
```
MERGE INTO output_db.output_table t
  USING stream_events s
  ON t.id = s.id
   AND (t.`created_at` >= <MIN in Bucket-1> AND t.`created_at` <= <MAX in Bucket-1>
          OR t.`created_at` >= <MIN in Bucket-2> AND t.`created_at` <= <MAX in Bucket-2>
          OR t.`created_at` >= <MIN in Bucket-3> AND t.`created_at` <= <MAX in Bucket-3>)
  WHEN MATCHED
       and s.operation = "DELETE"
    THEN DELETE
  WHEN MATCHED
       and s.operation != "DELETE"
    THEN UPDATE SET *
  WHEN NOT MATCHED
       and s.operation != "DELETE"
    THEN INSERT *;
```

Optimizations In Depth

Optimizations For Merge – No-update Merge For Deduplication

A feature that can be turned on for de-duplication use cases in merge non-CDC scenario

```
xxxColumns: [
  {columnName: id, type: String, isPrimaryKey: true}
  {columnName: event_time, type: Timestamp,
enableMinMaxMergeOptimization: true}
]
partitionColumns: [
    {columnName: event_time, generatedPartitionType: date_hour}
]
features: [
  {
    name: no_update_merge
  }
]
```



```
MERGE INTO output_db.output_table t
  USING stream_events s
  ON t.id = s.id
     AND (t.event_time >= <min_event_time> AND t.event_time <=
<max_event_time>)
  WHEN NOT MATCHED
    THEN INSERT *;
```

Optimizations In Depth

Optimizations For Merge - Merge-with-Insert optimization

Use INSERT query instead of MERGE when micro-batch only contains “INSERT” events

- Fall back to normal merge-CDC query otherwise

Applicable use case: when majority of the CDC changes are “insert”s

- Partitioned Postgres tables where "active" and "historic" data are in different tables
- Mostly "insert" events for the "historic" table (records are immutable)
  - “Historic” table can be huge
  - Deletes due to GDPR

Optimizations In Depth

Optimizations For Better Read and Less Storage

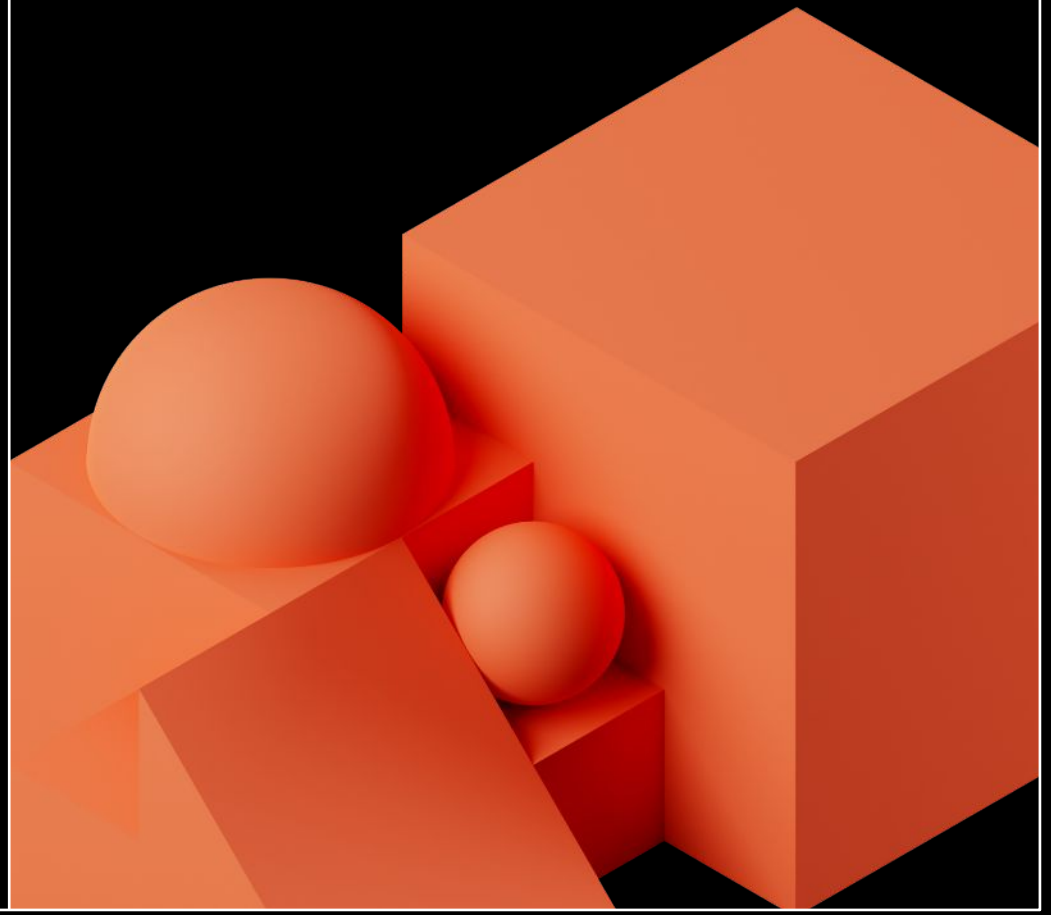
Automatically onboarded to scheduled optimize and vacuum pipelines

- Partitioned SOON tables
  - Incremental z-order optimize with partition predicates and z-ordered by merge keys or configured columns
  - High watermarks for optimize runs saved in an external state store
- Non-partitioned SOON tables
  - Full table z-order optimize based on merge keys or configured columns

SOON jobs and optimize jobs coordination

- External state store to track job state: ZORDER, MERGE, or IDLE

Incremental Load to Other Systems



Incremental Load to Other Systems

Incremental Load to Snowflake

Change Data Feed (CDF) based incremental load to Snowflake for merge-updated tables

- Support hard deletes
- Airflow dags to export changes of different types to S3, and load incrementally using delete + merge queries in Snowflake

```
16 select * from table_changes('events.test', 1, 5);
```

Results +

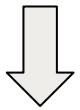
#	id_col	value_col	created_at	_change_type	_commit_version	_commit_timestamp
1	2	200	2023-05-05 17:56:58.290	insert	2	2023-05-05 17:56:59.000
2	3	300	2023-05-05 17:57:05.333	insert	3	2023-05-05 17:57:06.000
3	3	301	2023-05-05 17:57:05.333	update_postimage	4	2023-05-05 17:57:41.000
4	3	300	2023-05-05 17:57:05.333	update_preimage	4	2023-05-05 17:57:41.000
5	1	100	2023-05-05 17:56:37.798	delete	5	2023-05-05 17:57:46.000

CDF changes with _change_type, _commit_version, _commit_timestamp columns

Incremental Load to Other Systems

Incremental deletes to Snowflake

```
CREATE TABLE cdc_schema._tmp_src_delete_<epoch>
USING PARQUET LOCATION 's3a://<path>' AS
SELECT
     pk
FROM
     TABLE_CHANGES('cdc_schema.src',
<commit_version_dest_high_watermark>,
<src_max_commit_version>)
WHERE
    _change_type = 'delete'
```

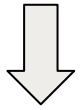


Load S3 path as Snowflake table
output_db._DELETED_ROWS

```
DELETE FROM output_db.dest t
     USING output_db._DELETED_ROWS_dest s
WHERE
     t.pk = s.pk
```

Incremental upserts to Snowflake

```
CREATE TABLE cdc_schema._tmp_src_upsert_<epoch> USING
PARQUET LOCATION 's3://<path>' AS
SELECT
     *,
     <src_max_commit_version>
FROM
     cdc_schema.src@<src_max_commit_version>
WHERE
     timestamp_watermark_column >=
<dest_timestamp_high_watermark>-<small_lookback_window>
```



Load S3 path as Snowflake table
output_db._UPSERT_ROWS

```
MERGE INTO output_db.dest t
     USING output_db._UPSERT_ROWS_dest s
ON t.pk = s.pk
WHEN MATCHED
     THEN UPDATE SET t.col1 = s.col1, t.col2=s.col2
WHEN NOT MATCHED
     THEN INSERT (co1, col2) values (col1, col2);
```

Thank you!

Q&A

Coinbase Engineering Blogs

- SOON (Spark cOntinuOus iNgestion) for near real-time data at Coinbase - Part 1
- Optimizations in SOON (Spark cOntinuOus iNgestion) - Part 2

Chen Guo
Staff Software Engineer, Data Platform & Service Team