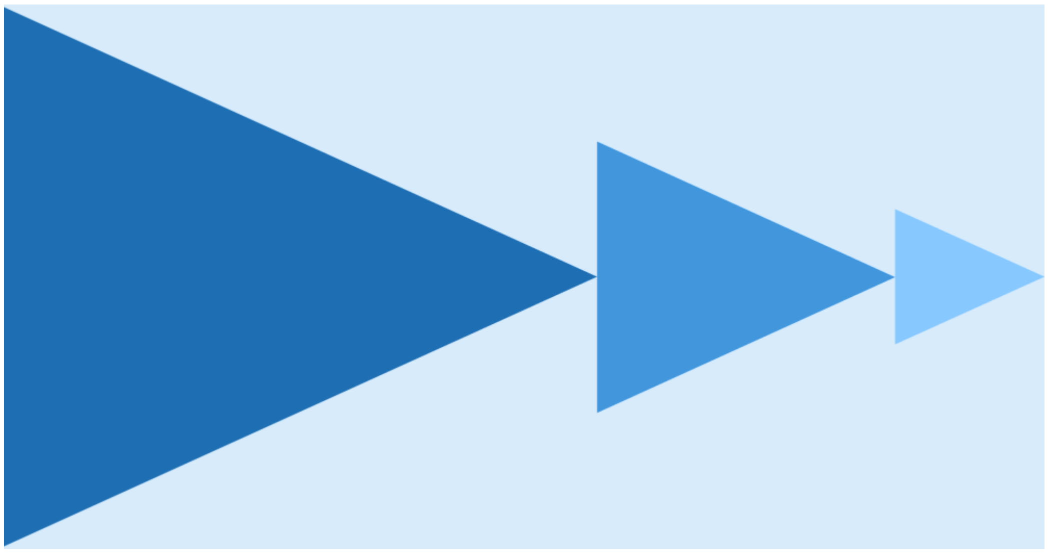
How to Simplify CDC With Delta Lake's Change Data Feed



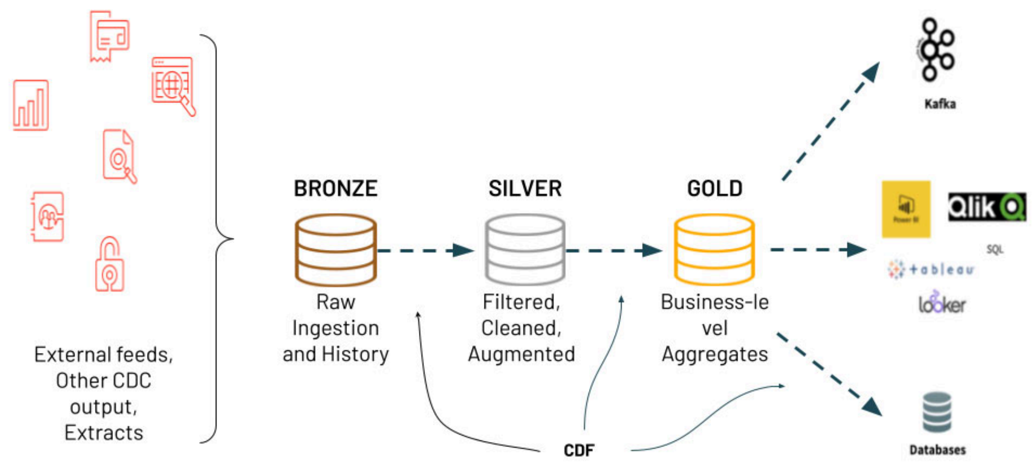
Published: June 9, 2021

Data Engineering | 5 min read

By Surya Sai Turaga and John O'Dwyer

Try this notebook in Databricks

deep dive on the topic here. Typically we see CDC used in an ingestion to analytics architecture called the *medallion architecture*. The medallion architecture that takes raw data landed from source systems and refines the data through bronze, silver and gold tables. CDC and the medallion architecture provide multiple benefits to users since only changed or added data needs to be processed. In addition, the different tables in the architecture allow different personas, such as Data Scientists and BI Analysts, to use the correct up-to-date data for their needs. We are happy to announce the exciting new Change Data Feed (CDF) feature in Delta Lake that makes this architecture simpler to implement and the MERGE operation and log versioning of Delta Lake possible!



Get an early preview of O'Reilly's new ebook for the step-by-step guidance you need to start using Delta Lake.

Why is the CDF feature needed?

Many customers use Databricks to perform CDC, as it is simpler to implement with Delta Lake compared to other Big Data technologies. However, even with the right tools, CDC can still be challenging to execute. We designed CDF to make coding even simpler and address the biggest pain points around CDC, including:

- **Inefficiency** - It can be inefficient to account for non-changing rows since the current version changes are at the file and not the row level.

Here is how Change Data Feed (CDF) implementation helps resolve the above issues:

- **Simplicity and convenience** - Uses a common, easy-to-use pattern for identifying changes, making your code simple, convenient and easy to understand.
- **Efficiency** - The ability to only have the rows that have changed between versions, makes downstream consumption of Merge, Update and Delete operations extremely efficient.

CDF captures changes **only** from a Delta table and is **only** forward-looking once enabled.

Change Data Feed in Action!

Let's dive into an example of CDF for a common use case: financial predictions. The notebook referenced at the top of this blog ingests financial data. Estimated Earnings Per Share (EPS) is financial data from analysts predicting a company's quarterly earnings per share. The raw data can come from many different sources and from multiple analysts for multiple stocks.

With the CDF feature, the data is simply inserted into the bronze table (raw ingestion), then filtered, cleaned and augmented in the silver table and, finally, aggregate values are computed in the gold table based on the changed data in the silver table.

While these transformations can get complex, thankfully, now the row-based CDF feature is simple and efficient. But how do you use it? Let's dig in!

NOTE: The example here focuses on the SQL version of CDF and also on a specific way to use the operations, to evaluate variations, please

Enabling CDF on a Delta Lake Table

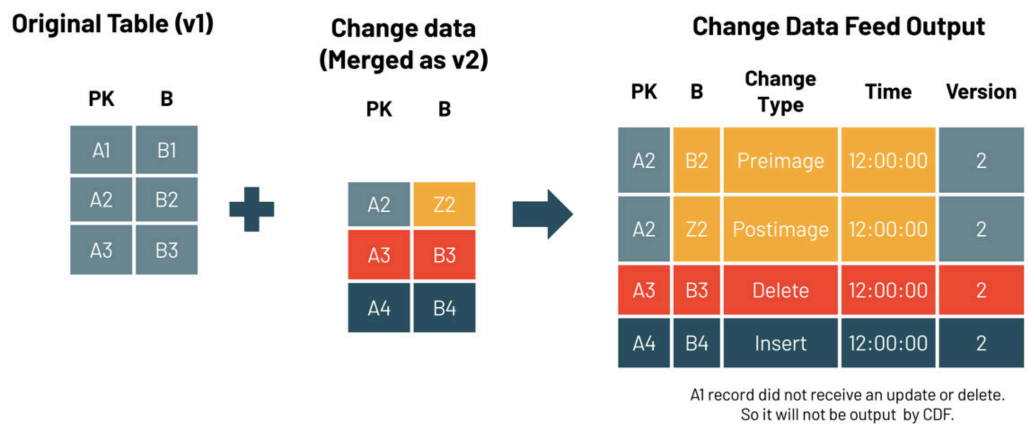
To have the CDF feature available on a table, you must first enable the feature on said table. Below is an example of enabling CDF for the bronze table at table creation. You can also enable CDF on a table as an update to the table. In addition, you can enable CDF on a cluster for all tables created by the cluster. For these variations, please see the documentation here.

```
CREATE TABLE bronze_eps
  (date STRING, stock_symbol STRING, analyst INT, estimated_eps DOUBLE)
  USING DELTA
  TBLPROPERTIES (delta.enableChangeDataFeed = true);
```

Change Data Feed is a forward looking feature, it will capture changes once the table property is set up and not earlier

Querying the change data

To query the change data, use the *table_changes* operation. The example below includes inserted rows and two rows that represent the pre- and post-image of an updated row, so that we can evaluate the differences in the changes if needed. There is also a *delete* Change Type that is returned for deleted rows.

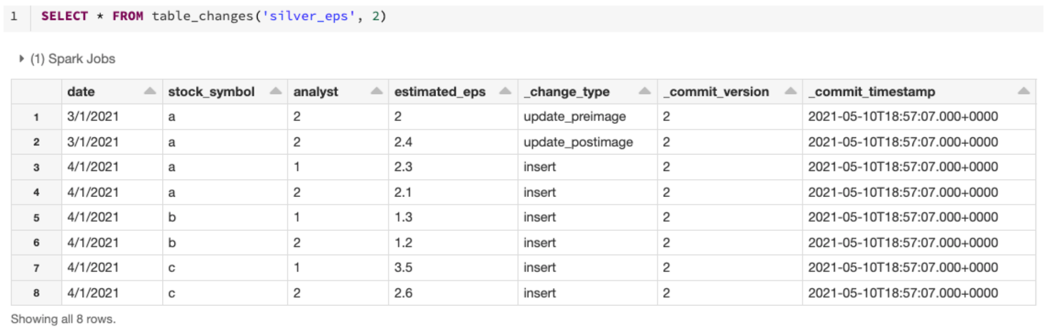

Original Table (v1)
Change data (Merged as v2)
Change Data Feed Output
A1 record did not receive an update or delete. So it will not be output by CDF.

PK	B
A1	B1
A2	B2
A3	B3

PK	B
A2	Z2
A3	B3
A4	B4

PK	B	Change Type	Time	Version
A2	B2	Preimage	12:00:00	2
A2	Z2	Postimage	12:00:00	2
A3	B3	Delete	12:00:00	2
A4	B4	Insert	12:00:00	2

This example accesses the changed records based on the *starting version,* but you can also cap the versions based on the *ending* in Python, Scala, Java and R. For these variations, please see the documentation here.



```sql
SELECT * FROM table_changes('silver_eps', 2)
```

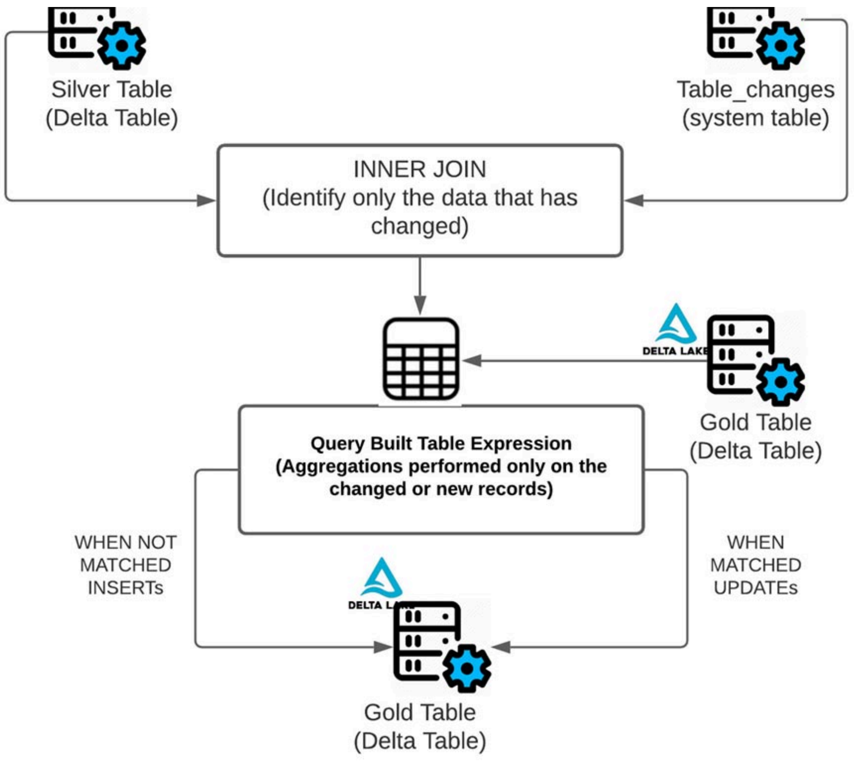
(1) Spark Jobs

	date	stock_symbol	analyst	estimated_eps	_change_type	_commit_version	_commit_timestamp
1	3/1/2021	a	2	2	update_preimage	2	2021-05-10T18:57:07.000+0000
2	3/1/2021	a	2	2.4	update_postimage	2	2021-05-10T18:57:07.000+0000
3	4/1/2021	a	1	2.3	insert	2	2021-05-10T18:57:07.000+0000
4	4/1/2021	a	2	2.1	insert	2	2021-05-10T18:57:07.000+0000
5	4/1/2021	b	1	1.3	insert	2	2021-05-10T18:57:07.000+0000
6	4/1/2021	b	2	1.2	insert	2	2021-05-10T18:57:07.000+0000
7	4/1/2021	c	1	3.5	insert	2	2021-05-10T18:57:07.000+0000
8	4/1/2021	c	2	2.6	insert	2	2021-05-10T18:57:07.000+0000

Showing all 8 rows.

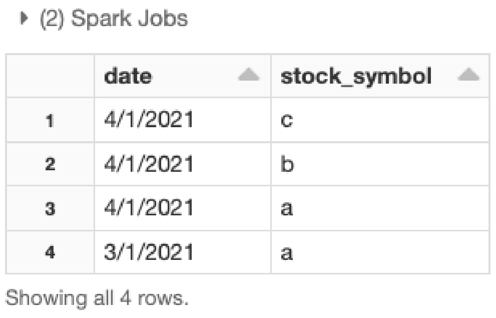
Using CDF row data in a MERGE statement

Aggregate MERGE statements, like the merge into the gold table, can be complex by nature, but the CDF feature makes the coding of these statements simpler and more efficient.



As seen in the above diagram, CDF makes it simple to derive which rows have changed, as it only performs the needed aggregation on the data that has changed or is new using *table_changes* operation. Below, you can see how to use the changed data to determine which dates and stock symbols have changed.

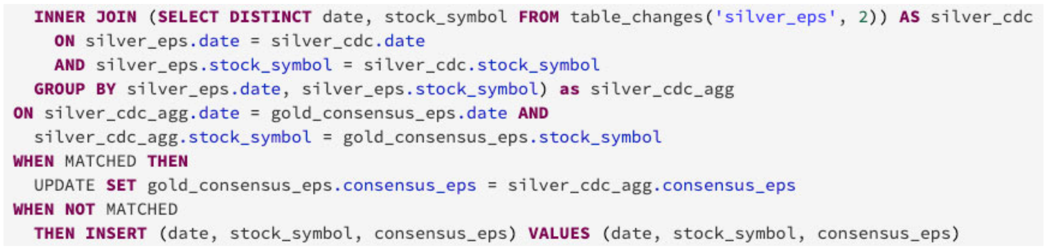
```sql
-- We only want to update for the dates and stocks that have changed
-- so in the SQL below we do a SELECT DISTINCT FROM table_changes...
SELECT DISTINCT date, stock_symbol FROM table_changes('silver_eps', 2)
```


(2) Spark Jobs

	date	stock_symbol
1	4/1/2021	c
2	4/1/2021	b
3	4/1/2021	a
4	3/1/2021	a

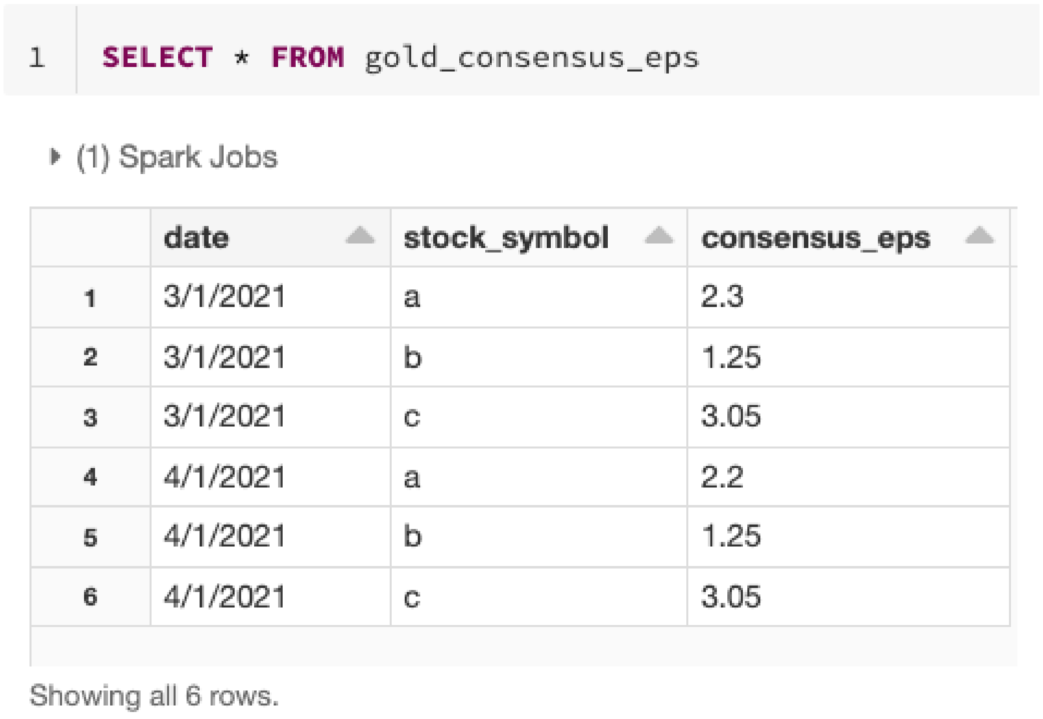
Showing all 4 rows.

As shown below, you can use the changed data from the silver table to aggregate only the data on the rows that need to be updated or inserted into the gold table. To do this, use **INNER JOIN** on the *table_changes('**table_name**','**version**')*



```
  INNER JOIN (SELECT DISTINCT date, stock_symbol FROM table_changes('silver_eps', 2)) AS silver_cdc
    ON silver_eps.date = silver_cdc.date
    AND silver_eps.stock_symbol = silver_cdc.stock_symbol
  GROUP BY silver_eps.date, silver_eps.stock_symbol) as silver_cdc_agg
ON silver_cdc_agg.date = gold_consensus_eps.date AND
  silver_cdc_agg.stock_symbol = gold_consensus_eps.stock_symbol
WHEN MATCHED THEN
  UPDATE SET gold_consensus_eps.consensus_eps = silver_cdc_agg.consensus_eps
WHEN NOT MATCHED
  THEN INSERT (date, stock_symbol, consensus_eps) VALUES (date, stock_symbol, consensus_eps)
```

The end result is a clear and concise version of a gold table that can incrementally change over time!



```
SELECT * FROM gold_consensus_eps
```

(1) Spark Jobs

	date	stock_symbol	consensus_eps
1	3/1/2021	a	2.3
2	3/1/2021	b	1.25
3	3/1/2021	c	3.05
4	4/1/2021	a	2.2
5	4/1/2021	b	1.25
6	4/1/2021	c	3.05

Showing all 6 rows.

Typical use cases

Here are some common use cases and benefits of the new CDF feature:

Silver & gold tables

Improve Delta performance by processing only changes following initial MERGE comparison to accelerate and simplify ETL/ELT operations.

Create up-to-date, aggregated views of information for use in BI and analytics without having to reprocess the full underlying tables, instead updating only where changes have come through.

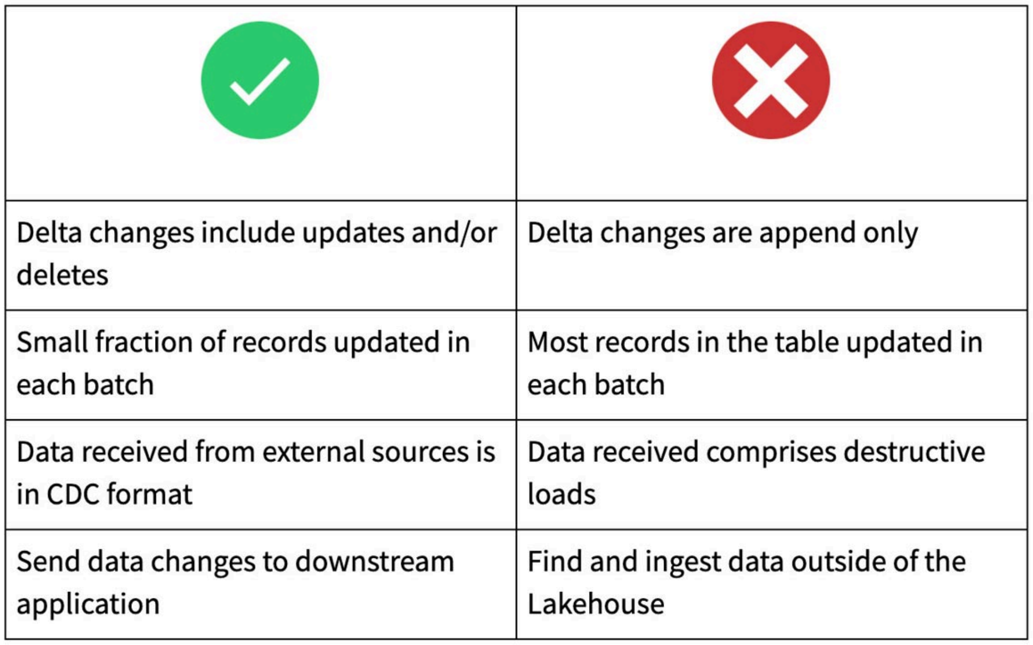
Transmit changes

Send Change Data Feed to downstream systems such as Kafka or RDBMS that can use it to incrementally process in later stages of data pipelines.

Audit trail table

Capturing Change Data Feed outputs as a Delta table provides perpetual storage and efficient query capability to see all changes over time, including when deletes occur and what updates were made.

When to use Change Data Feed



✓	✗
Delta changes include updates and/or deletes	Delta changes are append only
Small fraction of records updated in each batch	Most records in the table updated in each batch
Data received from external sources is in CDC format	Data received comprises destructive loads
Send data changes to downstream application	Find and ingest data outside of the Lakehouse

Conclusion

impossible at scale until Delta Lake was created. Now we are making it simpler and more efficient with the exciting Change Data Feed (CDF) feature!

Try this notebook in Databricks

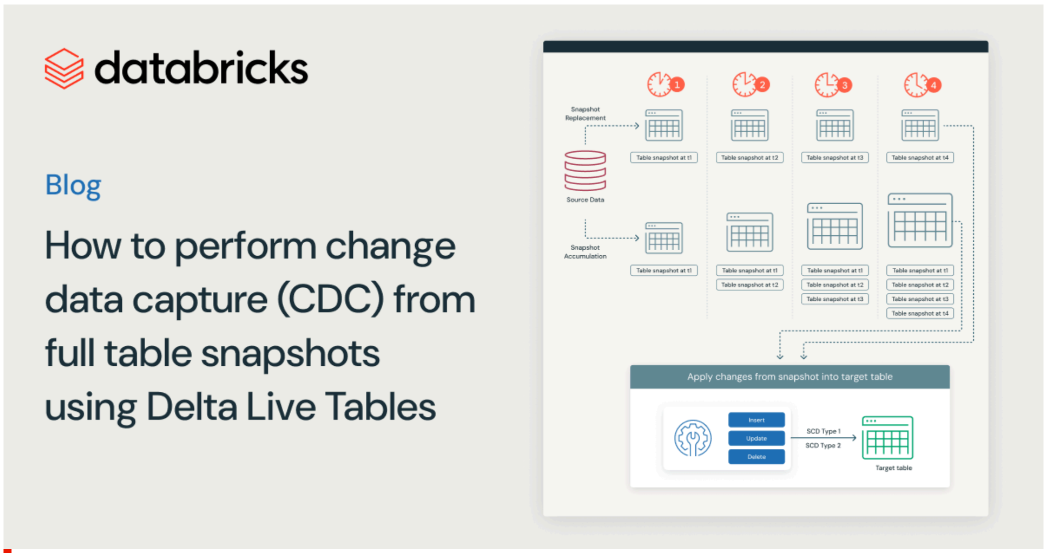
What's next?

Explore more from the authors

Getting Started With Ingestion into Delta Lake

How Incremental ETL Makes Life Simpler With Data Lakes

10 Powerful Features to Simplify Semi-structured Data Management in the Databricks Lakehouse


databricks
Blog
How to perform change data capture (CDC) from full table snapshots using Delta Live Tables
Snapshot Replacement
Source Data
Snapshot Accumulation
Apply changes from snapshot into target table
Insert
Update
Delete
SCD Type 1
SCD Type 2
Target table

DATA ENGINEERING

AUGUST 26, 2024 / 15 MIN READ

How to perform change data capture (CDC) from full table snapshots using Delta Live Tables

SOLUTION ACCELERATORS

SEPTEMBER 4, 2024 / 8 MIN READ

Training Highly Scalable Deep Recommender Systems on Databricks (Part 1)

160 Spear Street, 15th Floor
San Francisco, CA 94105
1-866-330-0121

See Careers
at Databricks