



Change Data Capture With Flink SQL and Debezium

Marta Paes (@morsapaes)
Developer Advocate

About Ververica



Original Creators of
Apache Flink®



Enterprise Stream Processing
With Ververica Platform



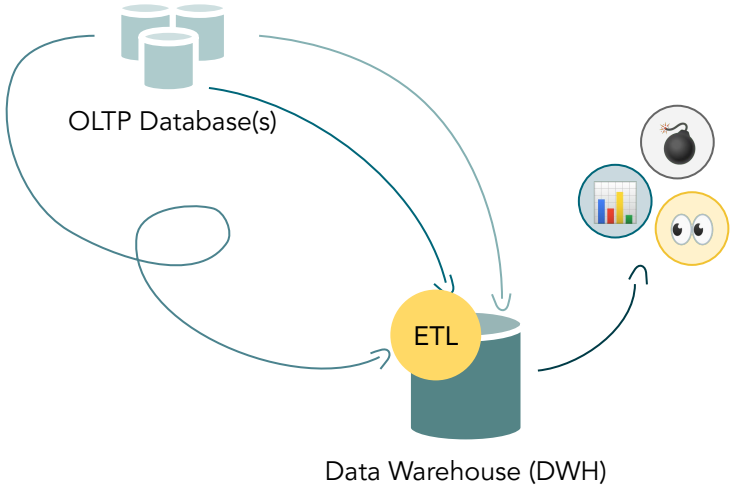
Part of
Alibaba Group



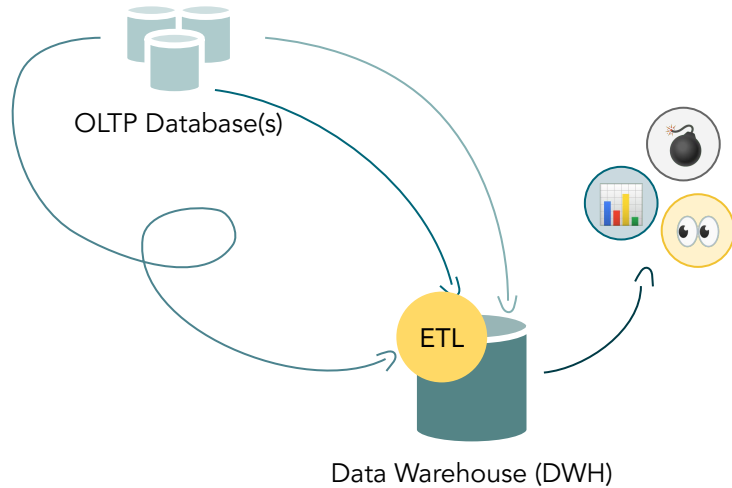
What's Wrong?



The data in your DB is not dead...



The data in your DB is not dead...

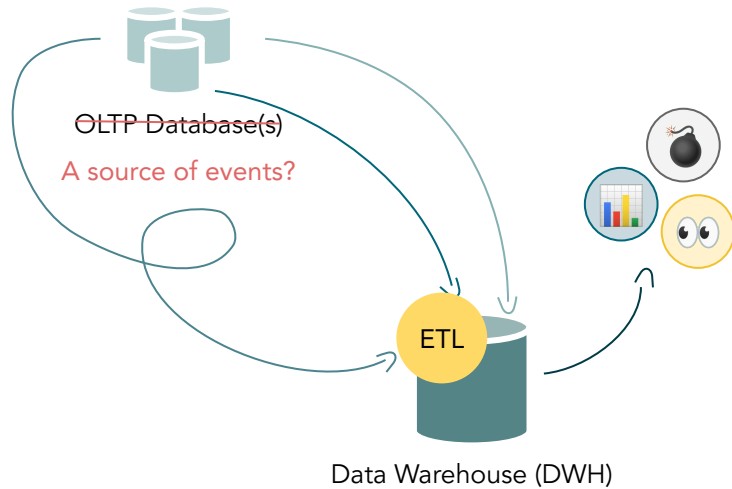


In the end:

- Most source data is continuously produced
- Most logic is not changing that frequently



The data in your DB is not dead...



In the end:

- Most source data is continuously produced
- Most logic is not changing that frequently



- Why are we looking at yesterday's data?
- Why are we not distributing the workload?
- Why are we letting the data go "stale"?

...but your integrations might be killing its value (and also some DBAs).

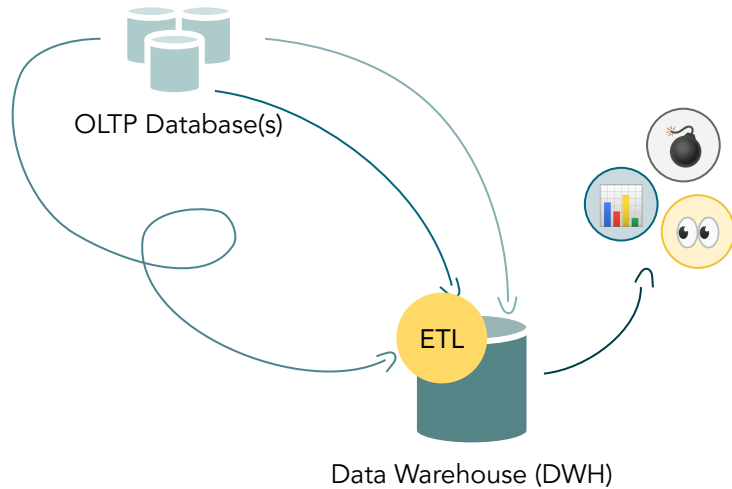


Can't wait to scan a production database for changes using a 100-line query with 1000 business logic conditions.

— No one



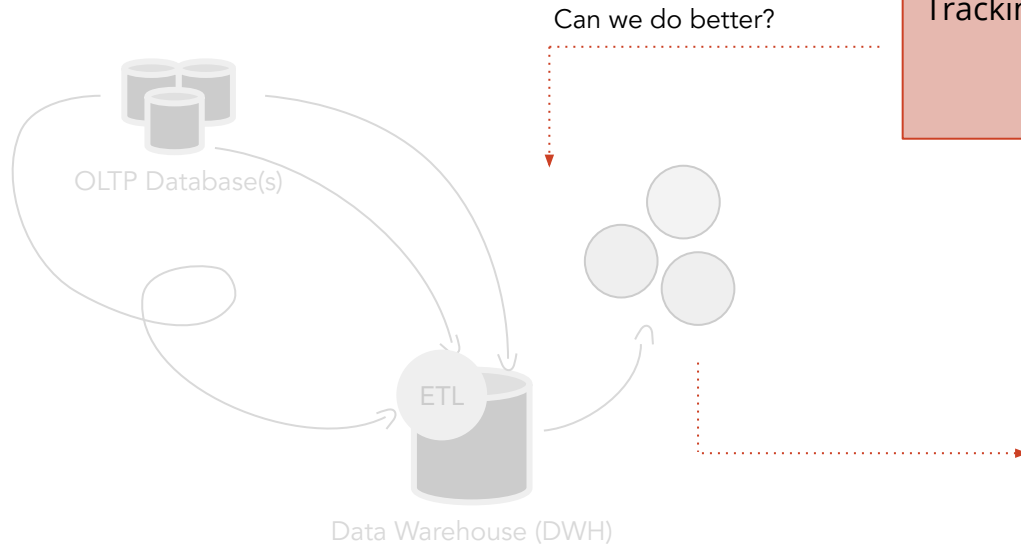
Change Data Capture (CDC)



Tracking and propagating data changes in a database to downstream consumers.



Change Data Capture (CDC)



Tracking and propagating data changes in a database to downstream consumers.



Gunnar "Not Home Alone" Morling 
@gunnarmorling

Change data capture is one giant enabler; it lets you

- * replicate data
- * feed search indexes
- * update caches
- * run streaming queries
- * sync data between microservices
- * maintain denormalized views
- * create audit logs and so much more.

Ultimately, it's liberation for your data.

1:46 PM · Apr 30, 2019 · [Twitter for Android](#)



Not all CDC is created equal

Query-based CDC

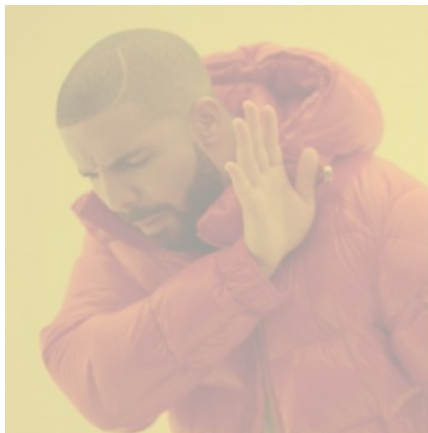


- ✗ Some data changes might get lost
- ✗ DELETE operations are not captured
- ✗ Trade-off: frequency vs. load on source DBs
- ✗ Can't propagate schema changes



Not all CDC is created equal

Query-based CDC

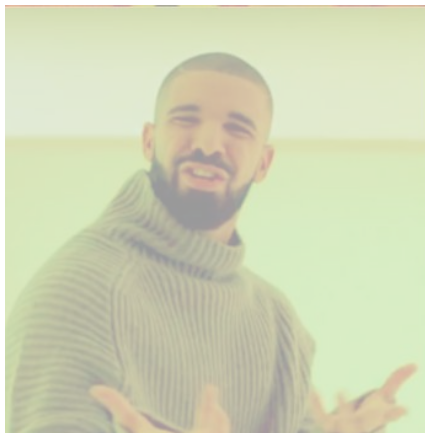
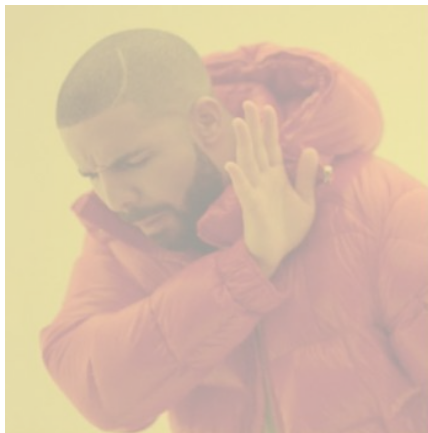


What if we tapped into the transaction log?



Not all CDC is created equal

Query-based CDC



Log-based CDC



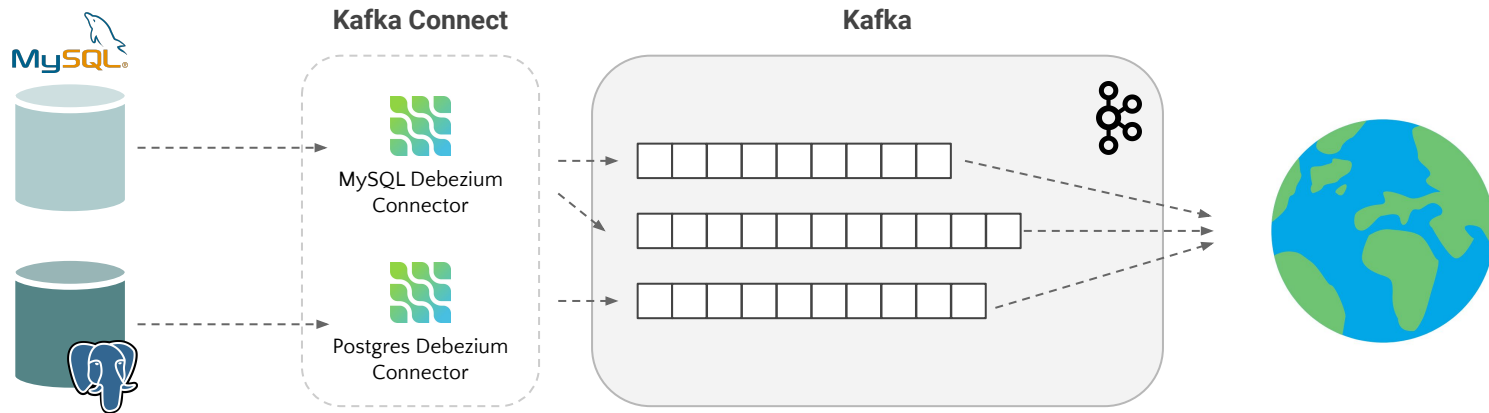
- ✓ All data changes are captured
- ✓ More context on the actual changes
- ✓ Low propagation delay (i.e. near real-time)
- ✓ Minimal impact on the source DBs



Debezium

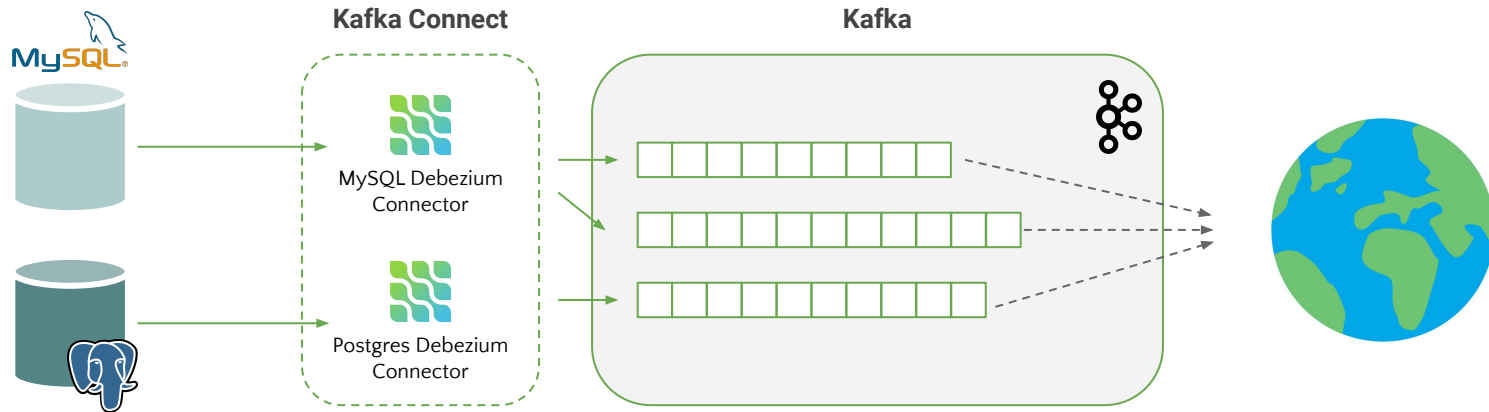
Debezium is an open source distributed platform for **log-based CDC**.

- Canonical format for change events → Different sources, same output
- Support for most common data sources (MySQL, Postgres, MongoDB, ...)



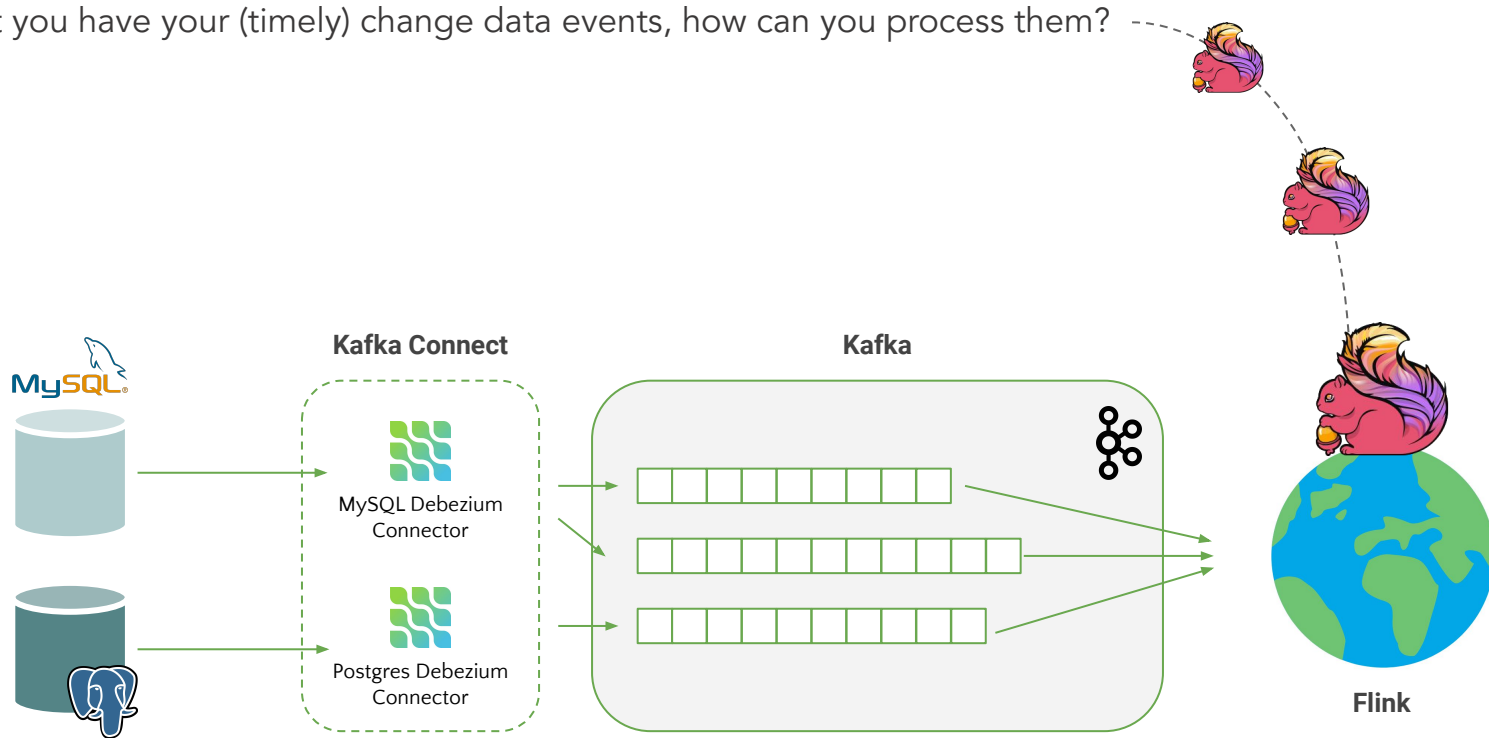
Change Data Captured. Now what?

Now that you have your (timely) change data events, how can you process them?



Change Data Captured. Now what?

Now that you have your (timely) change data events, how can you process them?



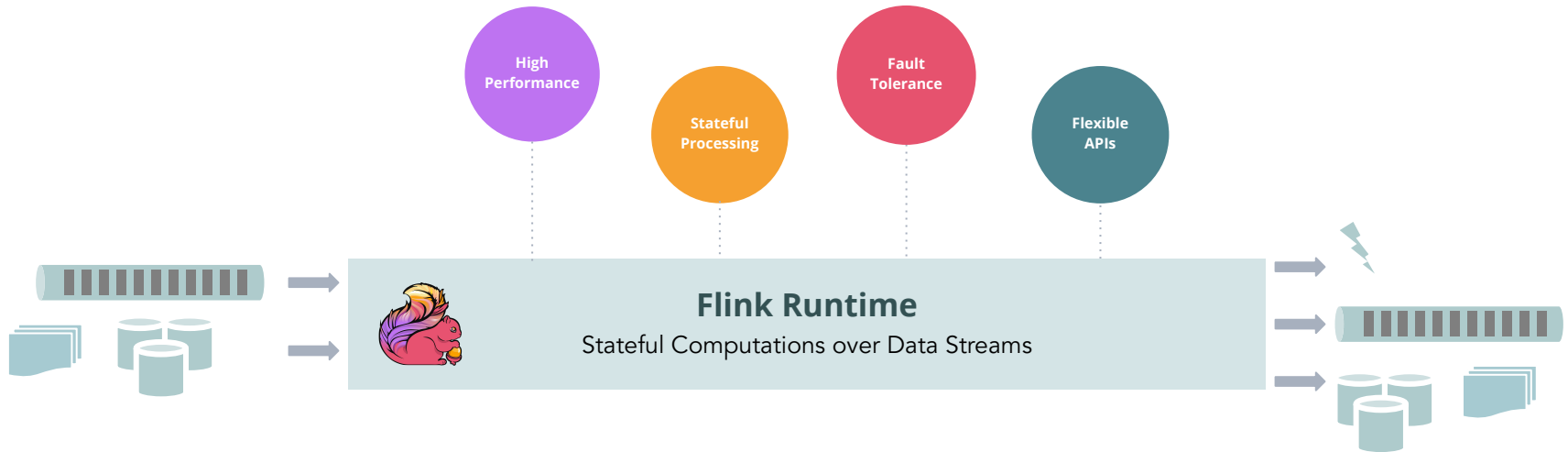
What is Apache Flink?

Flink is an **open source** framework and distributed engine for **stateful stream processing**.



What is Apache Flink?

Flink is an **open source** framework and distributed engine for **stateful stream processing**.



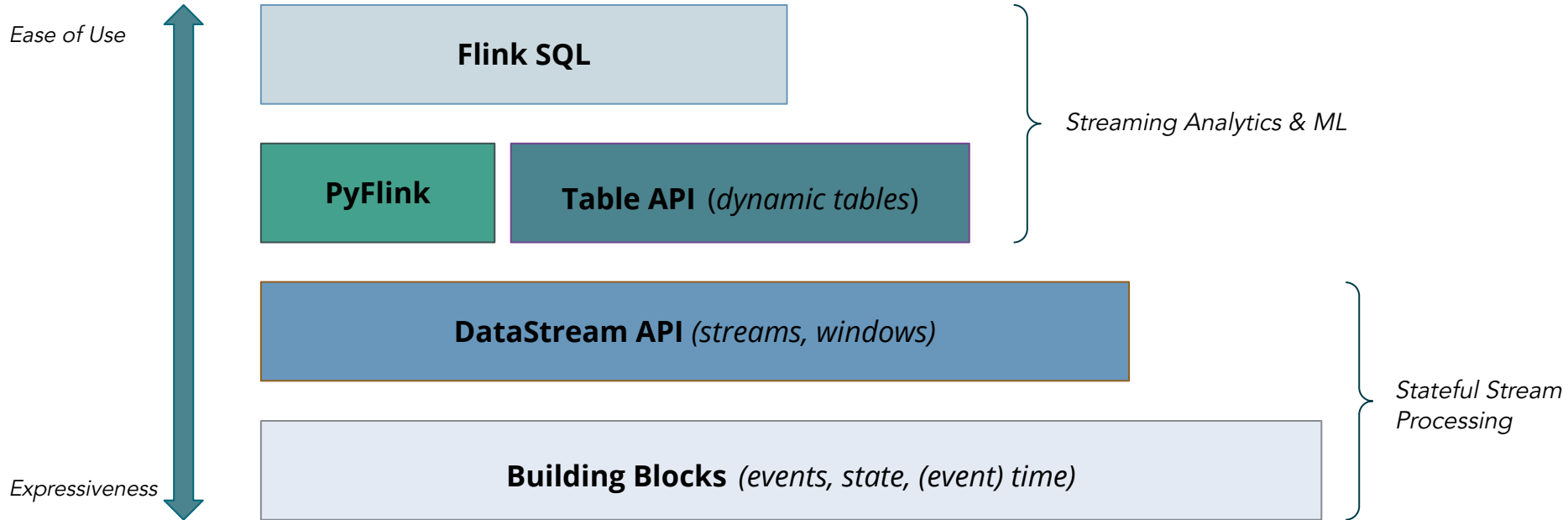
What is Apache Flink?

This gives you a robust foundation for a wide range of use cases:



The Flink API Stack

Flink has layered APIs with different tradeoffs for **expressiveness** and **ease of use**. You can mix and match all the APIs!



The Flink API Stack

For some use cases, you need Flink's full workhorse power.

Ease of Use



Flink SQL

PyFlink

Table API (*dynamic tables*)

DataStream API (*streams, windows*)

Building Blocks (*events, state, (event) time*)

- Explicit control over core primitives (events, state, time)
- Complex computations and customization
- Maximize **performance** and **reliability**



The Flink API Stack

But for a lot of others, you don't.

Ease of Use



Flink SQL

PyFlink

Table API (*dynamic tables*)

DataStream API (*streams, windows*)

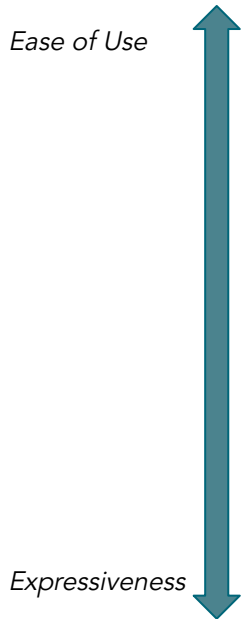
Building Blocks (*events, state, (event) time*)

- Focus on logic, not implementation
- Mixed workloads (batch and streaming)
- Maximize **developer speed** and **autonomy**



The Flink API Stack

But for a lot of others, you don't.



Flink SQL

PyFlink

Table API (*dynamic tables*)

DataStream API (*streams, windows*)

Building Blocks (*events, state, (event) time*)

- Focus on logic, not implementation
- Mixed workloads (batch and streaming)
- Maximize **developer speed** and **autonomy**



Flink SQL

“Everyone knows SQL, right?”

```
SELECT user_id, COUNT(url) AS cnt
FROM clicks
GROUP BY user_id;
```

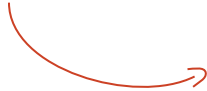
 This is standard SQL (ANSI SQL)



Flink SQL

“Everyone knows SQL, right?”

```
SELECT user_id, COUNT(url) AS cnt  
FROM clicks  
GROUP BY user_id;
```

 This is ~~standard SQL (ANSI SQL)~~
also Flink SQL



A Streaming SQL Engine

Ingest all changes as they happen

Continuously update the result

user	cTime	url
Mary	12:00:00	https://...
Bob	12:00:00	https://...
Mary	12:00:02	https://...
Liz	12:00:03	https://...

```
SELECT user_id,  
       COUNT(url) AS cnt  
FROM clicks  
GROUP BY user_id;
```

user	cnt
Mary	2
Bob	1
Liz	1



Flink SQL in a Nutshell

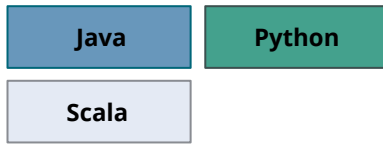
- SQL syntax and semantics (i.e. not a “SQL-flavor”)
- Unified APIs for batch and streaming
- Support for advanced operations (e.g. temporal joins, pattern matching/CEP)

Execution

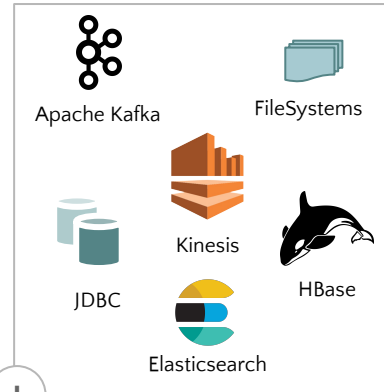


TPC-DS Coverage

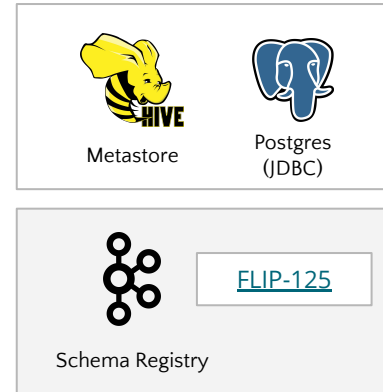
UDF Support



Native Connectors



Data Catalogs

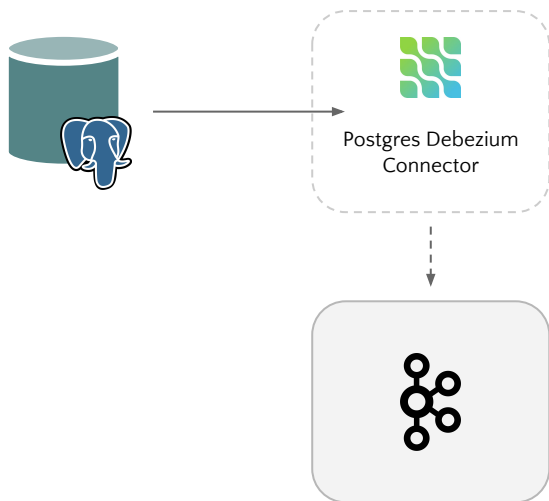


Notebooks



Flink SQL + CDC

- Available from **Flink 1.11** * (released Jul. 2020)
- Initial implementation:
 - **JSON-encoded** changelogs;
 - **Kafka** as a changelog source.



```
SELECT user_id, COUNT(url) AS cnt  
FROM clicks  
GROUP BY user_id;
```

...

```
CREATE TABLE clicks (  
...  
) WITH (  
  'connector'='kafka',  
  'format'='debezium-json',  
  'debezium-json.schema-include'='false'  
);
```





(Un)Demo



What are we doing?



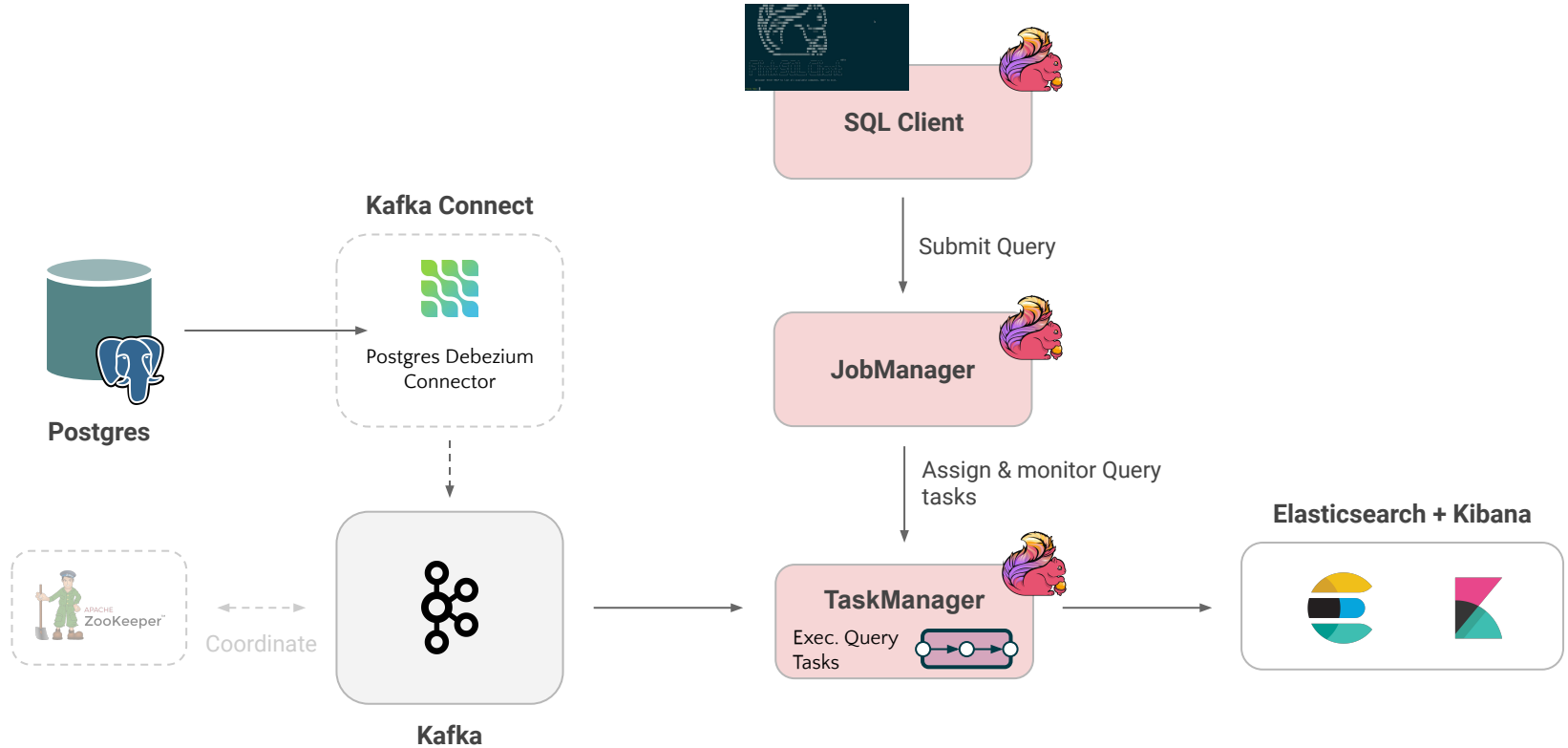
A Cassowary aka the world's most dangerous bird.

Processing (fake) insurance claim data related to animal attacks in Australia.

- Get Debezium up and running with Kafka
- Register a Postgres catalog to access external table metadata
- Create a changelog source to consume Debezium CDC data from Kafka
- See CDC in action!
- Maintain a Materialized View (MV) in Elasticsearch
- Visualize the results in Kibana



The Demo Environment



1. Start the Postgres client to check the source tables and run some DML statements later:

```
1. Postgres (bash)
Postgres (bash)
morsapaes@local:flink-sql-CDC (main)$
```

```
SELECT * FROM information_schema.tables WHERE table_schema='claims';
```


Tables:

- `claims.members`
- `claims.accident_claims` (1000 records)



2. Register the Debezium Postgres connector in Kafka Connect.

```
curl -i -X POST -H "Accept:application/json" -H "Content-Type:application/json"  
http://localhost:8083/connectors/ -d @register-postgres.json
```



A terminal window titled "2. Kafka (bash)" is shown. The terminal prompt is "morsapaes@local:flink-sql-CDC (main)\$". The terminal is currently empty, indicating that the command from the previous block has not yet been executed.



2. Register the Debezium Postgres connector in Kafka Connect.

```
{
  "name": "claims-connector",
  "config": {
    "connector.class": "io.debezium.connector.postgresql.PostgresConnector",
    "tasks.max": "1",
    "database.hostname": "postgres",
    "database.port": "5432",
    "database.user": "postgres",
    "database.password": "postgres",
    "database.dbname" : "postgres",
    "database.server.name": "pg_claims",
    "table.whitelist": "claims.accident_claims",
    "value.converter": "org.apache.kafka.connect.json.JsonConverter",
    "value.converter.schemas.enable": false,
    "decimal.handling.mode": "double",
    "tombstones.on.delete":false
  }
}
```

Connector configuration

register-postgres.json



3. Check that the snapshot events have been pushed to the `pg_claims.claims.accident_claims` Kafka topic.



```
2. Kafka (bash)
Kafka (bash)
morsapaes@local:flink-sql-CDC (main)$
```



3. Check that the snapshot events have been pushed to the pg_claims.claims.accident_claims Kafka topic.

```
2. Kafka (bash)
Kafka (bash)
{"schema":{"type":"struct","fields":[{"type":"int32","optional":false,"field":"claim_id"},"optional":false,"name":"pg_claims.claims.
accident_claims.Key"},"payload":{"claim_id":1000}} {"before":null,"after":{"claim_id":1000,"claim_total":3447.16,"claim_total_re
ceipt":null,"claim_currency":"EUR","member_id":27,"accident_date":"2020-03-31 16:14:04","accident_type":"Collision","accident_detail
":"Cassowary","claim_date":"2020-04-03 14:23:31","claim_status":"IN REVIEW","ts_created":"2020-09-28 04:48:09","ts_updated":"2020-09-2
8 04:48:09"},"source":{"version":"1.2.1.Final","connector":"postgresql","name":"pg_claims","ts_ms":1601317692834,"snapshot":"true","d
b":"postgres","schema":"claims","table":"accident_claims","txId":2076,"lsn":25264904,"xmin":null},"op":"r","ts_ms":1601317692834,"tra
nsaction":null}
^CProcessed a total of 1000 messages This checks out!
```

Debezium change events (JSON):

- Event key
- Event value: before, after, op, ts_ms



4. Is Debezium working?

```
1. Postgres (docker)
Postgres (docker)
table_catalog | table_schema | table_name | table_type | self_referencing_column_name | reference_generation | user_defined_type_catalog | user_defined_type_schema | user_defined_type_name | is_insertable_into | is_typed | commit_action
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
postgres | claims | members | BASE TABLE | | | | | | | | |
postgres | claims | accident_claims | BASE TABLE | | | | | | | | |
(2 rows)
postgres=# █
```

INSERT

```
docker
28 04:48:09}], "source": {"version": "1.2.1.Final", "connector": "postgresql", "name": "pg_claims", "ts_ms": 1601317692834, "snapshot": "true", "db": "postgres", "schema": "claims", "table": "accident_claims", "txId": 2076, "lsn": 25264904, "xmin": null}, "op": "r", "ts_ms": 1601317692834, "transaction": null}
{"schema": {"type": "struct", "fields": [{"type": "int32", "optional": false, "field": "claim_id"}]}, "optional": false, "name": "pg_claims.claims.accident_claims.Key", "payload": {"claim_id": 1000} {"before": null, "after": {"claim_id": 1000, "claim_total": 3447.16, "claim_total_receipt": null, "claim_currency": "EUR", "member_id": 27, "accident_date": "2020-03-31 16:14:04", "accident_type": "Collision", "accident_detail": "Cassowary", "claim_date": "2020-04-03 14:23:31", "claim_status": "IN REVIEW", "ts_created": "2020-09-28 04:48:09", "ts_updated": "2020-09-28 04:48:09"}, "source": {"version": "1.2.1.Final", "connector": "postgresql", "name": "pg_claims", "ts_ms": 1601317692834, "snapshot": "true", "db": "postgres", "schema": "claims", "table": "accident_claims", "txId": 2076, "lsn": 25264904, "xmin": null}, "op": "r", "ts_ms": 1601317692834, "transaction": null}
█
```



5. Start the Flink SQL Client and register a Postgres catalog to access the metadata of existing tables over JDBC.

SQL Client



```
morsapaes@local:flink-sql-cdc (main)$
```

Catalog DDL

```
CREATE CATALOG postgres WITH (  
  'type'='jdbc',  
  'property-version'='1',  
  'base-url'='jdbc:postgresql://postgres:5432/',  
  'default-database'='postgres',  
  'username'='postgres',  
  'password'='postgres'  
);
```



6. Create a changelog table to consume the change events from the `pg_claims.claims.accident_claims` topic.

Source Table DDL

```
CREATE TABLE accident_claims
WITH (
  'connector' = 'kafka',
  'topic' = 'pg_claims.claims.accident_claims',
  'properties.bootstrap.servers' = 'kafka:9092',
  'properties.group.id' = 'test-consumer-group',
  'format' = 'debezium-json',
  'scan.startup.mode' = 'earliest-offset'
)
LIKE `claims.accident_claims` (
  EXCLUDING OPTIONS);
```

Define the source connector (Kafka)

Derive schema from the original table (Postgres)



7. Is Debezium+Flink working?

2. Flink SQL (docker)

```
Postgres (docker)
(2 rows)
postgres=# INSERT INTO accident_claims (claim_total, claim_total_receipt, claim_currency, member_id, accident_date, accident_type, ac
cident_detail, claim_date, claim_status) VALUES (500, 'PharetraMagnaVestibulum.tiff', 'AUD', 321, '2020-08-01 06:43:03', 'Collision',
'Blue Ringed Octopus', '2020-08-10 09:39:31', 'INITIAL');
INSERT 0 1
postgres=# UPDATE accident_claims SET claim_total_receipt = 'CorrectReceipt.pdf' WHERE claim_id = 1001;
UPDATE 1
postgres=# DELETE FROM accident_claims WHERE claim_id = 1001;
DELETE 1
postgres=#
```

Flink SQL (docker)

SQL Query Result (Table)
Refresh: 1 s Page: Last of 167 Updated: 14:53:41.139

claim_id	claim_total	claim_total_receipt	claim_currency	member_id
997	8181.15	(NULL)	EUR	17
998	6149.35	(NULL)	AUD	73
999	3393.16	(NULL)	EUR	385
1000	3447.16	(NULL)	EUR	27

Quit Refresh Goto Page Last Page Next Page Prev Page Open Row



8. Continuously write results to Elasticsearch and visualize the changes using Kibana.

Sink Table DDL

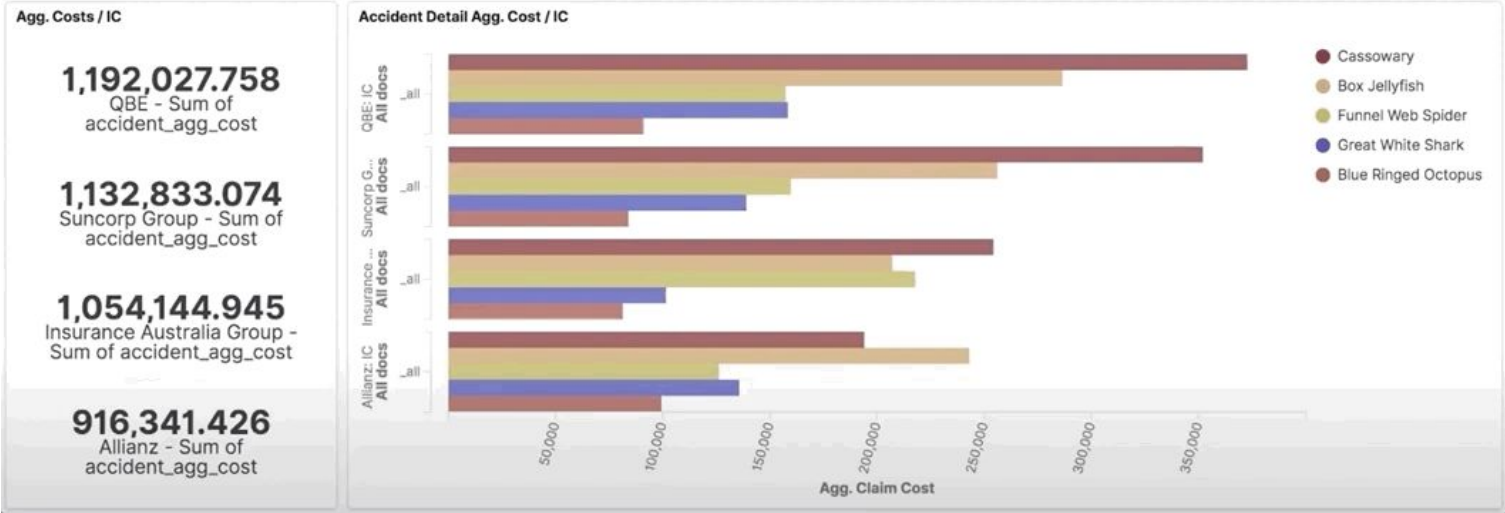
```
CREATE TABLE agg_insurance_costs (  
  es_key STRING PRIMARY KEY NOT ENFORCED,  
  insurance_company STRING,  
  accident_detail STRING,  
  accident_agg_cost DOUBLE  
) WITH (  
  'connector' = 'elasticsearch-7',  
  'hosts' = 'http://elasticsearch:9200',  
  'index' = 'agg_insurance_costs'  
);
```

Continuous SQL Query

```
INSERT INTO agg_insurance_costs  
SELECT UPPER(SUBSTRING(m.insurance_company,0,4) || '_' ||  
SUBSTRING (ac.accident_detail,0,4)) es_key,  
  m.insurance_company,  
  ac.accident_detail,  
  SUM(ac.claim_total) member_total  
FROM accident_claims ac  
JOIN members m  
ON ac.member_id = m.id  
WHERE ac.claim_status <> 'DENIED'  
GROUP BY m.insurance_company, ac.accident_detail;
```



8. Continuously write results to Elasticsearch and visualize the changes using Kibana.



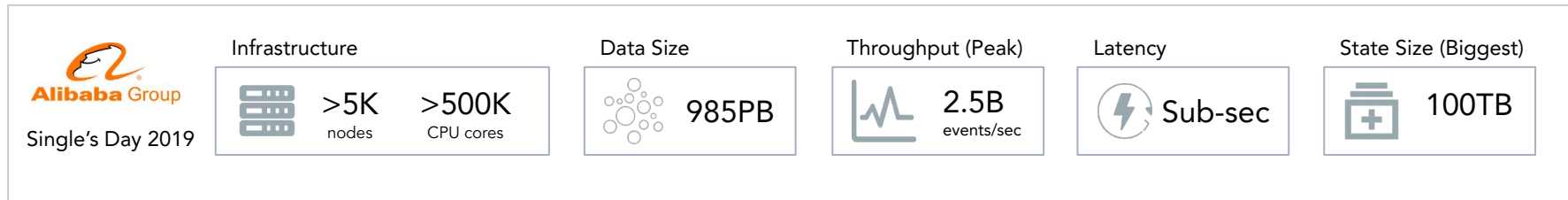


(Un)Demo



To wrap it up...

- Flink SQL is used at massive scale in companies like Alibaba, Uber, Yelp and Lyft.



- Flink SQL is standard, provides unified APIs and has a growing ecosystem of integrations around it.

Upcoming CDC Improvements

- Umbrella ticket ([FLINK-18822](#)):
 - Avro-encoded** changelogs;
 - Temporal joins** with changelog sources;
 - Batch** support.

Check out these open-sourced Flink CDC connectors:
<https://github.com/ververica/flink-cdc-connectors>



Want to learn more about Flink?

#flinkforward

**Join the Apache Flink community virtually at
Flink Forward Global 2020!**

Register for free:

flink-forward.org/global-2020

**FLINK
FORWARD** 
October 19-22, 2020





Thank you, ApacheCon!

Follow me on Twitter: @morsapaes

Learn more about Flink: <https://flink.apache.org/>