A CDC use-case: Designing an Evergreen Cache

Nicolas Fränkel

Me, myself and I

- Developer
- Developer Advocate
- Interested in CDC and data streaming

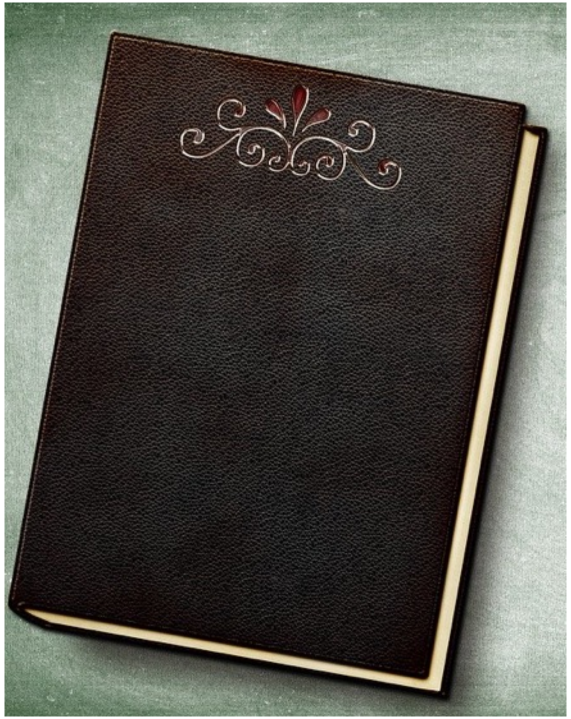
Hazelcast

HAZELCAST IMDG is an **operational, in-memory**, distributed computing platform that manages data using in-memory storage and performs parallel execution for breakthrough application speed and scale.

HAZELCAST JET is the ultra fast, application embeddable, 3rd generation stream processing engine for low latency batch and stream processing.

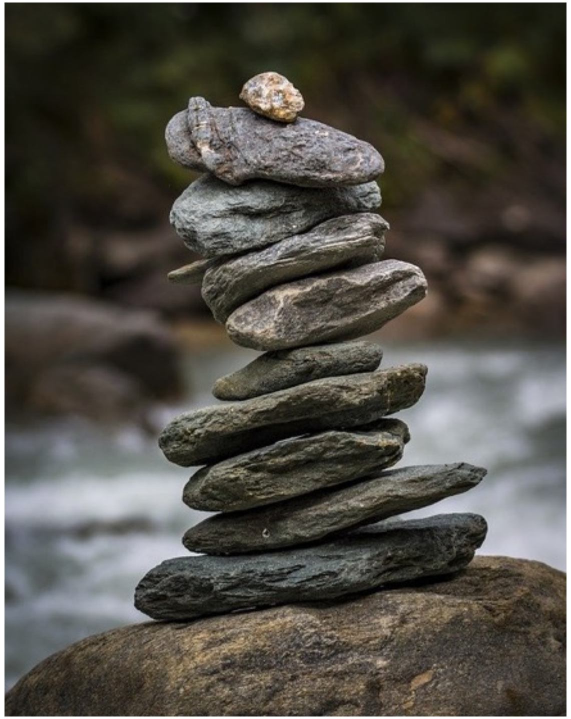
Agenda

1. Why cache?
2. Alternatives to keeping the cache in sync
3. Change-Data-Capture (CDC)
4. Debezium, a CDC implementation
5. Hazelcast Jet + Debezium
6. Demo!

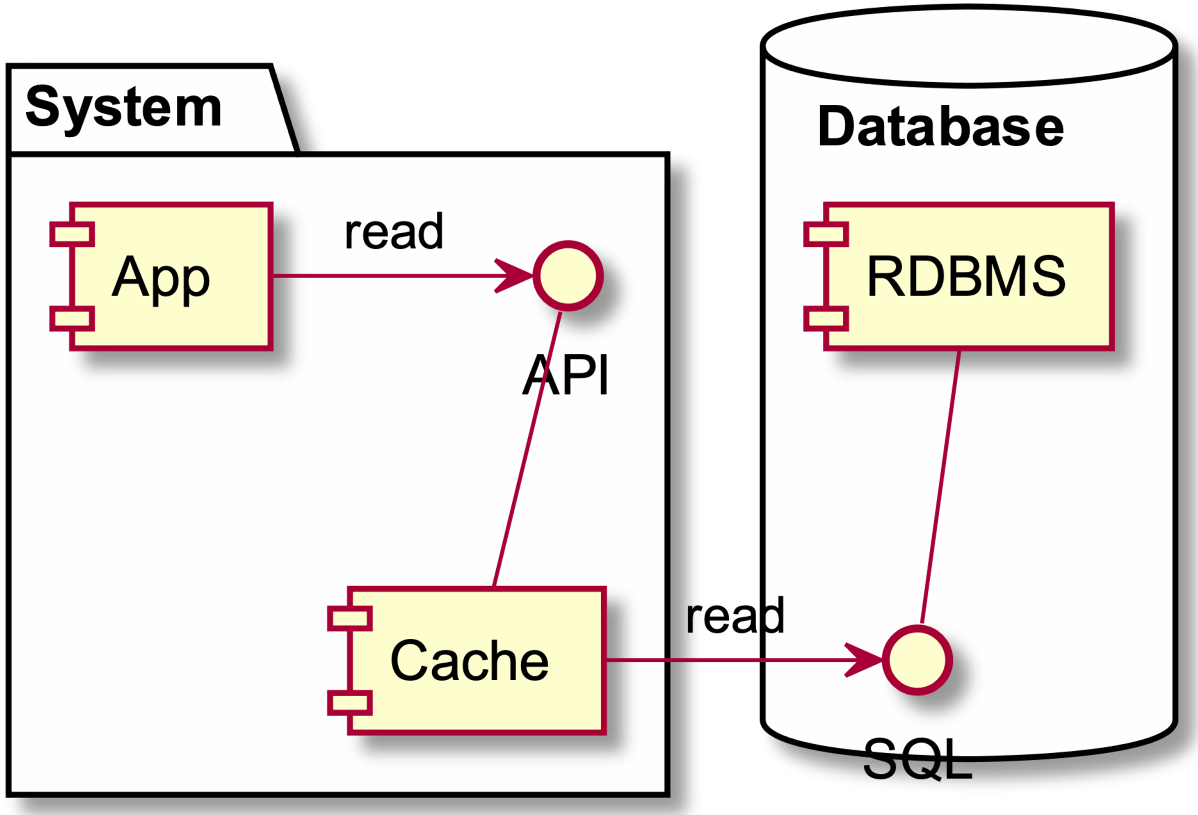


The caching trade-off

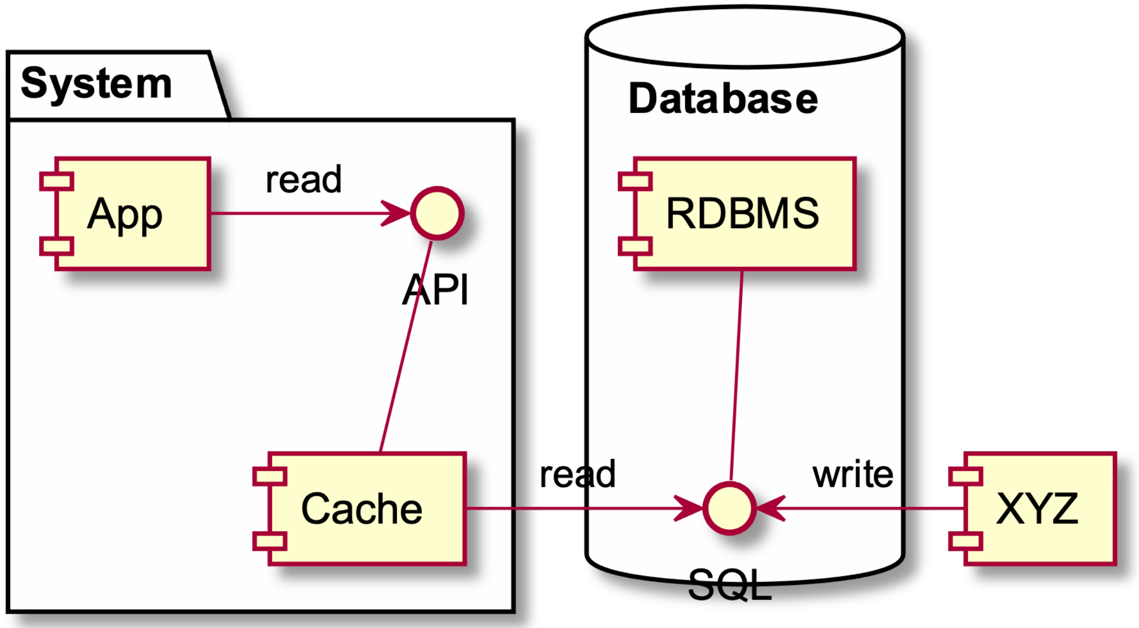
- Improved performance/availability
- Stale data



The initial state



Aye, there's the rub!



- A new component writes to the database
- E.g.: a table holding references needs to be updated every now and then

How to keep the cache in sync with the DB?

Cache invalidation

“There are two hard things in computer science:

1. Naming things
2. Cache invalidation
3. And off-by-one errors”

Cache eviction vs Time-To-Live

- **Cache eviction**: which entities to evict when the cache is full
  - Least Recently Used
  - Least Frequently Used
- **TTL**: how long will an entity be kept in the cache

Choosing the “correct” TTL

- Less frequent than the update frequency
  - Miss updates
- More frequent than the update frequency
  - Waste resources

External process

Same issue regarding the frequency

OIL AND GAS · OIL AND GAS · OIL AND GAS
PARTNERSHIP · PARTNERSHIP

Event-driven for the win!

1. If no writes happen, there's no need to update the cache
2. If a write happens, then the relevant cache item should be updated accordingly

RDMBS triggers

- Not all RDBMS implement triggers
- How to call an external process from the trigger?

The example of MySQL: User-defined function

- Functions must be written in C++
- The OS must support dynamic loading
- Becomes part of the running server
  - Bound by all constraints that apply to writing server code
- Etc.

-- *https://dev.mysql.com/doc/refman/8.0/en/adding-udf.html*

lib_mysqludf_sys

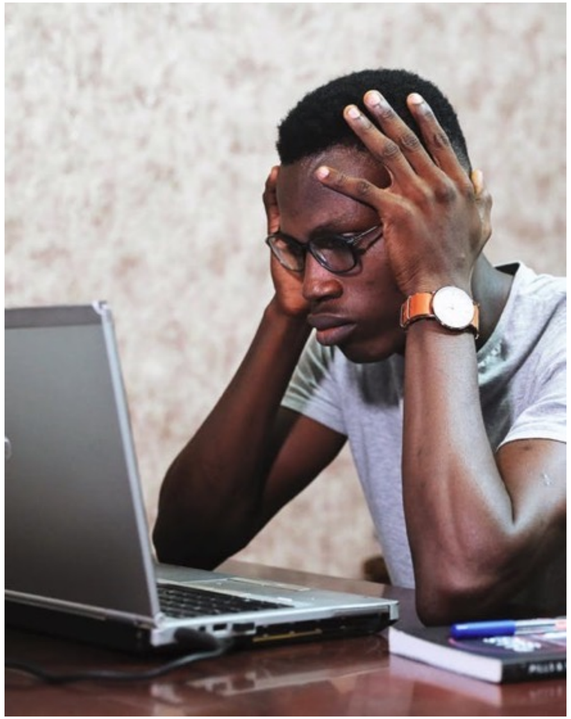
UDF library with functions to interact with the operating system

```
CREATE TRIGGER MyTrigger
AFTER INSERT ON MyTable
FOR EACH ROW
BEGIN
 DECLARE cmd CHAR(255);
 DECLARE result INT(10);
 SET cmd = CONCAT('update_row', '1');
 SET result = sys_exec(cmd);
END;
```

-- https://github.com/mysqludf/lib_mysqludf_sys

Cons

- Implementation-dependent
- Fragile
- Who maintains/debugs it?
- Resource-consuming if done frequently



Change-Data-Capture

"In databases, Change Data Capture is a set of software design patterns used to **determine and track the data that has changed** so that action can be taken using the changed data.

CDC is an approach to data integration that is based on the **identification, capture and delivery of the changes made to enterprise data sources**."

-- *https://en.wikipedia.org/wiki/Change_data_capture*

CDC implementation options

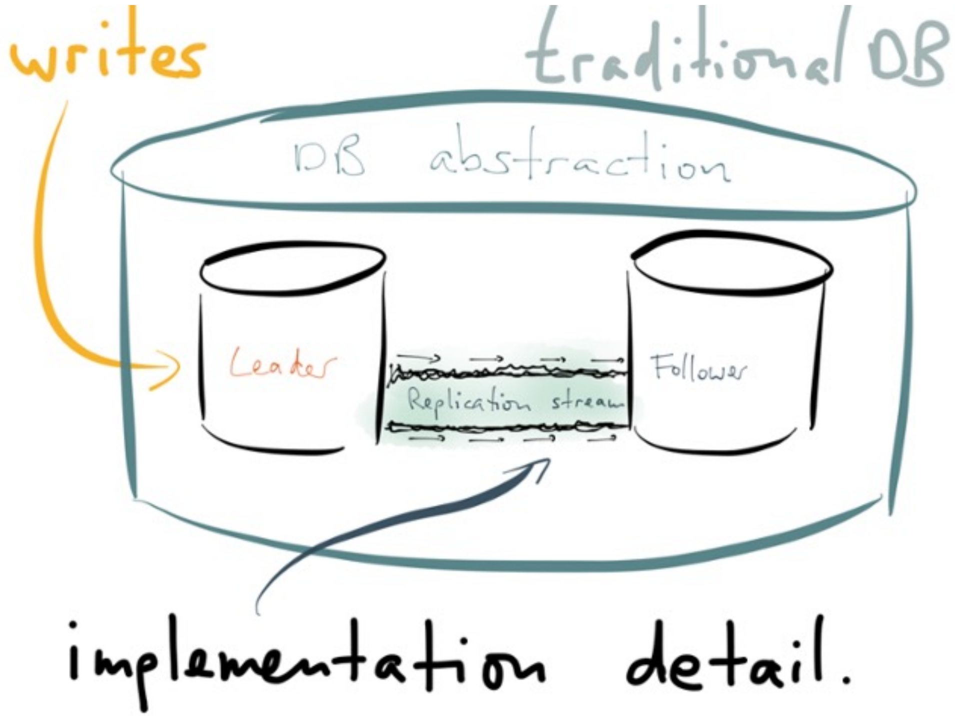
1. Polling + Timestamps on rows
2. Polling + Version numbers on rows
3. Polling + Status indicators on rows
4. *Triggers on tables*
5. **Log scanners**

-- https://en.wikipedia.org/wiki/Change_data_capture

hazelcast®

“Turning the database inside out” - Martin Kleppman

-- https://www.confluent.io/blog/turning-the-database-inside-out-with-apache-samza/



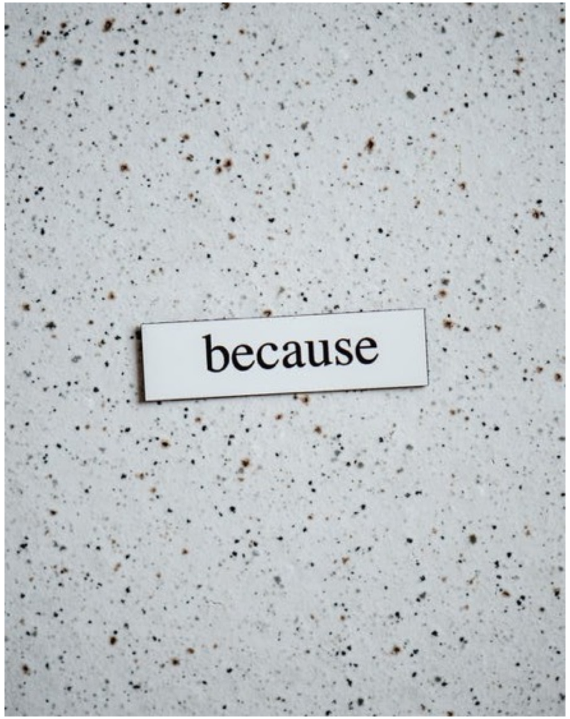
What is a transaction/binary/etc. log?

"The binary log contains 'events' that describe database changes such as table creation operations or changes to table data."

-- *https://dev.mysql.com/doc/refman/8.0/en/binary-log.html*

Reasons for the log

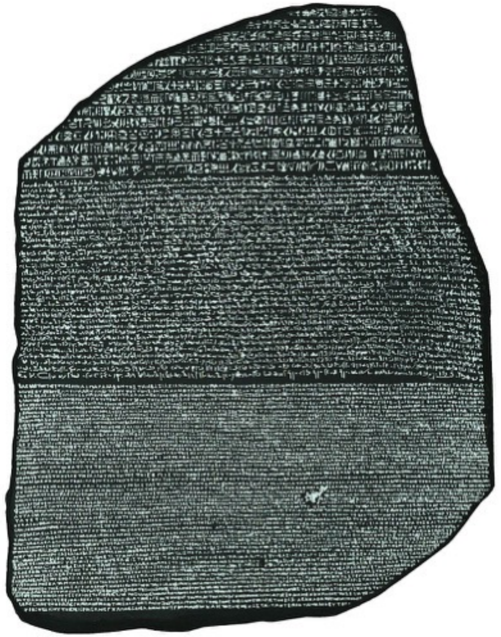
1. Data recovery
2. Replication



What if we “hacked” the log?

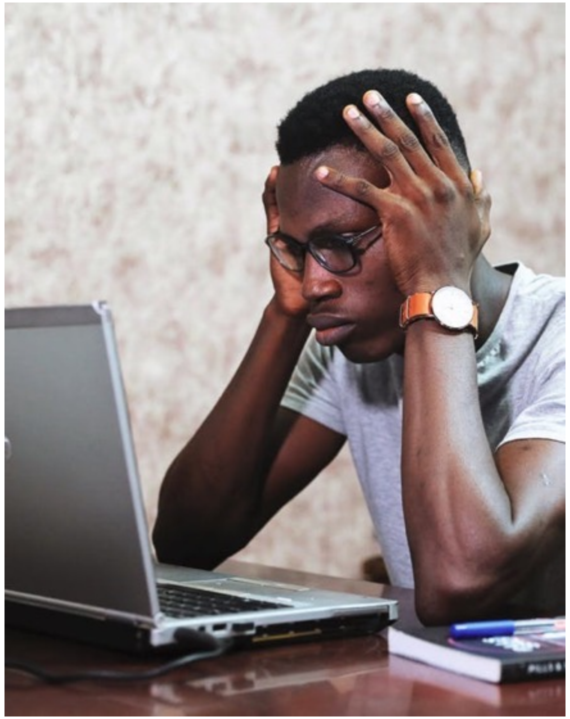
Sample MySQL binlog

```
### UPDATE `test`.`t`
### WHERE
###   @1=1 /* INT meta=0 nullable=0 is_null=0 */
###   @2='apple' /* VARSTRING(20) meta=20 nullable=0 is_null=0 */
###   @3=NULL /* VARSTRING(20) meta=0 nullable=1 is_null=1 */
### SET
###   @1=1 /* INT meta=0 nullable=0 is_null=0 */
###   @2='pear' /* VARSTRING(20) meta=20 nullable=0 is_null=0 */
###   @3='2009:01:01' /* DATE meta=0 nullable=1 is_null=0 */
# at 569
#150112 21:40:14 server id 1  end_log_pos 617 CRC32 0xf134ad89
#Table_map: `test`.`t` mapped to number 251
# at 617
#150112 21:40:14 server id 1  end_log_pos 665 CRC32 0x87047106
#Delete_rows: table id 251 flags: STMT_END_F
```



Kind reminder…

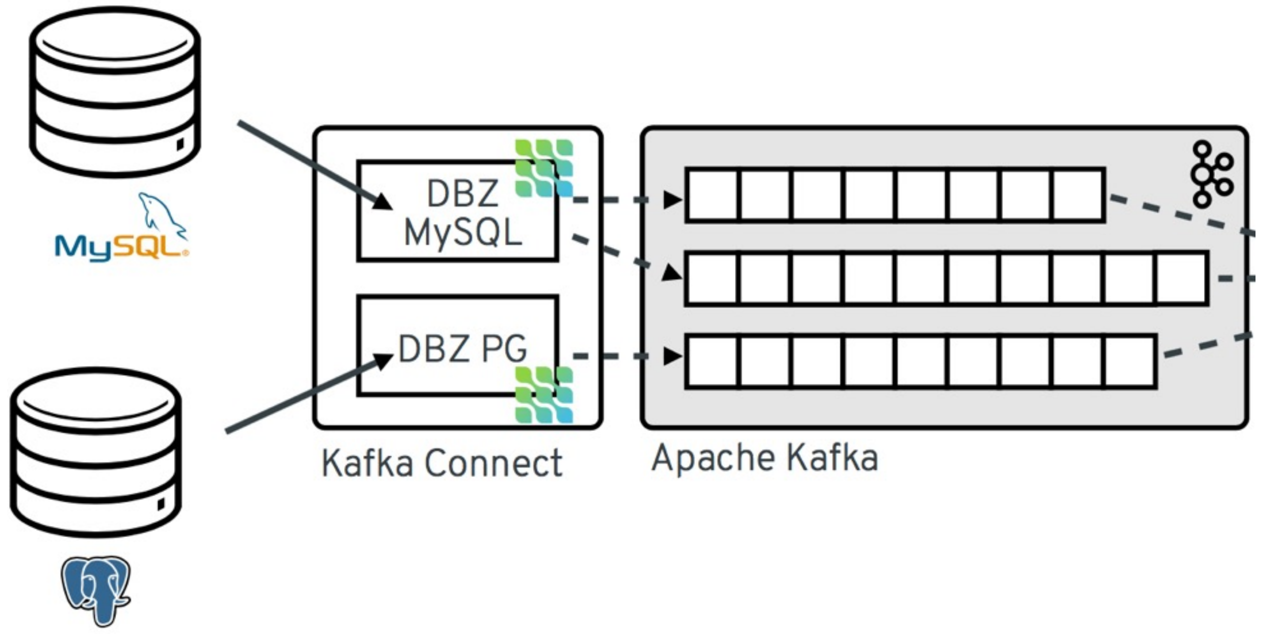
- Implementation-dependent
- Fragile
- **Who maintains/debugs it?**



Debezium to the rescue

- Java-based abstraction layer for CDC
- Provided by Red Hat
  - Apache v2 licensed

Debezium



"Debezium records all row-level changes within each database table in a change event stream"

-- https://debezium.io/

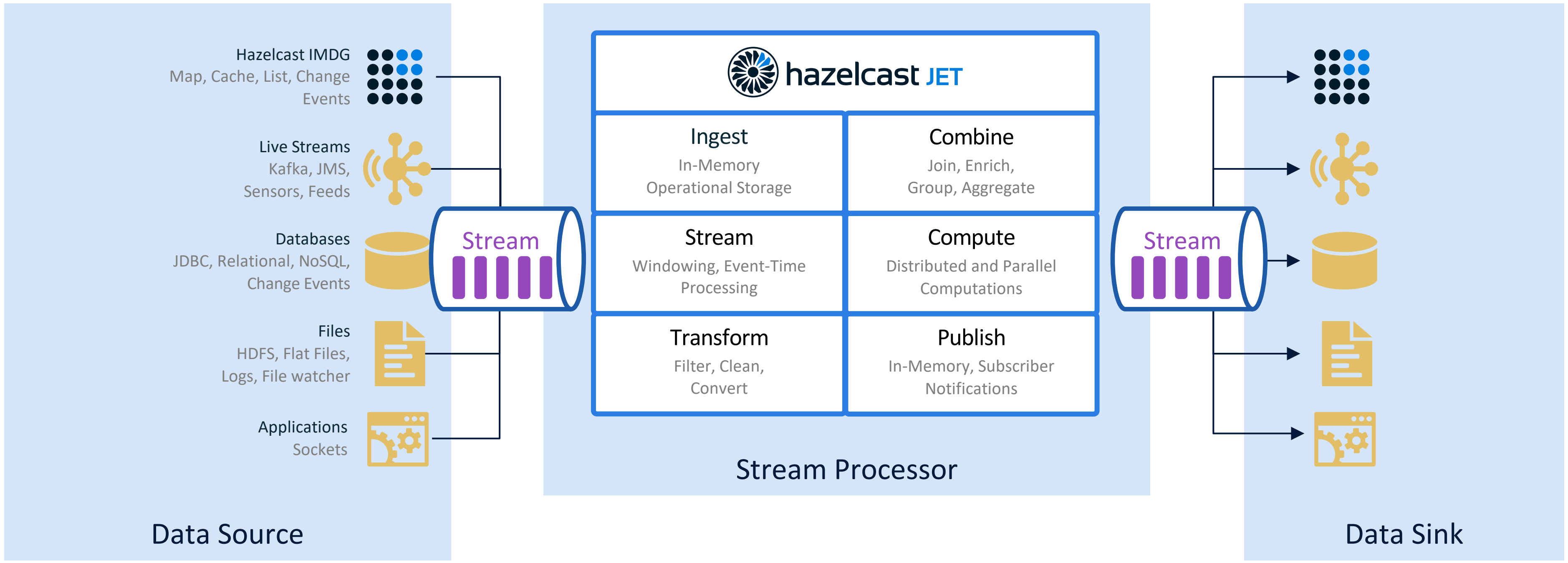
Debezium connector plugins

- Production-ready
  - MongoDB
  - MySQL
  - PostrgreSQL
  - SQL Server
  - DB2 (!)
- Incubating
  - Oracle
  - Cassandra
  - Vitess

Hazelcast Jet

- Stream Processing Engine (SPE)
- Distributed
- In-memory
- Apache v2 licensed
  - (Hazelcast Jet Enterprise offering)

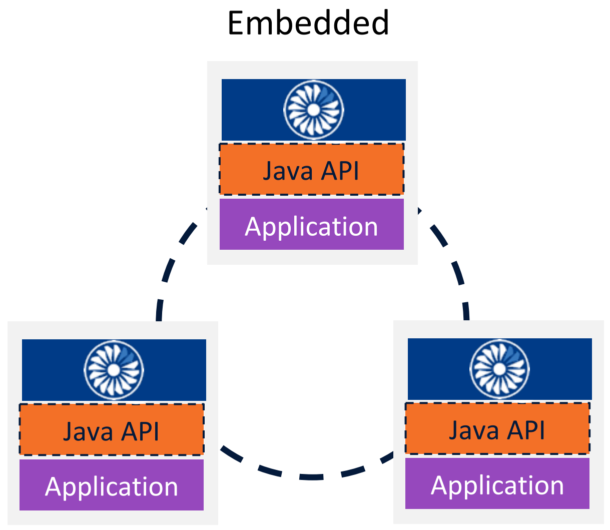
Jet overview


Hazelcast IMDG
Map, Cache, List, Change Events
Live Streams
Kafka, JMS, Sensors, Feeds
Databases
JDBC, Relational, NoSQL, Change Events
Files
HDFS, Flat Files, Logs, File watcher
Applications
Sockets
Data Source
Stream
hazelcast JET
Ingest
In-Memory Operational Storage
Combine
Join, Enrich, Group, Aggregate
Stream
Windowing, Event-Time Processing
Compute
Distributed and Parallel Computations
Transform
Filter, Clean, Convert
Publish
In-Memory, Subscriber Notifications
Stream Processor
Stream
Data Sink

@nicolas_frankel

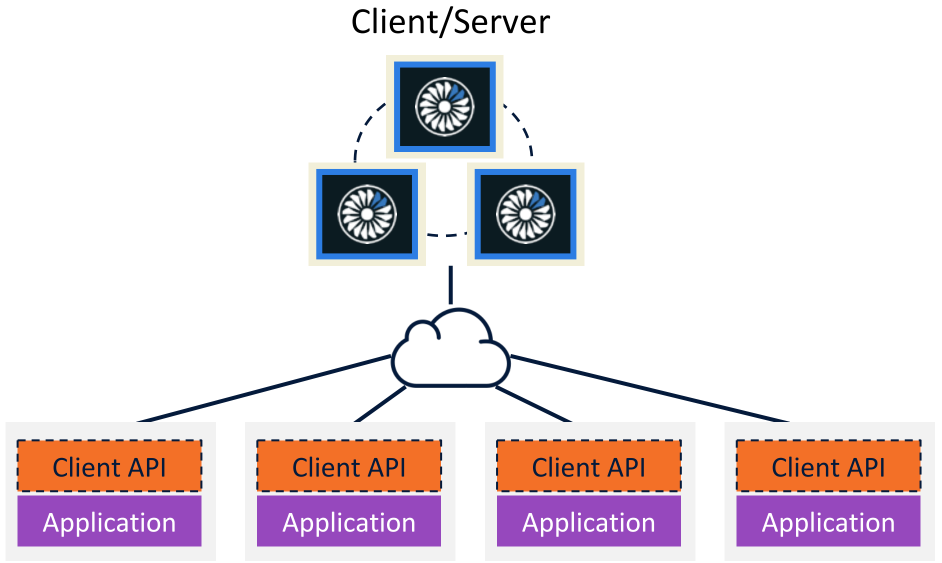
hazelcast®

Deployment modes



```
// Create new cluster member

JetInstance jet = Jet.newJetInstance();
```



```
// Connect to  running cluster

JetInstance jet = Jet.newJetClient();
```

@nicolas_frankel

hazelcast®

Pipeline

- Declarative code that defines and links sources, transforms, and sinks
- Platform-specific SDK
- Client submits pipeline to the SPE

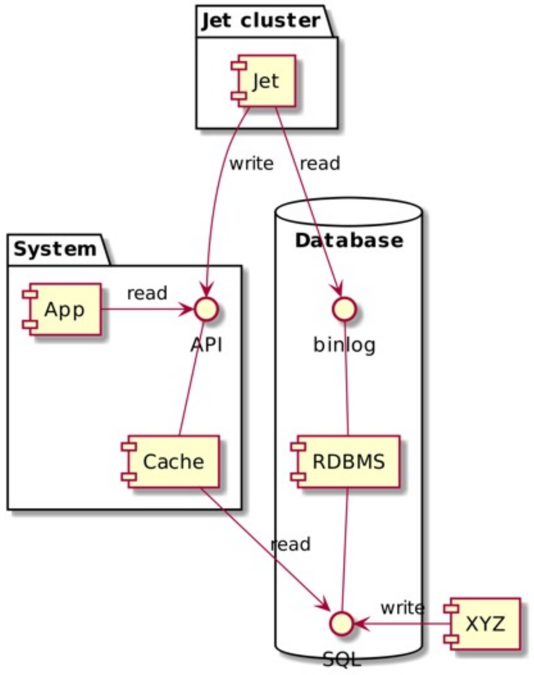
Job

- Running instance of pipeline in SPE
- SPE executes the pipeline
  - Code execution
  - Data routing
  - Flow control

Back to our use-case

A Jet job:

1. Watches change events in the database
2. Analyzes the change event
3. Updates the cache accordingly

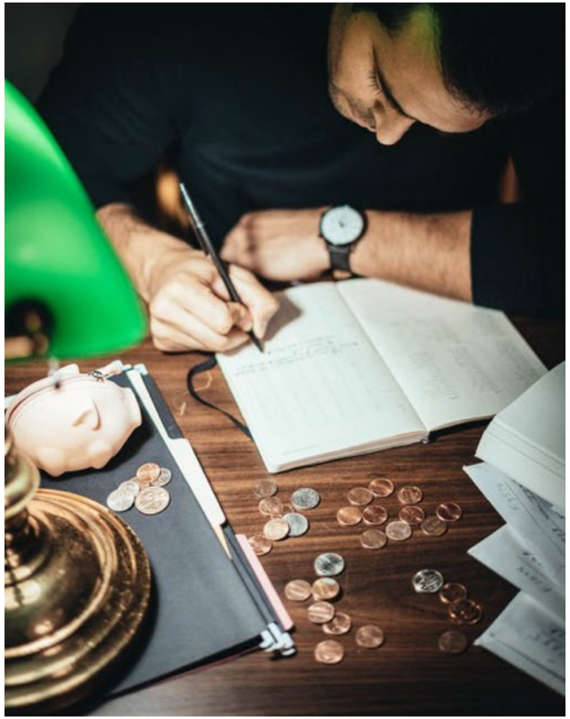


hazelcast®

Talk is cheap, show me the code!

Recap

- The caching trade-off
- CDC bypasses this trade-off
- Implementation through Hazelcast Jet



@nicolas_frankel

hazelcast®

Thanks for your attention!

- https://blog.frankel.ch/
- @nicolas_frankel
- https://jet-start.sh/docs/tutorials/cdc
- https://bit.ly/evergreen-cache
- https://slack.hazelcast.com/
- https://training.hazelcast.com/