HAZELCAST

Running Real-time Machine Learning Analytics On Traces

Fawaz Ghali, PhD

Principal Data Architect and Head of Developer Relations

@fawazghali

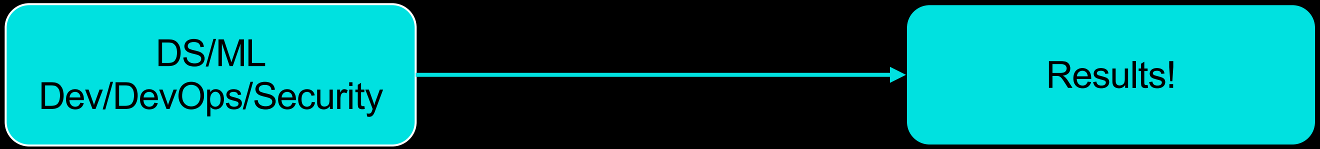
$28B

2020
Actual global credit and debit card fraud losses

$408B

2021-2031
Forecasted global card industry losses to fraud

Sources: (*) ”2022 Half-year fraud update”, UK Banking & Finance Industry (**) Card fraud worldwide, Nilson report, December 2021

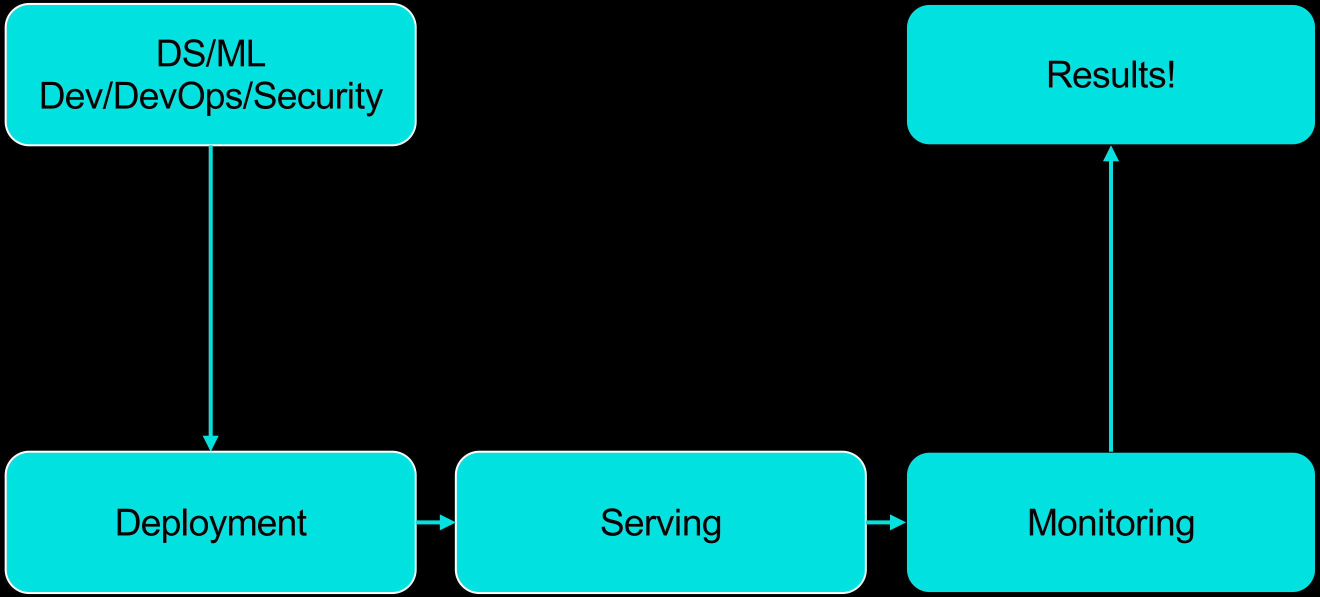


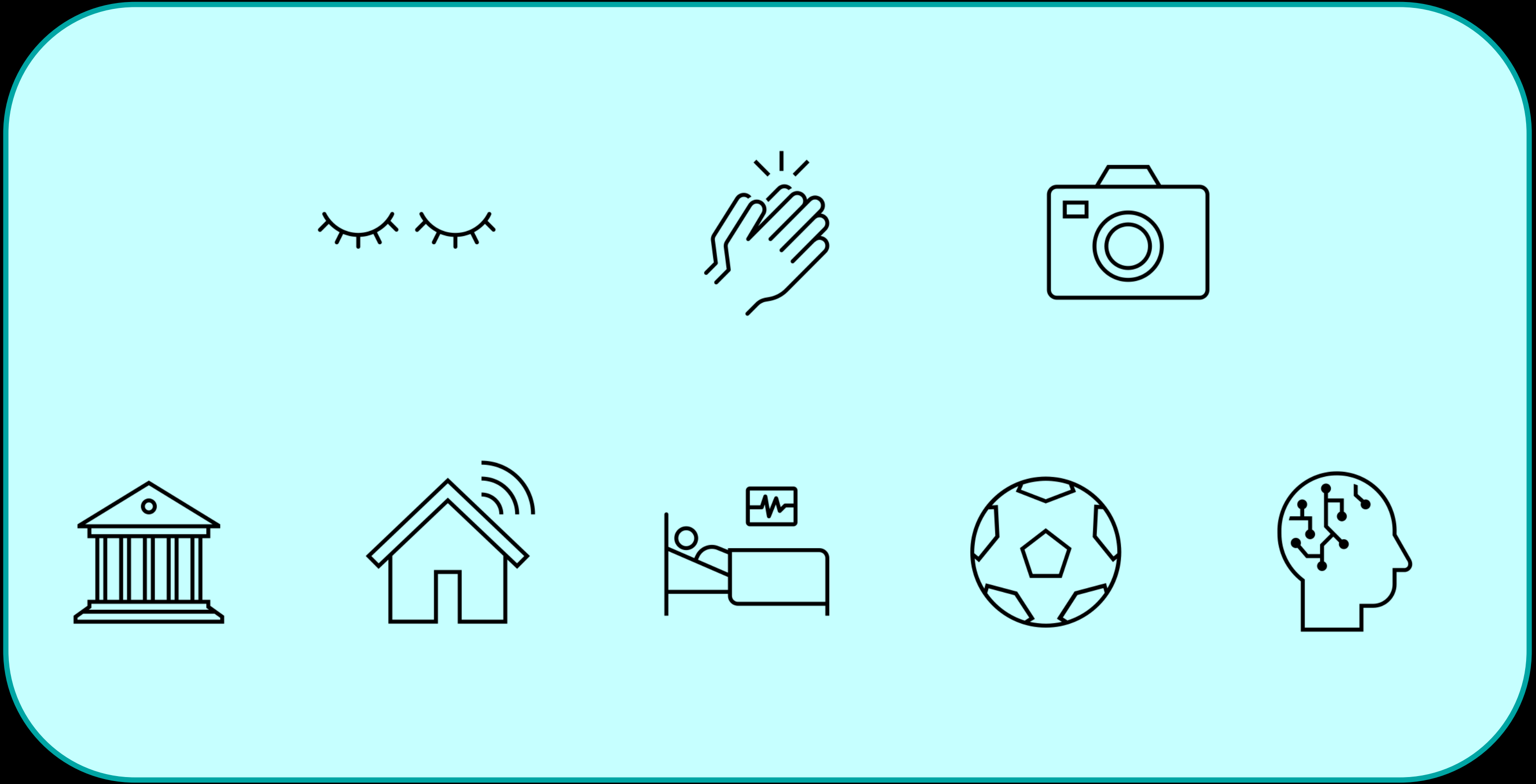
DS/ML
Dev/DevOps/Security

Results!

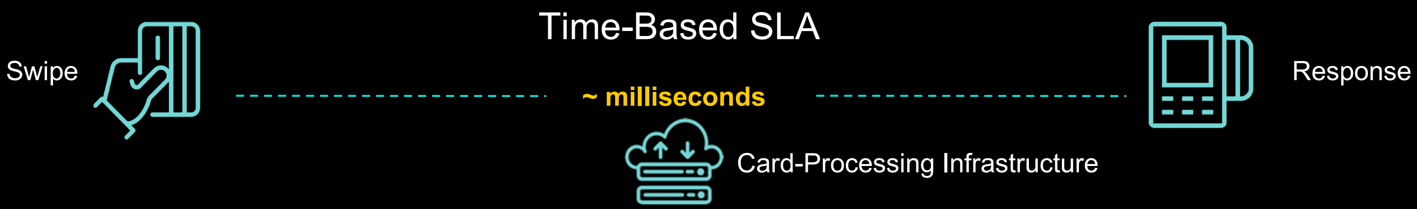
I have the **perfect** App/ML/Service

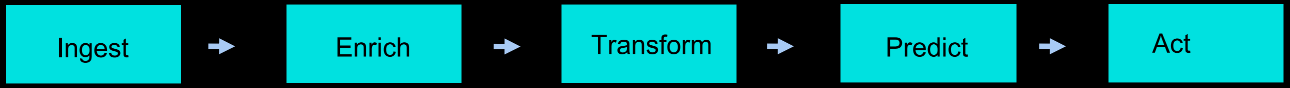
But…


DS/ML
Dev/DevOps/Security
Deployment
Serving
Monitoring
Results!



Card Payment Fraud Detection: Behind the Scenes





{
transaction_number
cc_num = 3002 230..
Amount = 50
Merchant = 931
Device = {iphone}
GeoLocation = 33'N
}

{
as before + Context
Customer data =
[gender, job, city, age group, avg spend]
Merchant data =
[category, avg spend]
Realtime features =
[Distance from home]
}

{
as before plus
“Feature Engineering”:
imputation, scaling, discretization, embeddings, categorical features
}

{
Run ML Model: scikit-learn,LightGBM, Python, Pytorch, Tensorflow Model, Graph ML or call out ML endpoint

Get fraud probability prediction
}

Apply additional checks (business rules & emerging fraud checks)

Accept
or
Decline

High-Performance Real-time

=

Instant computation on both

new and historical data

AI/ML Workloads - Feature Store

Features are individual data attributes of an entity needed by a model to make a prediction

Fraud?	Age Group	Gender	Avg Daily Spend	Transaction Amount	Distance from home	# Transactions in last X mins	merchant type
Yes	18-25	F	20	4.99	2000	10	Gambling
No	26-35	M	8	4.00	2	2	Grocery
No	46-54	F	60	40.00	15	0	Fuel
No	36-45	F	40	260.00	450	1	Electronics
Yes	18-25	M	5	10,000.00	15	15	Jewellery

🟩 Precomputed features
(Can be stored in memory)

🟪 Real-time features

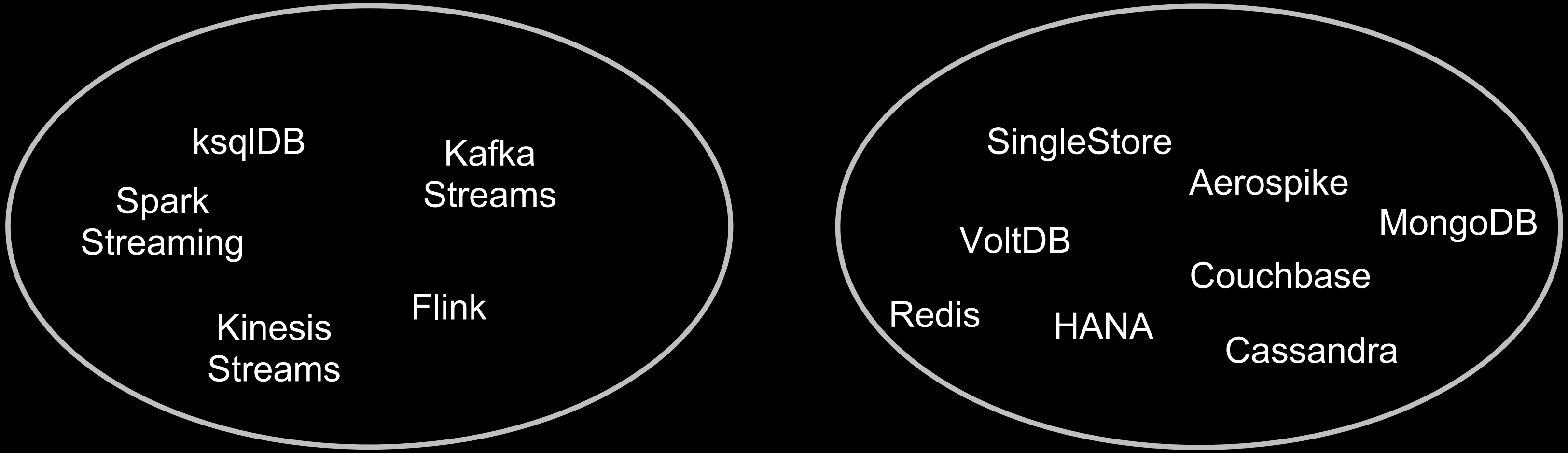
🟨 Streaming features
(calculated in near-real time)

HAZELCAST

Alerts

Trends

Enrichment



ksqlDB

Kafka Streams

Spark Streaming

Flink

Kinesis Streams

SingleStore

Aerospike

MongoDB

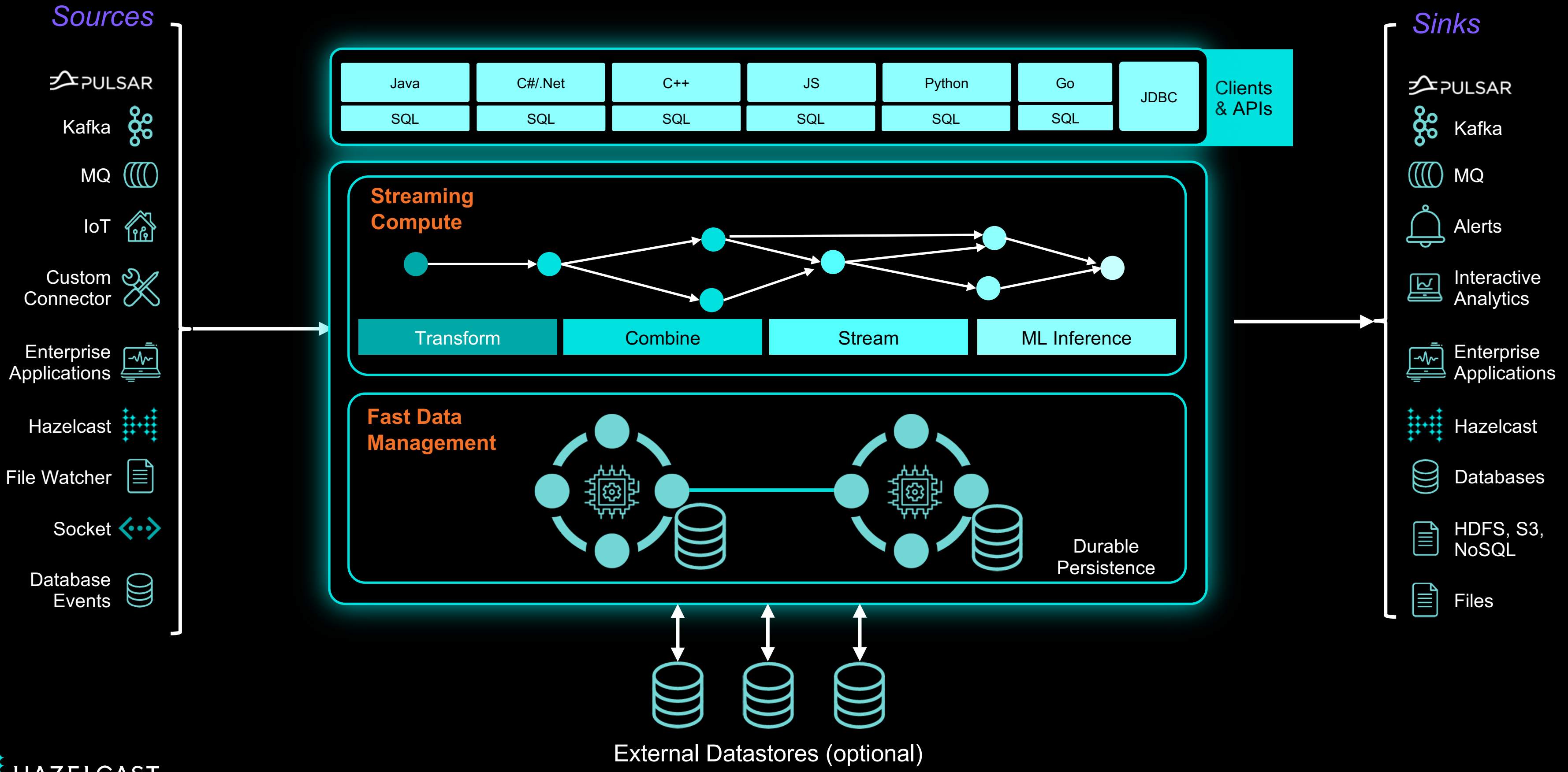
VoltDB

Couchbase

Redis

HANA

Cassandra



Sources

- PULSAR
- Kafka
- MQ
- IoT
- Custom Connector
- Enterprise Applications
- Hazelcast
- File Watcher
- Socket
- Database Events

Java	C#/.Net	C++	JS	Python	Go	JDBC
SQL	SQL	SQL	SQL	SQL	SQL	

Clients & APIs

Streaming Compute

Transform | Combine | Stream | ML Inference

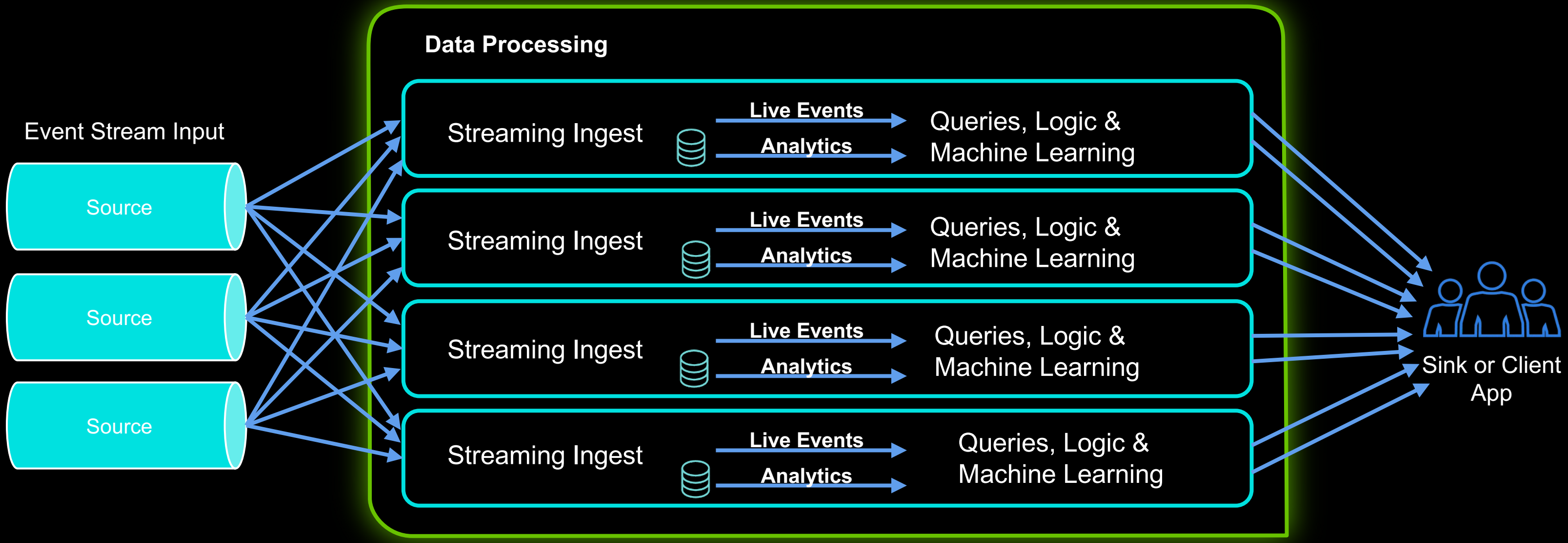
Fast Data Management

Durable Persistence

External Datastores (optional)

Sinks

- PULSAR
- Kafka
- MQ
- Alerts
- Interactive Analytics
- Enterprise Applications
- Hazelcast
- Databases
- HDFS, S3, NoSQL
- Files



Event Stream Input

Source

Source

Source

Data Processing

Streaming Ingest

Live Events

Analytics

Queries, Logic & Machine Learning

Streaming Ingest

Live Events

Analytics

Queries, Logic & Machine Learning

Streaming Ingest

Live Events

Analytics

Queries, Logic & Machine Learning

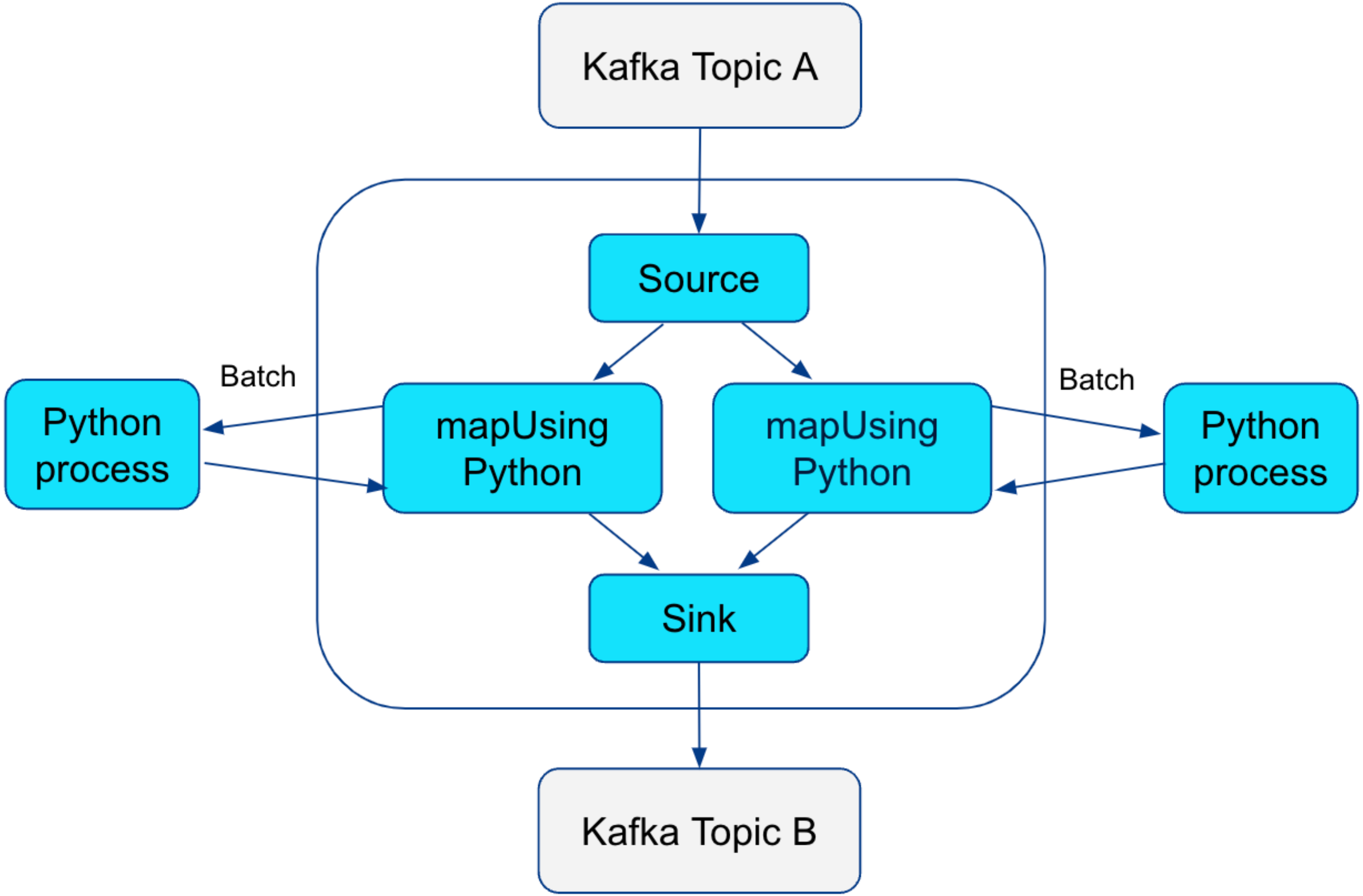
Streaming Ingest

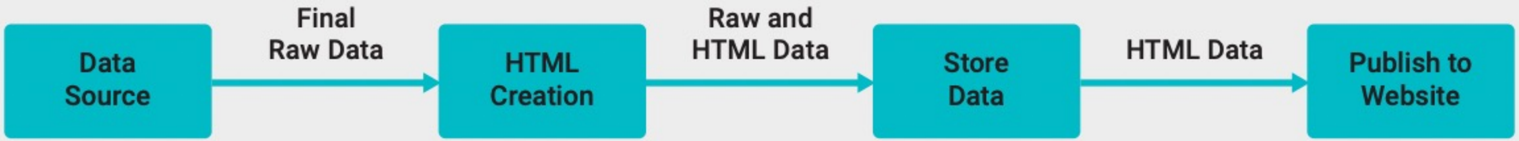
Live Events

Analytics

Queries, Logic & Machine Learning

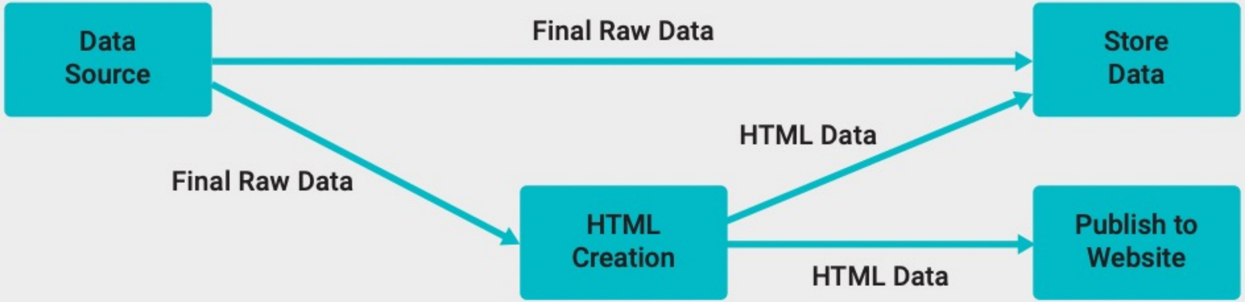
Sink or Client App


Kafka Topic A
Source
Batch
Python process
mapUsing Python
mapUsing Python
Batch
Python process
Sink
Kafka Topic B

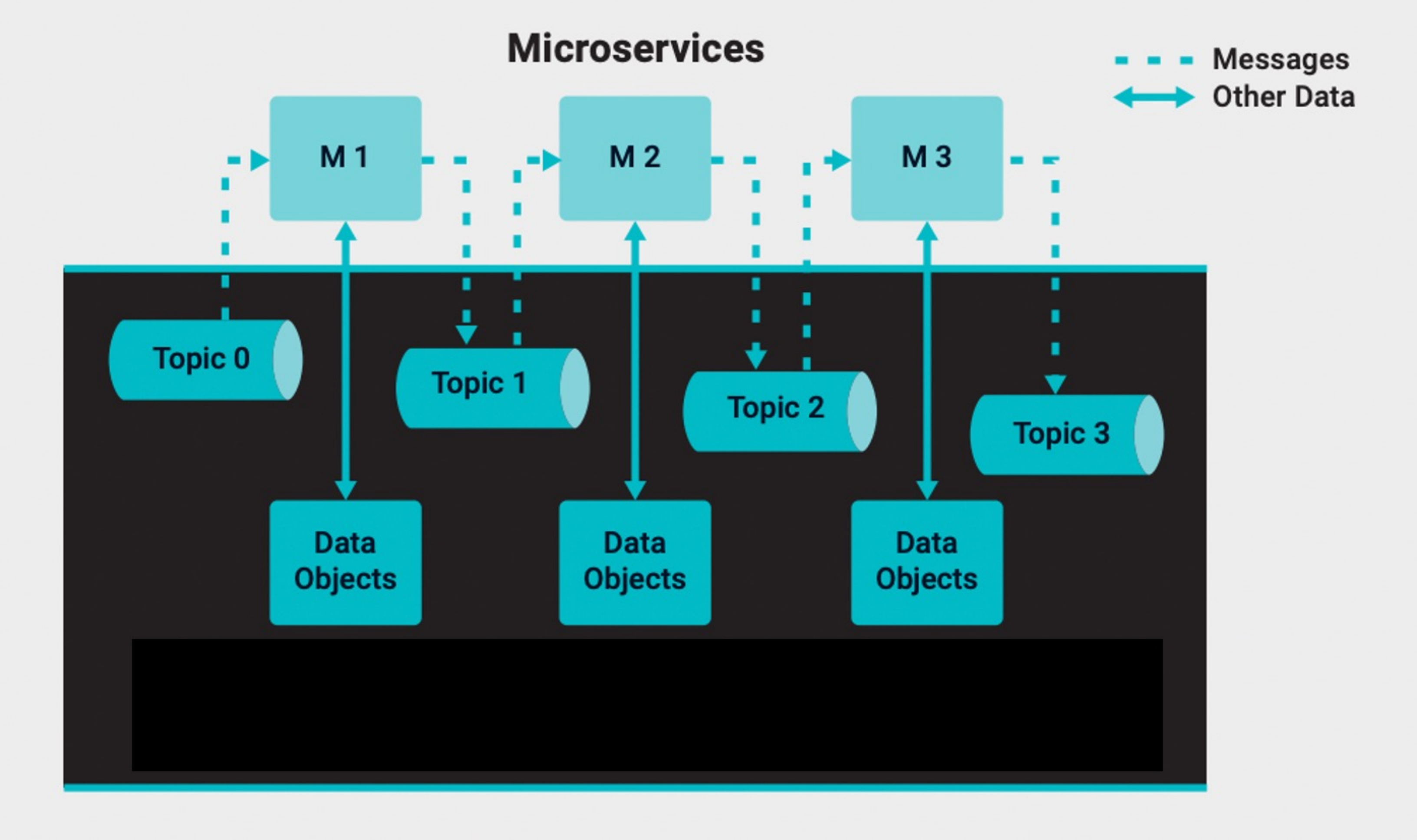


Linear Pipeline

Directed Acyclic Graph



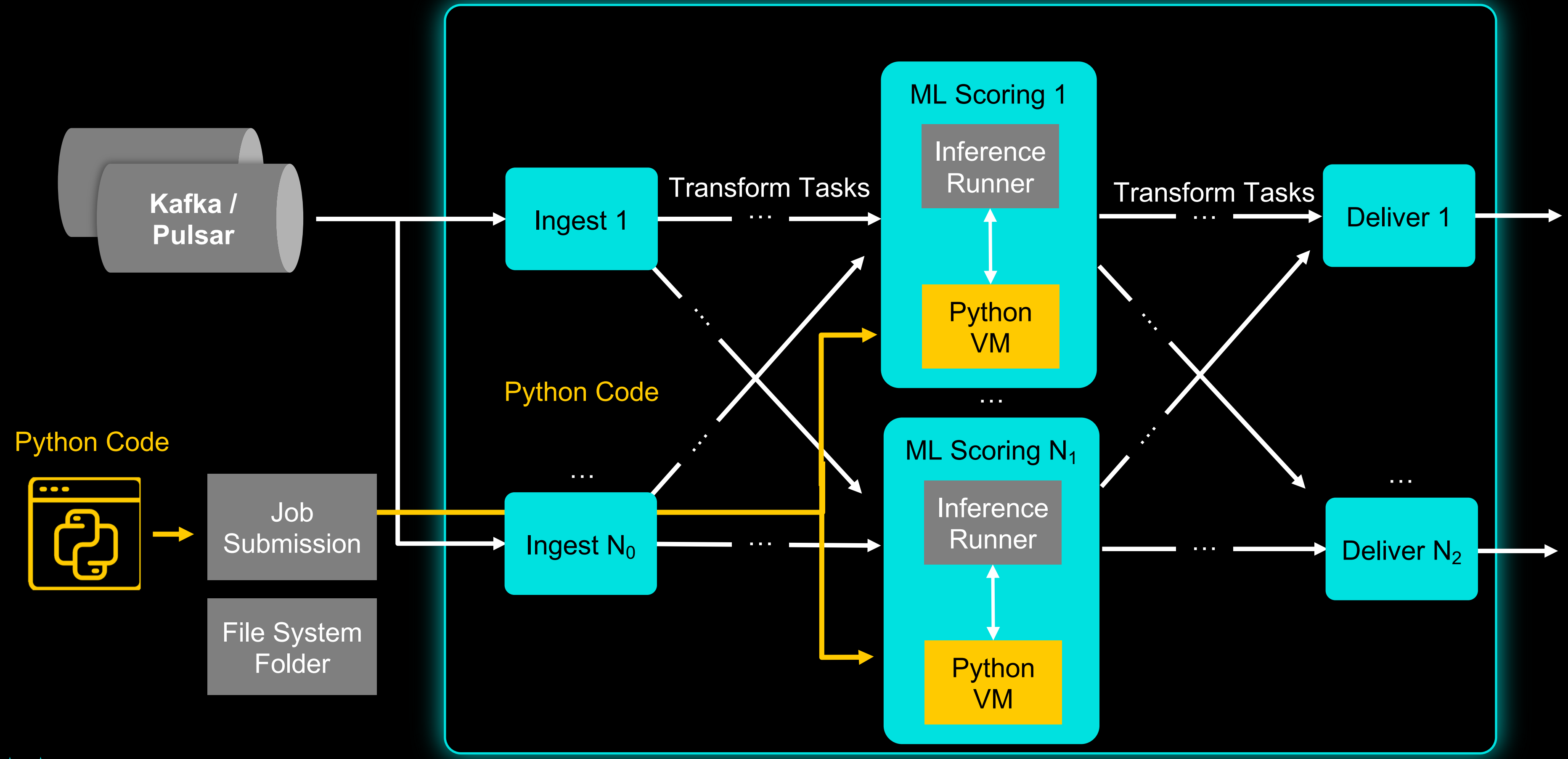
The data flow in a microservices architecture can be represented by a DAG.


Microservices
Messages
Other Data
M 1
M 2
M 3
Topic 0
Topic 1
Topic 2
Topic 3
Data Objects
Data Objects
Data Objects

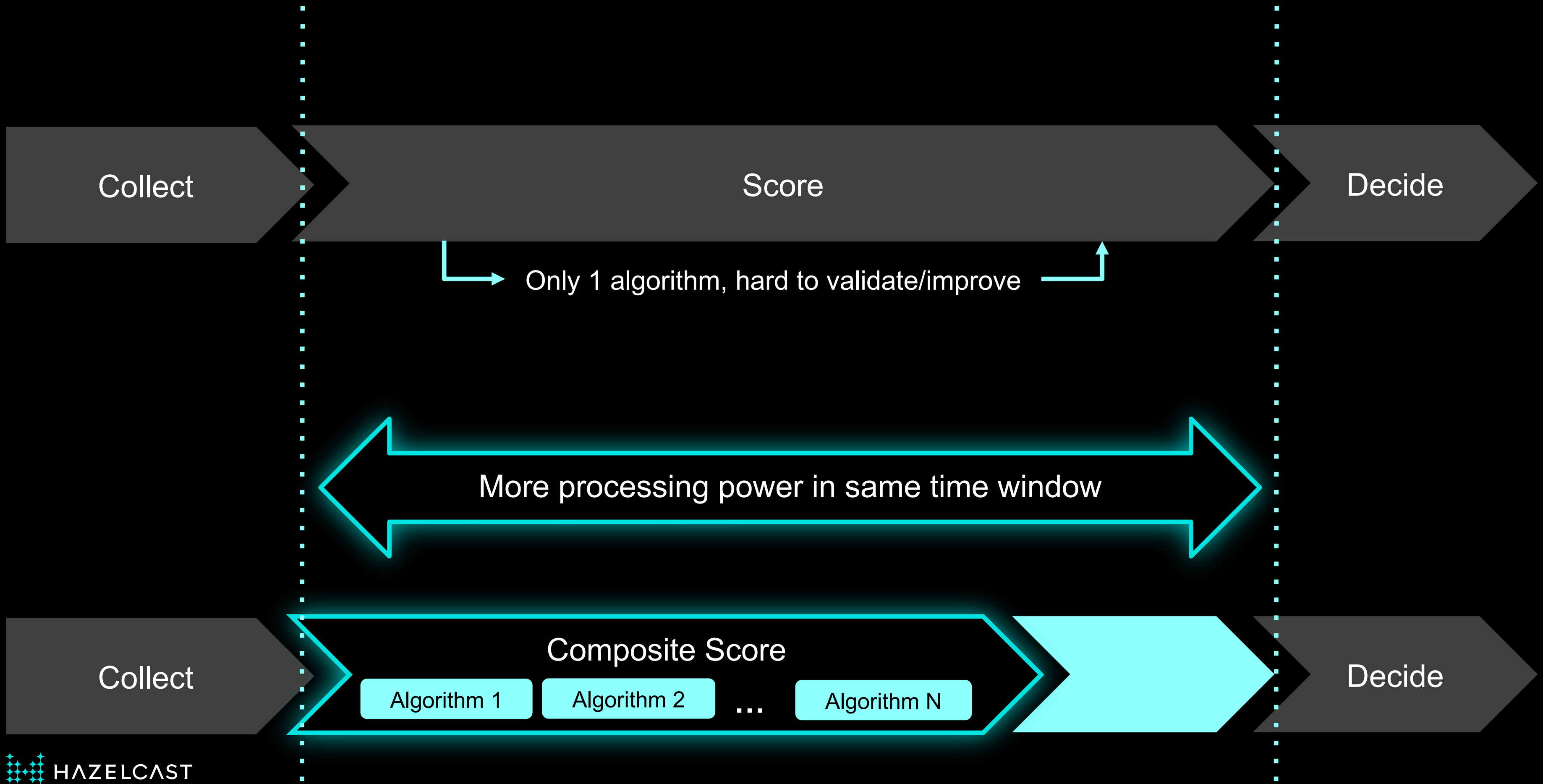
[redacted]

Why Microservices?

- Are Easier to Build and Enhance
- Are Easier to Deploy
- Are Easier to Maintain, Troubleshoot, and Extend
- Simplify Cross-Team Coordination
- Deliver Performance and Scale
- Simplify Real-Time Processing


Kafka / Pulsar
Python Code
Job Submission
File System Folder
Ingest 1
Python Code
…
Ingest N_0
Transform Tasks
…
…
…
ML Scoring 1
Inference Runner
Python VM
…
ML Scoring N_1
Inference Runner
Python VM
Transform Tasks
…
…
Deliver 1
…
Deliver N_2

HAZELCAST



Collect

Score

Decide

Only 1 algorithm, hard to validate/improve

More processing power in same time window

Collect

Composite Score

Algorithm 1

Algorithm 2

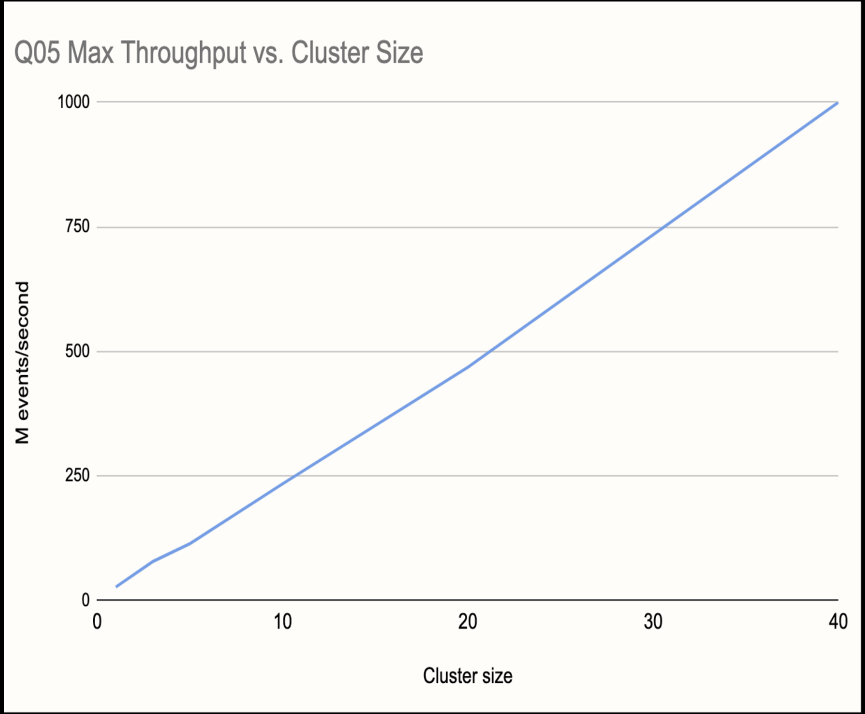
...

Algorithm N

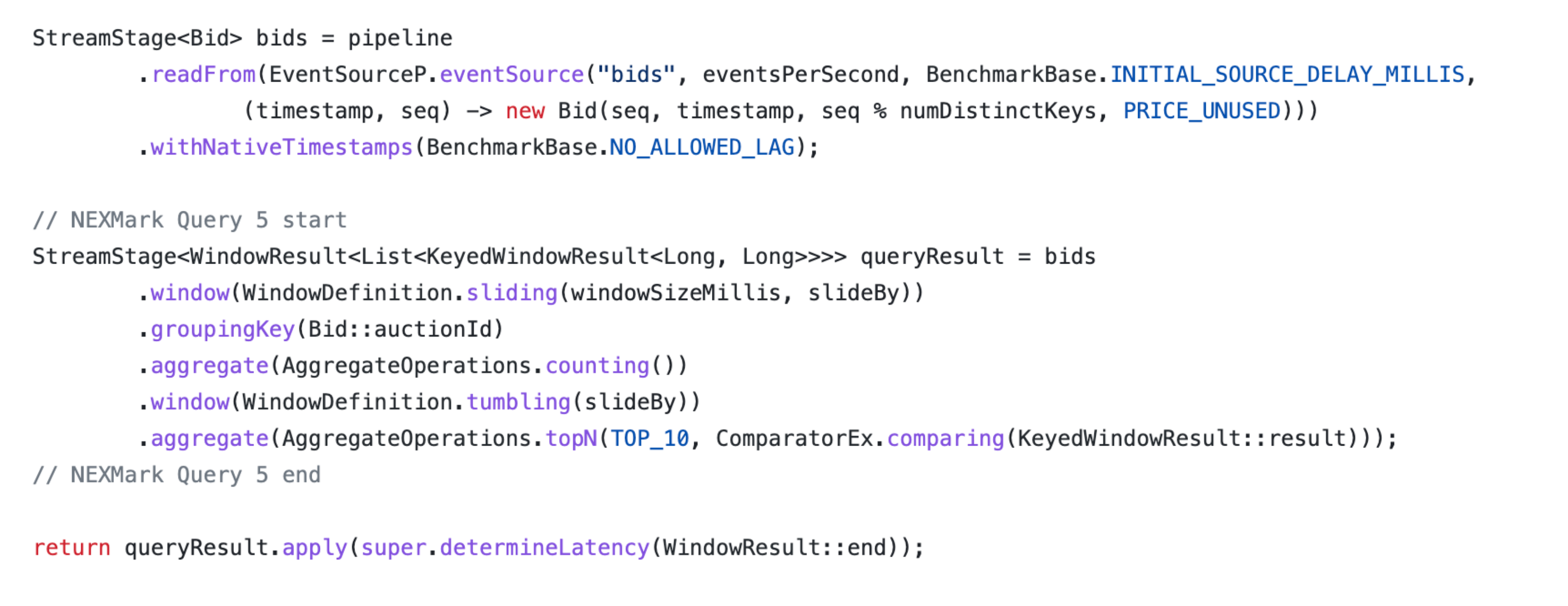
Decide

Gigascale Real-time Data Processing

- 1 BILLION TPS with 99% under 30ms latency
- 45 nodes
- Linear scaling with predictable latency
- `kubectl scale statefulset <<name>> --replicas=45`

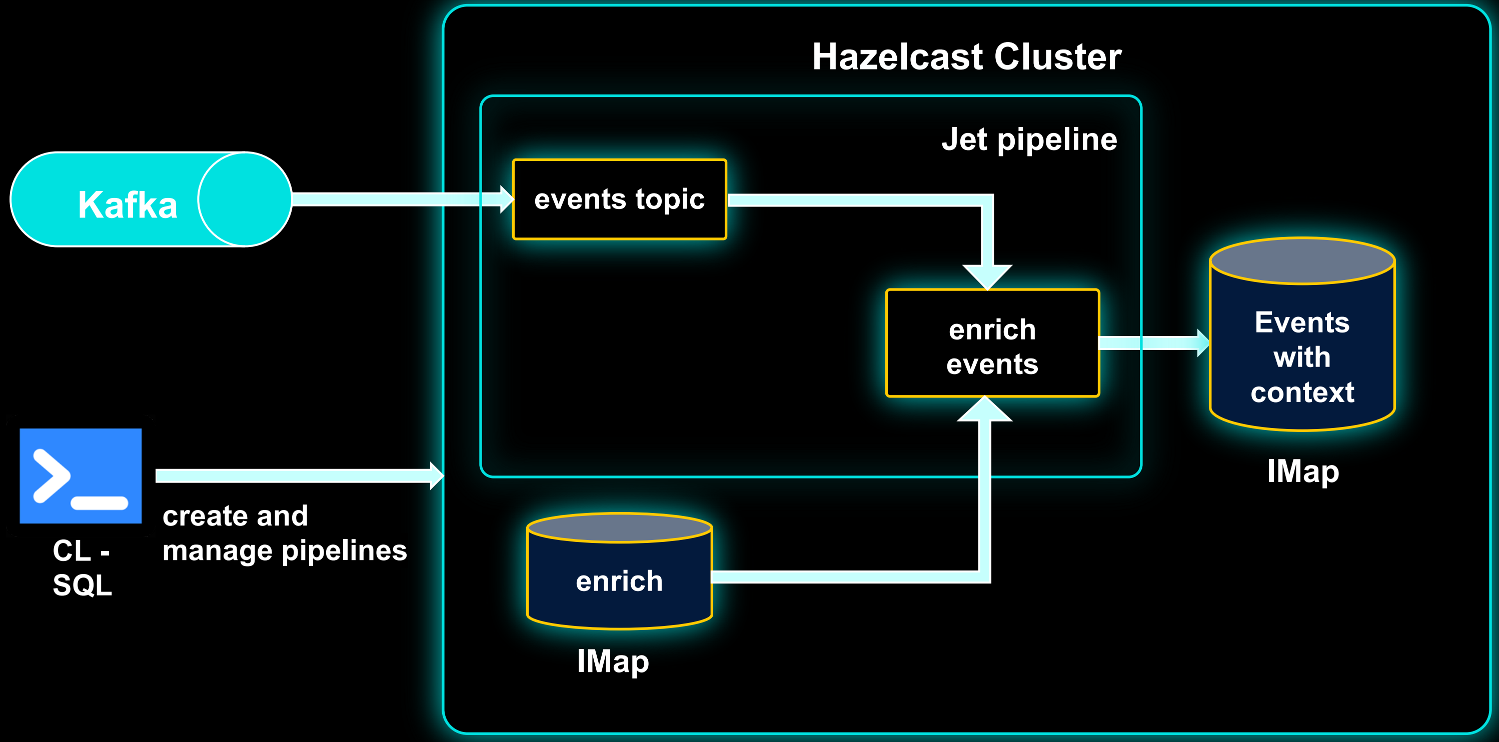


99th Percentile Latency at a Billion Events per Second



```
StreamStage<Bid> bids = pipeline
        .readFrom(EventSourceP.eventSource("bids", eventsPerSecond, BenchmarkBase.INITIAL_SOURCE_DELAY_MILLIS,
                (timestamp, seq) -> new Bid(seq, timestamp, seq % numDistinctKeys, PRICE_UNUSED)))
        .withNativeTimestamps(BenchmarkBase.NO_ALLOWED_LAG);

// NEXMark Query 5 start
StreamStage<WindowResult<List<KeyedWindowResult<Long, Long>>>> queryResult = bids
        .window(WindowDefinition.sliding(windowSizeMillis, slideBy))
        .groupingKey(Bid::auctionId)
        .aggregate(AggregateOperations.counting())
        .window(WindowDefinition.tumbling(slideBy))
        .aggregate(AggregateOperations.topN(TOP_10, ComparatorEx.comparing(KeyedWindowResult::result)));
// NEXMark Query 5 end

return queryResult.apply(super.determineLatency(WindowResult::end));
```

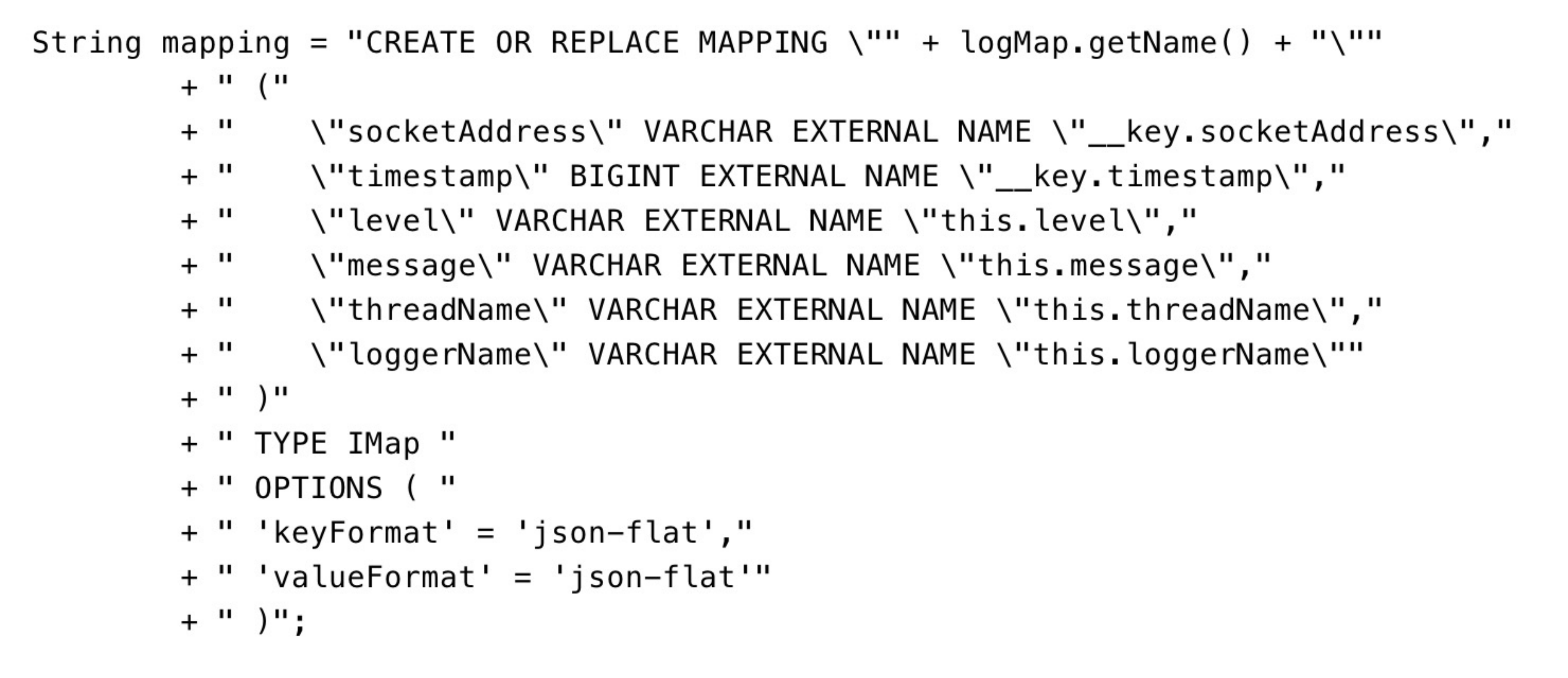
Architecture Overview



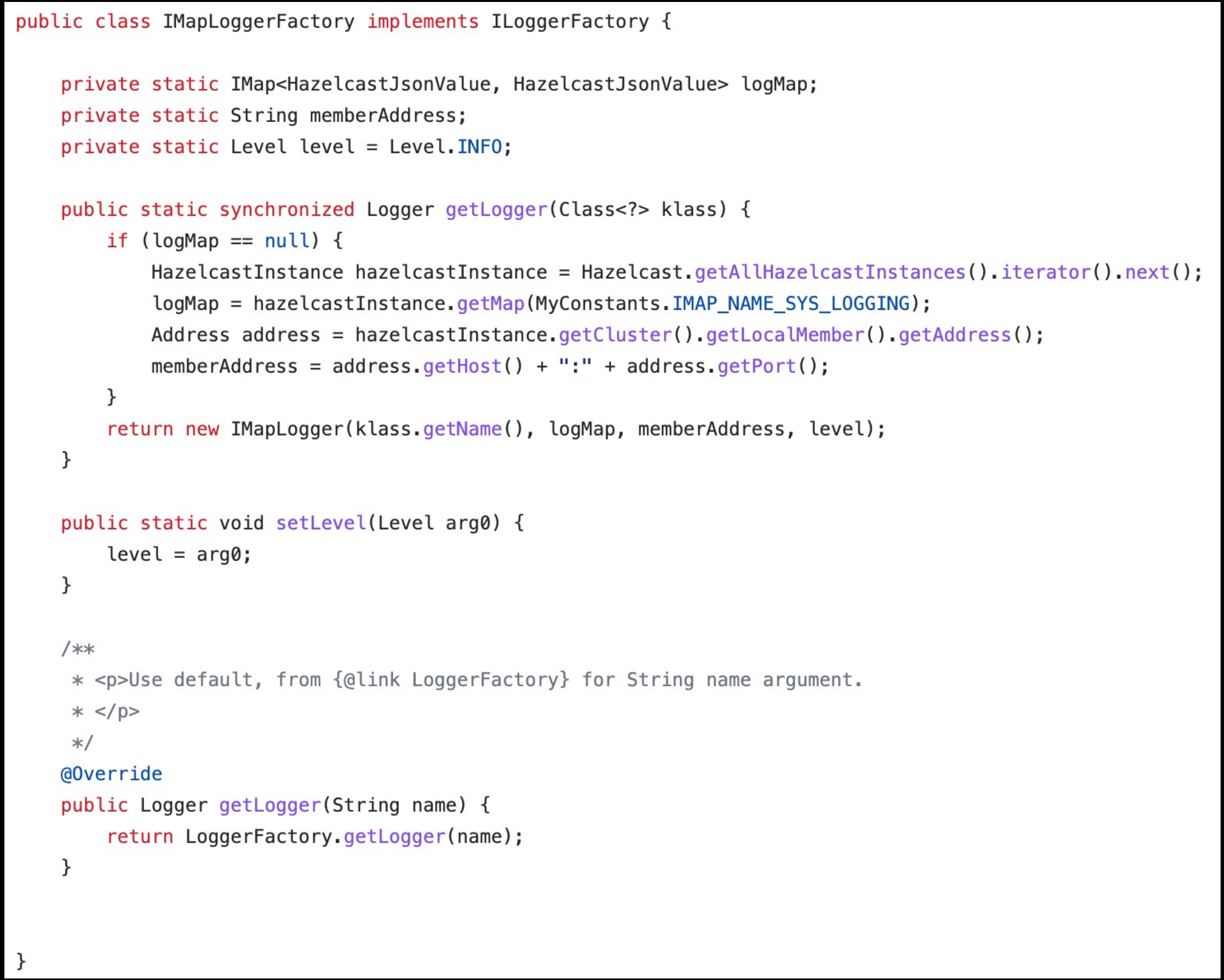
Setup

- Install Hazelcast
  - **hz-start**
  - **hz-cli sql**
- Install Kafka
  - cd Documents/kafka_2.13-3.4.0
  - **bin/zookeeper-server-start.sh config/zookeeper.properties**
  - **bin/kafka-server-start.sh config/server.properties**
  - bin/kafka-server-stop.sh
  - bin/zookeeper-server-stop.sh

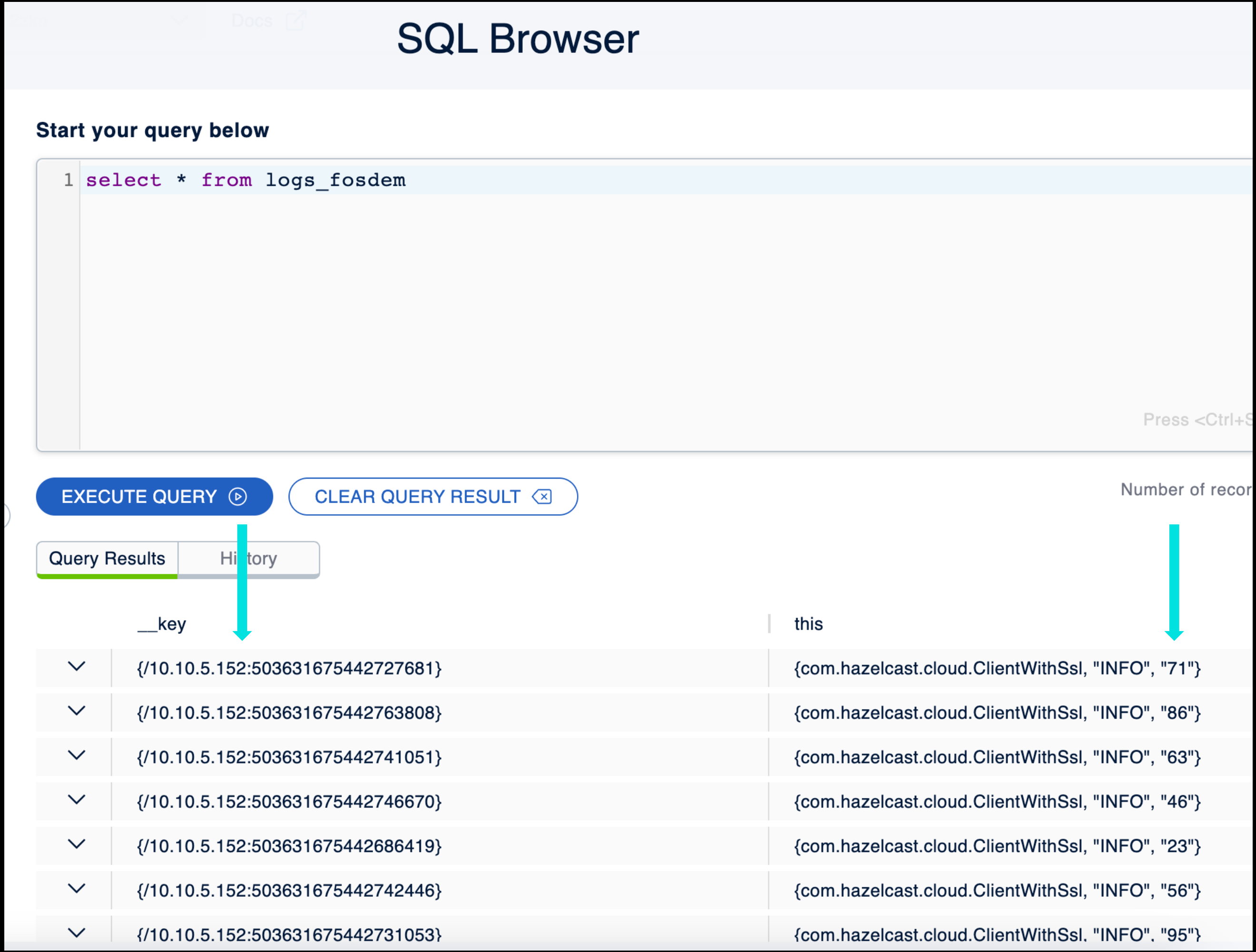
Option #1: Alerts



```
String mapping = "CREATE OR REPLACE MAPPING \"" + logMap.getName() + "\""
        + " ("
        + "    \"socketAddress\" VARCHAR EXTERNAL NAME \"__key.socketAddress\","
        + "    \"timestamp\" BIGINT EXTERNAL NAME \"__key.timestamp\","
        + "    \"level\" VARCHAR EXTERNAL NAME \"this.level\","
        + "    \"message\" VARCHAR EXTERNAL NAME \"this.message\","
        + "    \"threadName\" VARCHAR EXTERNAL NAME \"this.threadName\","
        + "    \"loggerName\" VARCHAR EXTERNAL NAME \"this.loggerName\""
        + " )"
        + " TYPE IMap "
        + " OPTIONS ( "
        + " 'keyFormat' = 'json-flat',"
        + " 'valueFormat' = 'json-flat'"
        + " )";
```



```java
public class IMapLoggerFactory implements ILoggerFactory {

    private static IMap<HazelcastJsonValue, HazelcastJsonValue> logMap;
    private static String memberAddress;
    private static Level level = Level.INFO;

    public static synchronized Logger getLogger(Class<?> klass) {
        if (logMap == null) {
            HazelcastInstance hazelcastInstance = Hazelcast.getAllHazelcastInstances().iterator().next();
            logMap = hazelcastInstance.getMap(MyConstants.IMAP_NAME_SYS_LOGGING);
            Address address = hazelcastInstance.getCluster().getLocalMember().getAddress();
            memberAddress = address.getHost() + ":" + address.getPort();
        }
        return new IMapLogger(klass.getName(), logMap, memberAddress, level);
    }

    public static void setLevel(Level arg0) {
        level = arg0;
    }

    /**
     * <p>Use default, from {@link LoggerFactory} for String name argument.
     * </p>
     */
    @Override
    public Logger getLogger(String name) {
        return LoggerFactory.getLogger(name);
    }


}
```



SQL Browser

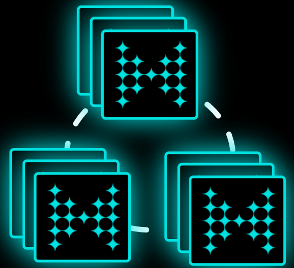
Start your query below

```
select * from logs_fosdem
```

__key	this
{/10.10.5.152:503631675442727681}	{com.hazelcast.cloud.ClientWithSsl, "INFO", "71"}
{/10.10.5.152:503631675442763808}	{com.hazelcast.cloud.ClientWithSsl, "INFO", "86"}
{/10.10.5.152:503631675442741051}	{com.hazelcast.cloud.ClientWithSsl, "INFO", "63"}
{/10.10.5.152:503631675442746670}	{com.hazelcast.cloud.ClientWithSsl, "INFO", "46"}
{/10.10.5.152:503631675442686419}	{com.hazelcast.cloud.ClientWithSsl, "INFO", "23"}
{/10.10.5.152:503631675442742446}	{com.hazelcast.cloud.ClientWithSsl, "INFO", "56"}
{/10.10.5.152:503631675442731053}	{com.hazelcast.cloud.ClientWithSsl, "INFO", "95"}

Option #2: Trends

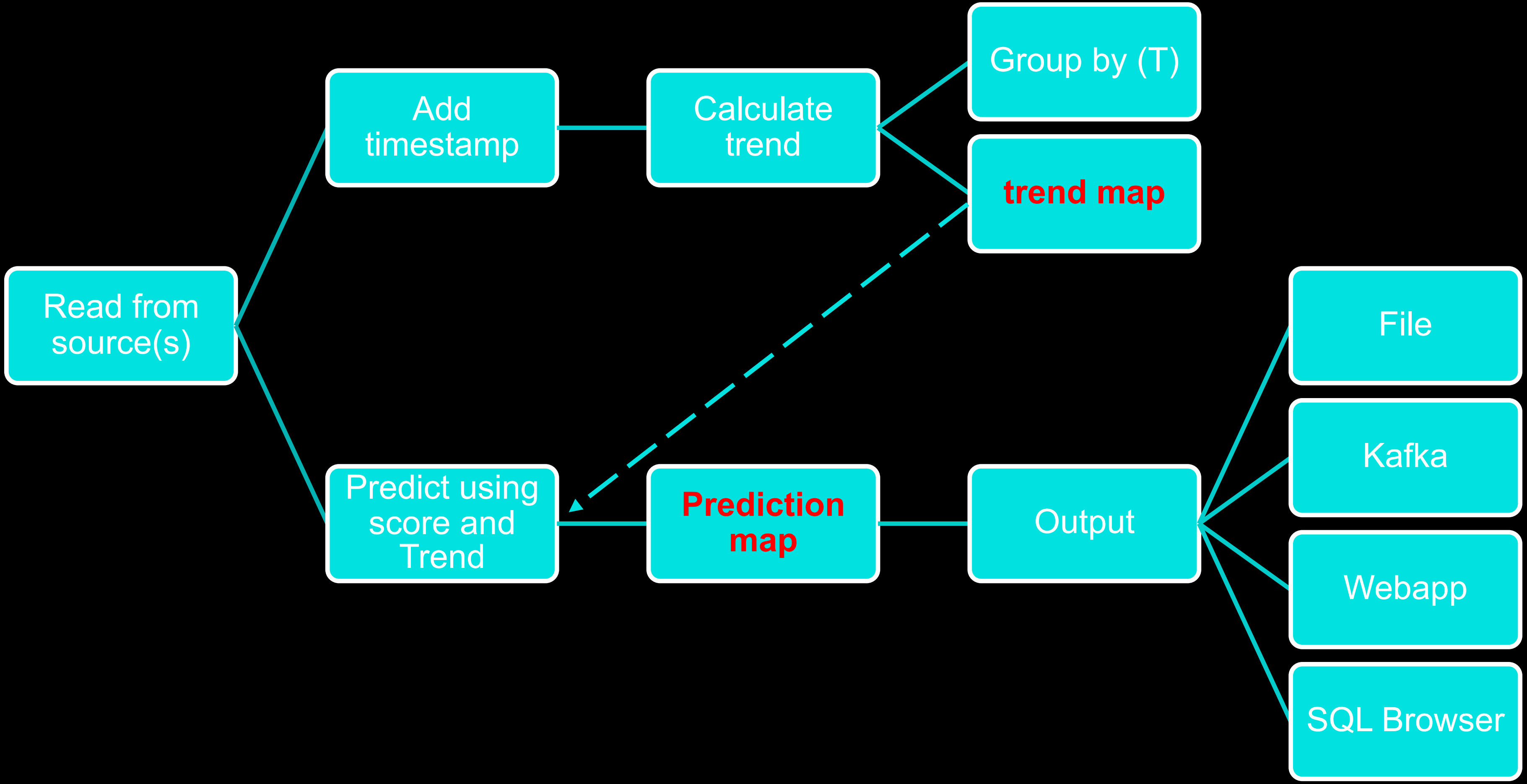
```
56.8.199.213,1652731706,86
149.249.213.20,1205849525,76
83.58.157.127,1660017403,76
120.231.214.220,1417539947,1
161.33.208.167,1636079422,67
211.90.113.86,1636079422,87
25.233.56.52,1653416533,78
165.172.37.69,962254265,66
153.23.56.53,1766405806,58
78.224.25.157,1766405806,91
```



Trend

Predict

```
56.8.199.213,1652731706,1
149.249.213.20,1205849525,
83.58.157.127,1660017403,1
120.231.214.220,1417539947,1
161.33.208.167,1636079422,0
211.90.113.86,1636079422,0
25.233.56.52,1653416533,0
165.172.37.69,962254265,0
153.23.56.53,1766405806,1
78.224.25.157,1766405806,1
```


Read from source(s)

Add timestamp

Calculate trend

Group by (T)

trend map

Predict using score and Trend

Prediction map

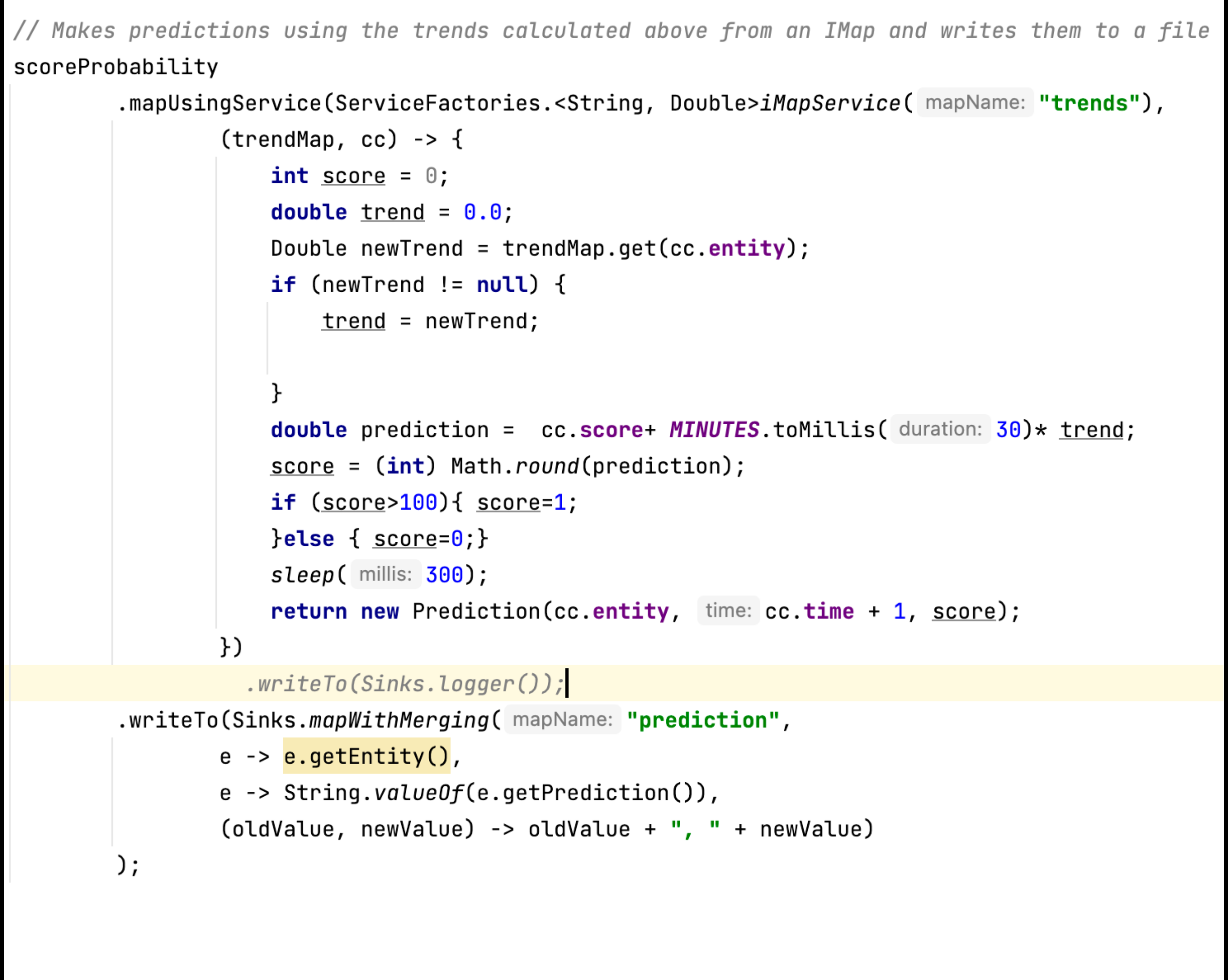
Output

File

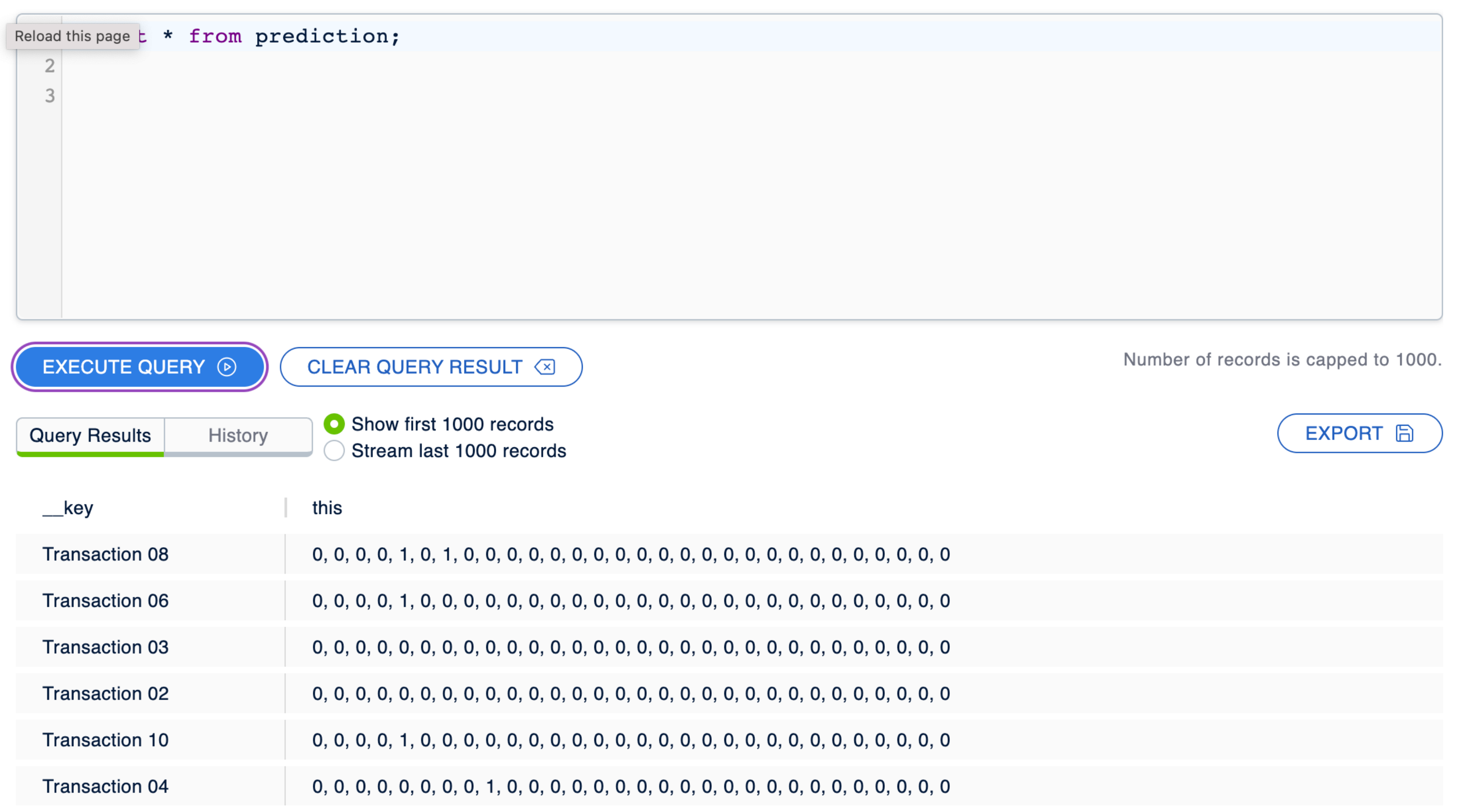
Kafka

Webapp

SQL Browser



```
// Makes predictions using the trends calculated above from an IMap and writes them to a file
scoreProbability
        .mapUsingService(ServiceFactories.<String, Double>iMapService( mapName: "trends"),
                (trendMap, cc) -> {
                    int score = 0;
                    double trend = 0.0;
                    Double newTrend = trendMap.get(cc.entity);
                    if (newTrend != null) {
                        trend = newTrend;

                    }
                    double prediction =  cc.score+ MINUTES.toMillis( duration: 30)* trend;
                    score = (int) Math.round(prediction);
                    if (score>100){ score=1;
                    }else { score=0;}
                    sleep( millis: 300);
                    return new Prediction(cc.entity,  time: cc.time + 1, score);
                })
                  .writeTo(Sinks.logger());
        .writeTo(Sinks.mapWithMerging( mapName: "prediction",
                e -> e.getEntity(),
                e -> String.valueOf(e.getPrediction()),
                (oldValue, newValue) -> oldValue + ", " + newValue)
        );
```



```
t * from prediction;
```

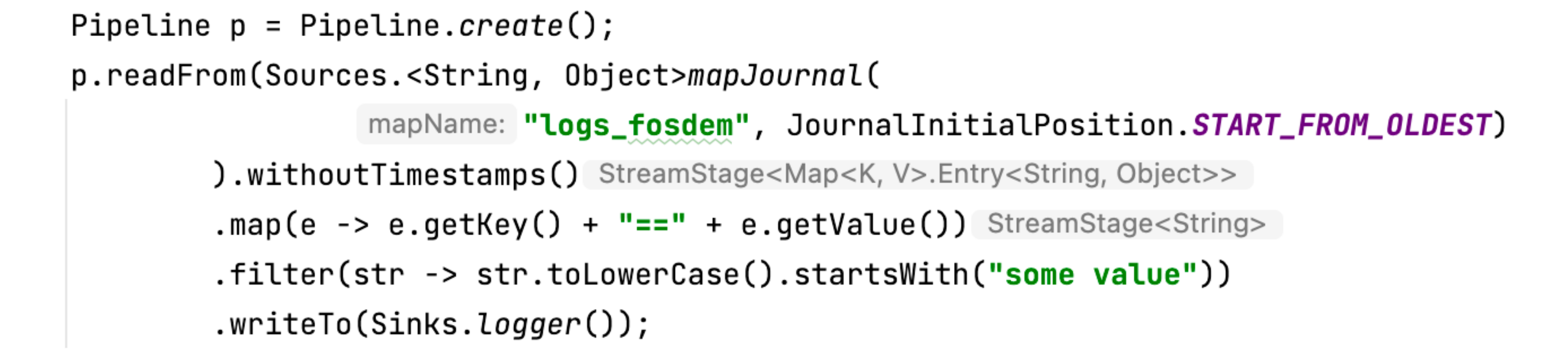
Number of records is capped to 1000.

● Show first 1000 records
○ Stream last 1000 records

__key	this
Transaction 08	0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
Transaction 06	0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
Transaction 03	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
Transaction 02	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
Transaction 10	0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
Transaction 04	0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

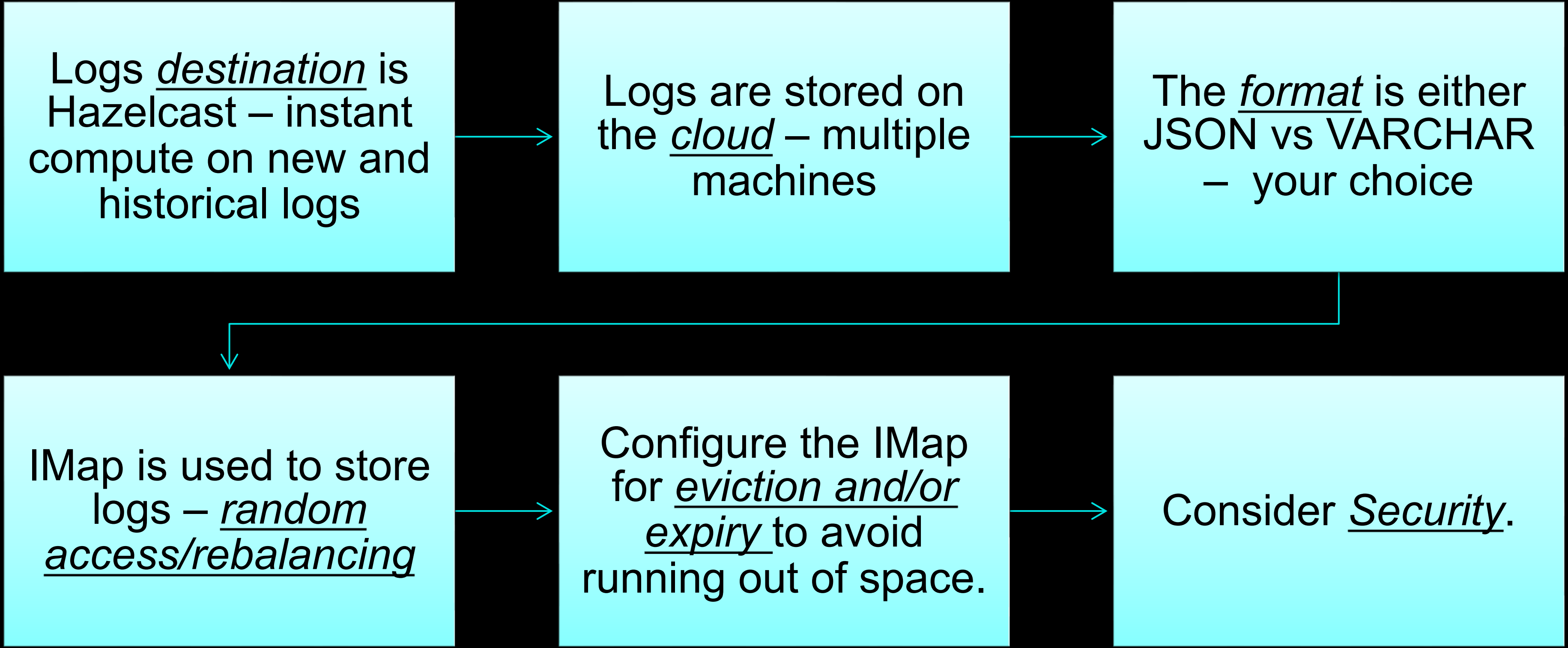
3 Ways to proceed

- (1) SQL CREATE JOB fosdem_job AS SINK INTO map2 SELECT * FROM map1
- (2) Pipeline: readFrom(Sources.map(”logs_fosdem"))
- (3) Pipeline: readFrom(Sources.mapJournal(" logs_fosdem ") -- Due in 5.3 for SQL
    - as this is continuous, you get changes to a map
    - The journal is a ringbuffer, so you can start your stream from the first or the last entry



```
Pipeline p = Pipeline.create();
p.readFrom(Sources.<String, Object>mapJournal(
                mapName: "logs_fosdem", JournalInitialPosition.START_FROM_OLDEST)
        ).withoutTimestamps() StreamStage<Map<K, V>.Entry<String, Object>>
        .map(e -> e.getKey() + "==" + e.getValue()) StreamStage<String>
        .filter(str -> str.toLowerCase().startsWith("some value"))
        .writeTo(Sinks.logger());
```

Summary



Logs *destination* is Hazelcast – instant compute on new and historical logs

Logs are stored on the *cloud* – multiple machines

The *format* is either JSON vs VARCHAR – your choice

IMap is used to store logs – *random access/rebalancing*

Configure the IMap for *eviction and/or expiry* to avoid running out of space.

Consider *Security*.

https://slack.hazelcast.com/