Ingesting Streaming Data for Analysis in Apache Ignite

Pat Patterson
StreamSets
pat@streamsets.com
@metadaddy
Agenda

Product Support Use Case

Continuous Queries in Apache Ignite

Integrating StreamSets Data Collector with Apache Ignite

Demo

Wrap-up
Who is StreamSets?

<table>
<thead>
<tr>
<th>Seasoned leadership team</th>
<th>Customer base from global 8000</th>
<th>Unique commercial downloaders</th>
</tr>
</thead>
<tbody>
<tr>
<td>cloudera</td>
<td>50%</td>
<td>2000+</td>
</tr>
<tr>
<td>Informatica</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Open source downloads worldwide</th>
<th>Broad connectivity</th>
<th>History of innovation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000,000+</td>
<td>50+</td>
<td></td>
</tr>
</tbody>
</table>

50% worldwide downloads

50+ broad connectivity

Spark

Gorgias
Use Case: Product Support

HR system (on-premises RDBMS) holds employee reporting hierarchy

Customer service platform (SaaS) holds support ticket status, assignment to support engineers

Device monitoring system (CSV files / JSON via Kafka) provides fault data

How do we query across data sources?

How do we get notifications of faults for high-priority tickets?
Apache Ignite Continuous Queries

Enable you to listen to data modifications occurring on Ignite caches

Specify optional initial query, remote filter, local listener

Initial query can be any type: Scan, SQL, or TEXT

Remote filter executes on primary and backup nodes
 Apache Ignite Continuous Queries

Local listener executes in your app’s JVM

Can use BinaryObjects for generic, performant code

Can use `ContinuousQueryWithTransformer` to run a remote transformer
  • Restrict results to a subset of the available fields
Continuous Query with Binary Objects – Setup

// Get a cache object
IgniteCache<Object, BinaryObject> cache = ignite.cache(cacheName).withKeepBinary();

// Create a continuous query
ContinuousQuery<Object, BinaryObject> qry = new ContinuousQuery<>();

// Set an initial query - match a field value
qry.setInitialQuery(new ScanQuery<>((IgniteBiPredicate<Object, BinaryObject>) (key, val) -> {
    System.out.println("### applying initial query predicate");
    return val.field(filterFieldName).toString().equals(filterFieldValue);
});

// Filter the cache updates
qry.setRemoteFilterFactory(() -> event -> {
    System.out.println("### evaluating cache entry event filter");
    return event.getValue().field(filterFieldName).toString().equals(filterFieldValue);
});
// Process notifications
qry.setLocalListener((evts) -> {
    for (CacheEntryEvent<? extends Object, ? extends BinaryObject> e : evts) {
        Object key = e.getKey();
        BinaryObject newValue = e.getValue();

        System.out.println("Cache entry with ID: " + e.getKey() + " was " + e.getEventType().toString().toLowerCase());
        BinaryObject oldValue = (e.isOldValueAvailable()) ? e.getOldValue() : null;
        processChange(key, oldValue, newValue);
    }
});
Continuous Query with Binary Objects – Run the Query

// Run the continuous query
try (QueryCursor<Cache.Entry<Object, BinaryObject>> cur = cache.query(qry)) {
    // Iterate over existing cache data
    for (Cache.Entry<Object, BinaryObject> e : cur) {
        processRecord(e.getKey(), e.getValue());
    }

    // Sleep until killed
    boolean done = false;
    while (!done) {
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            done = true;
        }
    }
}
Demo: Continuous Query Basics
Continuous Query with Transformer

```java
ContinuousQueryWithTransformer<Object, BinaryObject, String> qry =
    new ContinuousQueryWithTransformer<>();

// Transform result - executes remotely
qry.setRemoteTransformerFactory(() -> event -> {
    System.out.println("### applying transformation");
    return event.getValue().field(fieldName).toString();
});

// Process notifications - executes locally
qry.setLocalListener((values) -> {
    for (String value : values) {
        System.out.println(transformerField + ": " + value);
    }
});
```
Demo: Continuous Query Transformers
Key Learnings

Need to enable peer class loading so that app can send code to execute on remote node

• `<property name="peerClassLoadingEnabled" value="true"/></property>`

By default, CREATE TABLE City ... in SQL means you have to use

`ignite.cache("SQL_PUBLIC_CITY")`

• Override when creating table with `cache_name=city`

Binary Objects make your life simpler and faster

RTFM! `CacheContinuousQueryExample.java` is very helpful!
A Swiss Army Knife for Data
SerialNumber, Timestamp, FaultCode
7326001, 2018-09-18 00:00:00, 0
...

INSERT INTO FAULT (FAULT, ID, SERIAL_NUMBER, TIMESTAMP) VALUES (?, ?, ?, ?)
Ignite JDBC Driver

org.apache.ignite.IgniteJdbcThinDriver

*Thin* driver - connects to cluster node

Located in `ignite-core-version.jar`

JDBC URL of form:

```
jdbc:ignite:thin://hostname[:port1..port2][,hostname...][/schema][?<params>]
```

NOTE – when querying metadata: table, column names must be **UPPERCASE!!!** - IGNITE-9730

There is also the JDBC *Client* Driver – starts its own client node
Demo: Ingest from CSV
Demo: Ingest Kafka, MySQL, Salesforce
IGNITE-9606 breaks JDBC integrations 😞

metadata.getPrimaryKeys() returns _KEY as the column name, returns column name as primary key name

Had to build an ugly workaround into the JDBC Consumer to get my demo working:

```java
ResultSet result = metadata.getPrimaryKeys(connection.getCatalog(), schema, table);
while (result.next()) {
    // Workaround for Ignite bug
    String pk = result.getString(COLUMN_NAME);
    if ("_KEY".equals(pk)) {
        pk = result.getString(PK_NAME);
    }
    keys.add(pk);
}
```
Continuous Streaming Application

Listen for high priority service tickets

Get the last 10 sensor readings for the affected device
Continuous Streaming Application

// SQL query to run with serial number
SqlFieldsQuery sql = new SqlFieldsQuery(
    "SELECT timestamp, fault FROM fault WHERE serial_number = ? ORDER BY timestamp DESC LIMIT 10"
);

// Process notifications - executes locally
qry.setLocalListener((values) -> {
    for (String serialNumber : values) {
        System.out.println("Device serial number ": serialNumber);
        System.out.println("Last 10 faults:");
        QueryCursor<List<?>> query = faultCache.query(sql.setArgs(serialNumber));
        for (List<?> result : query.getAll()) {
            System.out.println(result.get(0) + " | " + result.get(1));
        }
    }
});
Demo: Ignite Streaming App
Other Common StreamSets Use Cases

- Data Lake Replatforming
- IoT
- Cybersecurity
- Real-time applications
Customer Success

“StreamSets allowed us to build and operate over 175,000 pipelines and synchronize 97% of our structured data in R&D to our Data Lake within 4 months. This will save us billions of dollars.”

“We chose StreamSets over NiFi as our enterprise-wide standard for our next generation data lake infrastructure because of their singular focus on solving deployment and operations challenges.”

“It’s simple and easy enough that we don’t need to find a StreamSets developer to create their own data pipelines. Before, it could take 90 days just to find a traditional ETL developer.”
Ignite’s continuous queries provide a robust notification mechanism for acting on changing data

Ignite’s thin JDBC driver is flawed, but useful

StreamSets Data Collector can read data from a wide variety of sources and write to Ignite via the JDBC driver
References

Apache Ignite Continuous Queries
apacheignite.readme.io/docs/continuous-queries

Apache Ignite JDBC Driver
apacheignite-sql.readme.io/docs/jdbc-driver

Download StreamSets
streamsets.com/opensource

StreamSets Community
streamsets.com/community
Thank You!

Pat Patterson
pat@streamsets.com
@metadaddy