Neo4j - Kafka - MySQL: Configuration - Part 1

With the new Neo4j Kafka streams now available, there has been a few articles such as A New Neo4j Integration with Apache Kafka and How to leverage Neo4j Streams and build a just-in-time data warehouse and Processing Neo4j Transaction Events with KSQL and Kafka Streams and finally How to embrace event-driven graph analytics using Neo4j and Apache Kafka.

In this post, I will discuss configuring a Neo4j cluster that will use the Neo4j Kafka Streams to connect to a Kafka server. I will also talk about configuring Maxwell's Daemon to stream data from MySQL to Kafka and then on to Neo4j.

The new Neo4j Kafka streams library is a Neo4j plugin that you can add to each of your Neo4j instances. It enables three types of Apache Kafka mechanisms:

- Producer: based on the topics set up in the Neo4j configuration file. Outputs to said topics will happen when specified node or relationship types change
- Consumer: based on the topics set up in the Neo4j configuration file. When events for said topics are picked up, the specified Cypher query for each topic will be executed
- Procedure: a direct call in Cypher to publish a given payload to a specified topic

You can get a more detailed overview of how each of these might look like here.

Kafka

For Kafka, I configured an AWS EC2 instance to serve as my Kafka machine. For the setup, I followed the instructions from the quick start guide up until step 2. Before we get Kafka up and running, we will need to set up the consumer elements in the Neo4j configuration files.

If you are using the Dead Letter Queue functionality in the Neo4j Kafka connector, you will have to create that topic. For the MySQL sync topics, the Maxwell Daemon will automatically create those based on the settings in the config.properties file.

MySQL

For MySQL, I set up a simple MySQL server on Ubuntu. A couple of things to note.

- I had to modify the /etc/mysql/my.cnf file and add:

```
[mysqld] server_id=21 log-bin=master binlog_format=row
```

- I had to create a Maxell user.

```
CREATE USER 'maxwell'@'%' IDENTIFIED BY 'YourStrongPassword';  GRANT ALL
ON maxwell.* TO 'maxwell'@'%';  GRANT SELECT, REPLICATION CLIENT, REPLICA
TION SLAVE ON *.* TO 'maxwell'@'%';
```

Maxwell's Daemon

For the sync between MySQL and Kafka, I used Maxwell's daemon, an application that reads MySQL binlogs and writes row updates as JSON to Kafka, Kinesis, or other streaming platforms. Maxwell has low operational overhead, requiring nothing but mysql and a place to write to. Its common use cases include ETL, cache building/expiring, metrics collection, search indexing and inter-service communication. Maxwell gives you some of the benefits of event sourcing without having to re-architect your entire platform.

I downloaded Maxwell's Daemon and installed it on the MySQL server. I then made the following configuration changes in the config.properties file.

```
producer=kafka kafka.bootstrap.servers=xxx.xxx.xxx.xxx:9092 # mysql logi
```

```
n info host=xxx.xxx.xxx.xxx user=maxwell password=YourStrongMaxwellPasswo
rd kafka_topic=blogpost_%{database}_%{table}
```

By configuring the kafka_topic to be linked to the database and the table, Maxwell automatically creates a topic for each table in the database.

Neo4j Cluster

Configuring a Neo4j cluster is covered in the Neo4j Operations Manual.

We will use the Neo4j Streams plugin. As the instructions say, we download the latest release jar from latest and copy it into $NEO4J_HOME/plugins on each of the Neo4j cluster members. Then we will need to do some configuration.

In the Neo4j.conf file, we will need to configure the Neo4j Kafka plugin. This configuration will be the same for all Core servers in the Neo4j cluster. The neo4j.conf configuration is as follows:

```
 ### Neo4j.conf  kafka.zookeeper.connect=xxx.xxx.xxx.xxx:2181 kafka.boots
trap.servers=xxx.xxx.xxx.xxx:9092 streams.sink.enabled=true streams.sink.
polling.interval=1000  streams.sink.topic.cypher.Neo4jPersonTest=MERGE (p
:Person{name: event.name, surname: event.surname}) MERGE (f:Family{name:
event.surname}) MERGE (p)-[:BELONGS_TO]->(f)  streams.sink.topic.cypher.m
usicbrainz_musicbrainz_artist=FOREACH(ignoreMe IN CASE WHEN event.type='i
nsert' THEN [1] ELSE [] END | MERGE (u:Artist{gid:event.data.gid}) on mat
ch set u.id = event.data.id, u.name=event.data.name, u.sort_name=event.da
ta.sort_name on create set u.id = event.data.id, u.name=event.data.name,
u.sort_name=event.data.sort_name) FOREACH(ignoreMe IN CASE WHEN event.typ
e='delete' THEN [1] ELSE [] END | MERGE (u:Artist{gid:event.data.gid})  d
etach delete u) FOREACH(ignoreMe IN CASE WHEN event.type='update' THEN [1
] ELSE [] END | MERGE (u:Artist{gid:event.data.gid})  set u.id = event.da
ta.id, u.name=event.data.name, u.sort_name=event.data.sort_name)  streams
.sink.topic.cypher.musicbrainz_musicbrainz_label=FOREACH(ignoreMe IN CASE
 WHEN event.type='insert' THEN [1] ELSE [] END | MERGE (u:Label{gid:event
.data.gid}) on match set u.id = event.data.id, u.name=event.data.name, u.
```

```
sort_name=event.data.sort_name,u.type=event.data.type on create set u.id
= event.data.id, u.name=event.data.name, u.sort_name=event.data.sort_name
,u.type=event.data.type) FOREACH(ignoreMe IN CASE WHEN event.type='delete
' THEN [1] ELSE [] END | MERGE (u:Label{gid:event.data.gid})  detach dele
te u) FOREACH(ignoreMe IN CASE WHEN event.type='update' THEN [1] ELSE []
END | MERGE (u:Label{gid:event.data.gid})  set u.id = event.data.id, u.na
me=event.data.name, u.sort_name=event.data.sort_name,u.type=event.data.ty
pe)  kafka.auto.offset.reset=earliest kafka.group.id=neo4j streams.sink.d
lq=person-dlq  kafka.acks=all kafka.num.partitions=1 kafka.retries=2 kafk
a.batch.size=16384 kafka.buffer.memory=33554432
```

The Neo4j kafka plug-in will poll to see who the Neo4j cluster leader is. The Neo4j cluster leader will automatically poll the Kafka topics for the data changes. If the cluster leader switches, the new leader will take over the polling and the retrieval from the topics.

Once all of the configuration is completed, I have a running MySQL instance, a Kafka instance, a Maxwell's Daemon configured to read from MySQL and write to Kafka topics and a Neo4j cluster that will read from the Kafka topics.

In part 2, we will show how this all works together.