Hbase Kafka Proxy

1. Use Cases

   There are two primary use cases for this. The first is the need to keep hbase table(s) in sync with an external data store (solr, external data warehouse). Currently, you can use hbase-indexer to keep solr in sync with hbase. However, it is specific to solr. Another case is that many applications only need to know what data has changed in a table. The users will write jobs that will scan a whole table by timestamp just to skim off the updated records.

The aim of this JIRA is to provide a pub-sub interface that allows a variety of downstream applications to listen to the changes that are happening in hbase and react to them.

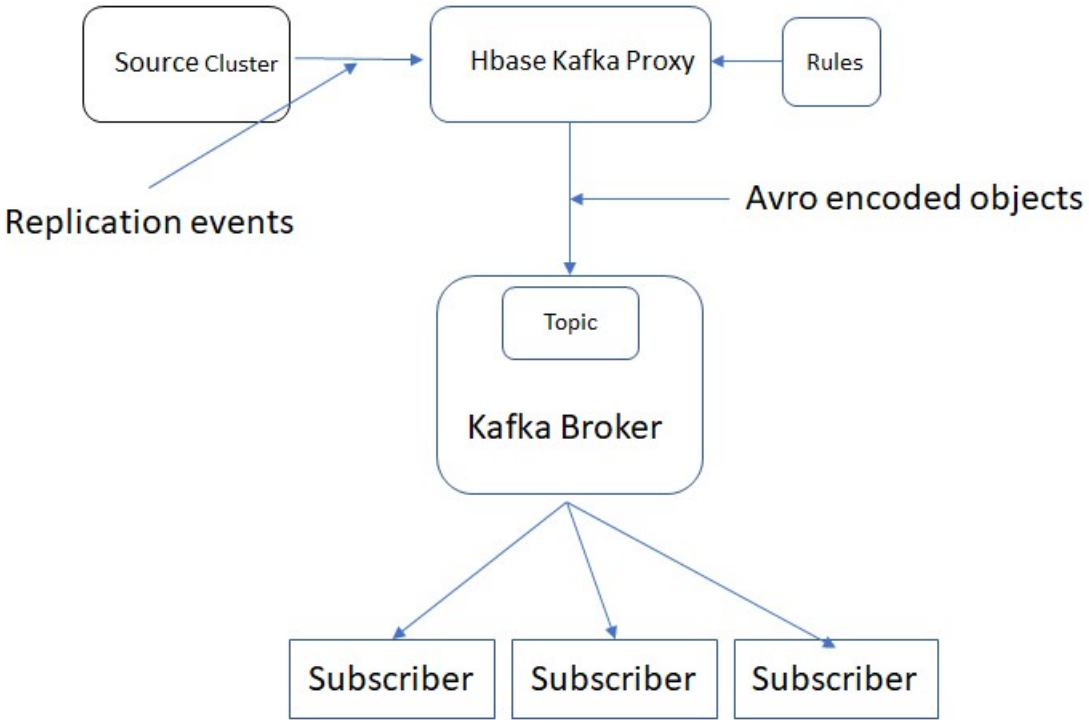
2. Design Goals
   a. Isolation – The code that forwards the mutations should not live in the core region server.
   b. High Performance – The proxy should get mutations into the kafka queue as fast as possible.
   c. Low Downstream Impact – The downstream applications shouldn't require direct hbase dependencies to decode the messages.

3. Existing solutions
   a. Write a job to scan the table and publish the messages.
      i. This works but requires a full table scan which can be expensive and does not allow applications to respond to messages in real time.
   b. SEP processor library – You can create peers using this library to create custom peers.
      i. it's how hbase-indexer works behind the scenes.
      ii. Each new peer requires a new hbase peer.
      iii. The version of the SEP library must match the version of hbase it is peered with.

4. Design Overview

   The approach is to set up a proxy hbase region server (just like the SEP library does) and peer it with the source hbase cluster. Once that happens, the proxy will receive the mutations from the source cluster. If the mutations pass the routing rules they get encoded into Avro format and forwarded to the appropriate topic.

   Making the proxy a pseudo regionserver has two large advantages: fault tolerance and scale. A large hbase cluster could generate a very large amount of traffic that would need to be forwarded to kafka. The current replication system can gracefully handle a slow peer and various error situations. If the proxy or kafka brokers are under duress, they are isolated from the main regions servers and will not affect processing. We can also start one proxy on each worker node of the cluster to allow the system to handle a very large volume of writes.



5. Topic Routing
   Since it may not desirable to not rout all hbase traffic into kafka, the proxy comes with a rules configuration that gives users fine grained control of what gets sent to kafka.

   The rules configuration looks like this:

```
<rules>
  <rule action="route/drop" table="" qualifier="" columnFamily="" topic="if
route, which topic to route the mutation to"/>
</rules>
```

   If you would like to route all mutations from table default:foo to kafka topic fooevent, the rule looks like this:

```
<rules>
  <rule action="route" table="default:foo" topic="fooevent"/>
</rules>
```

   Each time the proxy sees a cell that has table default:foo, it will create a HbaseKafkaEvent object, serialize it to bytes and put it on the kafka topic fooevent.

   If you would like all mutations only for the column family mycf, you would configure the rule like this:

```
<rules>
  <rule action="route" table="default:foo" columnFamily="mycf"
topic="fooevent"/>
```

</rules>

If you would like all mutations only for the column family mycf, except for qualifier noisy, you would configure two rules like this:

<rules>

```
  <rule action="drop" table="default:foo" columnFamily=”mycf” qualifier
=”noisy"/>
  <rule action="route" table="default:foo" columnFamily=”mycf”
topic="fooevent"/>
```

</rules>

The rules are evaluated in the order they appear in the file.

6. Message format

Avro was chosen for several reasons:

- It can handle binary field values without having to bas64 encode them
- Most toolsets handle it natively
- It does not require objects be generated.

Each message is based on one Cell value. Once the messages are pushed to the configured topic the downstream apps use a native avro library to decode the binary messages.

```
{"namespace": "org.apache.hadoop.hbase.kafka",
 "type": "record",
 "name": "HbaseKafkaEvent",
 "fields": [
    {"name": "key", "type": "bytes"},
    {"name": "timestamp",  "type": "long" },
    {"name": "delete",  "type": "boolean" },
    {"name": "value", "type": "bytes"},
    {"name": "qualifier", "type": "bytes"},
    {"name": "family", "type": "bytes"},
    {"name": "table", "type": "bytes"}
 ]
}
```

7. Running the proxy

The proxy is started like this:

bin/hbase kafkaproxy -r <path to file that contains routing rules> -k <kafka brokers> -a

-r is the path to the file that contains the routing rules.

-k is a comma delimited list of kafka brokers

-a will create a peer automatically in the zookeeper that is specified in hbase-site.xml for the source cluster.

-p If you prefer to create your own peer, pass the name of the peer in with the -p option. If you create your own peer, the znode is <zk quorum>/hbasekafkaproxy (instead of /hbase when you peer to a different cluster).