KIP-87 - Add Compaction Tombstone Flag

Status

Current state: *Under Discussion*

Discussion thread: here

JIRA: *here*

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

Motivation

This KIP tries to address the following issues in Kafka.

A log compacted topic is basically a kv-store, so a map API map.put(key, null) is not the same as map.remove(key), which should mean a null value should not represent a delete. a delete should be explicit (meaning flag).

Compaction currently requires a null value for it to mark as a tombstone for deletion. This is covered in KIP-82 as an issue that headers could resolve but is being separated out as is an independent issue in its own right.

This causes issues where organization use message envelopes or wrappers or where a business use case requires delivery of some information on the delete.

This became very apparent on the discussion thread for KIP-82 found *here and here,* where it has so far been consensus that this issue should be separated out and dealt with separately.

Some further use case that came up during this KIP's discussion:

1. Tracability. I would like to know who issued this delete tombstone. It might include the hostname, IP of the producer of the delete.
2. Timestamps. I would like to know when this delete was issued. This use case is already addressed by the availability of per-message timestamps that came in 0.10.0
3. Data provenance. I hope I'm using this phrase correctly, but what I mean is, where did this delete come from? What processing job emitted it? What input to the processing job caused this delete to be produced? For example, if a record in topic A was processed and caused a delete tombstone to be emitted to topic B, I might like the offset of the topic A message to be attached to the topic B message.
4. Distributed tracing for stream topologies. This might be a slight repeat of the above use cases. In the microservices world, we can generate call-graphs of webservices using tools like Zipkin/opentracing.io <http://opentracing.io/>, or something homegrown like https://engineering.linkedin.com/distributed-service-call-graph/real-time-distributed-tracing-website-performance-and-efficiency <https://engineering.linkedin.com/distributed-service-call-graph/real-time-distributed-tracing-website-performance-and-efficiency>. I can imagine that you might want to do something similar for stream processing topologies, where stream processing jobs carry along and forward along a globally unique identifier, and a distributed topology graph is generated.
5. Cases where processing a delete requires data that is not available in the message key. I'm not sure I have a good example of this, though. One hand-wavy example might be where I am publishing documents into Kafka where the documentId is the message key, and the text contents of the document are in the message body. And I have a consuming job that does some analytics on the message body. If that document gets deleted, then the consuming job might need the original message body in order to "delete" that message's impact from the analytics.
6. One potential use case is for schema registration. For example, in Avro, a null value corresponds to a Null schema. So, if you want to be able to keep the schema id in a delete message, the value can't be null

Public Interfaces

This KIP has the following public interface changes:

1. Use an attribute bit as a tombstone flag on the core message format.
2. This "tombstone" attribute bit is only used by the broker when a topic is configured for compaction.

    a. If the topic is not configured for compaction it will be set and on the record and available to be read but will not cause message deletion by the broker.
    b. This will need to be added to end user documention.
3. Add a tombstone boolean field to ProducerRecord and ConsumerRecord.
4. Add constructor param on ProducerRecord to set the tombstone on creation of a Record
5. Add accessor methods on the Producer/ConsumerRecord boolean isTombstone()
6. Add ProduceRequest/ProduceResponse V3 which uses the new message format.
7. Add FetchRequest/FetchResponse V3 which uses the new message format.

For more detail information of the above changes, please refer to the Proposed Changes section.

Proposed Changes

Wire protocol change - use attribute bit5 as flag for tombstone boolean flag.

- The attributes flag bit is used to keep the message size the same as before.

```
MessageAndOffset => Offset MessageSize Message
  Offset => int64
  MessageSize => int32

  Message => Crc MagicByte Attributes Timestamp KeyLength Key ValueLength Value
    Crc => int32
    MagicByte => int8  <---------------------- Bump the Magic Byte to 2
    Attributes => int8 <---------------------- Use Bit 5 as boolean flag for 'isTombstone' flag
    Timestamp => int64
    KeyLength => int32
    Key => bytes
    ValueLength => int32
    Value => bytes
```

LogCleaner

Update method "shouldRetainMessage"

- If the magic byte on message is 0, the broker should use the null value for log compaction.
- If the magic byte on message is 1, the broker should use the null value for log compaction.
- If the magic byte on message is 2, the broker should use the tombstone bit for log compaction.

This will be done at the per message level.

Compatibility, Deprecation, and Migration Plan

- **Migration plan :**
    - Currently the code version is 0.10.1, message.format.version = 0.10.1, IBP = 0.10.1.
    - Create internal ApiVersion 0.10.2-IV0 which uses ProduceRequest V3 (magic byte = 2) and FetchRequest V3 (magic byte =2).
    - We first upgrade the code to support ApiVersion 0.10.2-IV0 with a rolling upgrade.
    - Do a rolling bounce and set the IBP = 0.10.2
    - Upgrade clients to start producing and consuming using ProduceRequest V3 (magic byte = 2) and FetchRequest V3 (magic byte =2).
        - At this point if the broker receives ProduceRequest V3 from producer with the tombstone bit set in the message, it will down convert it to set it to null. ProduceRequest V2 will work as it does today.
        - At this point if the broker receives FetchRequest V2 from consumer, we will not loose zero copy because the message is down converted. If the broker receives FetchRequest V3, it will still work as the consumer will be able to understand the tombstone bit.
    - When all the clients have upgraded, bump up the message.format.version to 0.10.2 on the broker. We should be careful here to see that almost all consumers are upgraded since at this point if broker receives a FetchRequest V2, we might loose zero copy on the broker.
        - To note on a downgrade for an non-upgraded consumer, the downgrade would make the value null for a tombstone message.
    - From log compaction point of view :
        - If the magic byte on message is 0, the broker should use the null value for log compaction.
        - If the magic byte on message is 1, the broker should use the null value for log compaction.
        - If the magic byte on message is 2, the broker should use the tombstone bit for log compaction.
    - **NOTE** : With the new version of producer client using ProduceRequest V3 (magic byte = 2), a non tombstone (tombstone bit not set) and null value should not be allowed as the older version of consumer using FetchRequest V2 will think of this as a tombstone when its actually not.

- **Client Compatibility:**
  - *If simply upgrading client, but no changes to code/flow (e.g. still sending null tombstones via existing constructors), we do not expect any app changes needed.*
  - **Producer**
    - A new constructors for ProducerRecord will add an extra parameter tombstone:
    - The old constructors for ProducerRecord without this parameter will be kept but updated so that their default behaviour if setting null value will be the tombstone will be set to true to keep existing behaviour.
  - **Consumer**
    - The ConsumerRecord will expose a new method isTombstone will be exposed
    - Existing flows that use null tombstones and continue to do so will not be affected as the value for a tombstone will still be null
      - As such simply upgrading clients and not the flow behaviour there will be node change needed.
      - It is recommended that their logic is updated to use the isTombstone for clean-ness.
    - New/Amended flows created where a producer may send non-null tombstones, the new consumer code will need to use isTombstone.

Rejected Alternatives

Do nothing KIP-82 would remove the immediate issue.

Originally it was proposed in KIP-82 - Add Record Headers that by supporting headers would resolve this issue as then there would be no need for message wrappers as one of the issues being flagged up in that discussion thread.

As per its discussion it got agreed that we should address the compaction based on null value seperately as discussed *here, a*s it is cleaner to be explicit than still relying on a null payload even after header support. Relying on a null payload only was agreed a bad design decision made at that time.

Use Header field to be explicit based on KIP-82 Headers Proposal

Intent here is to use a header e.g. key = 5 (compaction tombstone) value type=boolean to make an explicit flag that can be used.

Advantages

- Avoids any further changes to the core message protocol , this is one of the intents of KIP-82 to be able to add additional platform level information without constant protocol changes
- Avoids using the attributes flags which are finite in size

Disadvantages

- Risk of the issues this KIP address being dependant on KIP-82, and as KIP-82 is more contentious risks not dealing with the immediate issue for the next release
  - This could always be re-worked in future to use a header in a future release.

Note: If KIP-82 gets agreed on and merged and solution agreed upon lends itself to server side being header aware (aka not a client only wrapper) in time, we should re-evaluate this solution instead of the current proposed