KIP-791: Add Record Metadata to State Store Context

Status

Current state: Accepted

Discussion thread: here

Vote thread: https://lists.apache.org/thread/4krwr1p9h71t3qs3kpjr5j1gnomcsn63

JIRA: here

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

Motivation

In order to provide stronger consistency across replicated state stores (e.g., Read-Your-Writes consistency) it's evident for state stores to collect metadata (e.g., offset information) of all the records that have been written to the store.

State stores access processor context via the StateStoreContext interface. Today, the AbstractProcessorContext class which implements/extends StateStoreContext already makes record metadata available via recordMetadata(). Unfortunately that method is not part of the StateStoreContext interface, thus, state stores cannot access it.

Public Interfaces

In this KIP we propose to add recordMetada() via the StateStoreContext. The following interface will change:

- org.apache.kafka.stream.processor.StateStoreContext - adding method recordMetadata()

Proposed Changes

The code segment below shows the actual code change together with the corresponding JavaDoc:

```
public interface StateStoreContext {
    ...

    /**
     * Return the metadata of the current topic/partition/offset if available.
     * This metadata in a StateStore is defined as the metadata of the record
     * that is currently been processed by the StreamTask that holds the store.
     * <p>
     * Note that the metadata is not defined during all store interactions, for
     * example, while the StreamTask is running a punctuation.
     */
      Optional<RecordMetadata> recordMetadata();

    ...
}
```

Note that recordMetadata() returns an Optional which accounts for the fact that there might not be any recordMetadata available if no records have been processed yet.

The following code snippet illustrates how the above interface can be used within the state store to keep track of the latest offset a store has processed:

```
public class RocksDBStore implements KeyValueStore<Bytes, byte[]>, BatchWritingStore {
     private StateStoreContext context;
     private Map<String, Map<Integer, Long>> seenOffsets = new HashMap<>();
         ...

     public synchronized void put(final Bytes key,
        final byte[] value) {
        Objects.requireNonNull(key, "key cannot be null");
        validateStoreOpen();
        dbAccessor.put(key.get(), value);

        if (context != null && context.recordMetadata().isPresent()) {
            final RecordMetadata meta = context.recordMetadata().get();
            //update seen offsets
            seenOffsets.computeIfAbsent(meta.topic(), t -> new HashMap<>())
                .put(meta.partition(), meta.offset());
        }*/
    }

        ...
}
```

Compatibility, Deprecation, and Migration Plan

The change of the StateStoreContext interface does not require any changes to any of the implementations of the interface, it merely exposes a method (recordMetadata) that is already implemented in AbstractProcessorContext. Therefore, we do not expect any compatibility issues. Moreover, recordMetadata returns an object of type RecordMetadata which is read-only, thus, protecting Kafka-internals from being tampered by applications.

Rejected Alternatives

Add a new Put method to the KeyValueStore interface

The idea would be to couple data and metadata together in a single Put call.

```
public interface KeyValueStore<K, V> extends StateStore, ReadOnlyKeyValueStore<K, V> {
        ...

    //new put including metadata
        void put(K key, V value, RecordMetadata metadata);

    //we keep old put for cases where there is no metadata

        ...
}
```

We rejected this approach because of the extra code complexity it causes. Each processor would have to explicitly manage which call (`put(key, value)` or `put(key, value, metadata)`) to invoke, depending on whether it's running in process or in punctuate. Then, the caching layer would also need to select which put call to call on the lower stores, based on whether the cached metadata is present or not.

Add a new UpdatePosition method to the KeyValueStore interface

```
public interface KeyValueStore<K, V> extends StateStore, ReadOnlyKeyValueStore<K, V> {
        ...


        void updatePosition(String topic, int partition, int offset);

        ...
}
```

We rejected this approach because of an increased risk for bugs. For instance, if a processor forgets to call update after a put it will be impossible for a state store to reason about consistency. Second, decoupling the data and the metadata update is also problematic because it creates a window of inconsistency where the data is already reflected by the store but the metadata is not.