KIP-634: Complementary support for headers and record metadata in Kafka Streams DSL

Status

Current state: Dormant/Inactive

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

Record metadata values (e.g. topic name, partition, offset, timestamp) are accessible at the Processor API, including headers (KIP-244).

Using these values for common stateless tasks like:

- filtering/branching based on topic name,
- logging incoming partition number/offset number,
- adding/modifying headers,

is not straightforward as it involves mixing Processors and DSL operators.

But also, using these values in more complex computations already available in DSL as joins/aggregations is more complex, as it will require reimplementing joins/aggregations with custom Processor. More info: https://issues.apache.org/jira/browse/KAFKA-7718

This KIP proposes to include operators to make record values accessible at the DSL level, allow using these values in stateful operations, and modifying underlying record headers.

Public Interfaces

In accordance with KStreams DSL Grammar, we introduce the following new elements to the KStream API:

- `KStream.mapValueToRecord` DSLOperation
- `KStream.setRecordHeaders` DSLOperation with the following operand
    - `RecordHeadersMapper` DSLObject
        - `Headers get(final K key, final V value)`

Apart from these changes, new public interfaces are added:

- `o.a.k.s.kstream.RecordSerde<K,V>` for serializing/deserializing `Record<K, V>` values with the following binary structure:
    - `varint keySize`
    - `bytes key`
    - `varint valueSize`
    - `bytes value`
    - `varint topicNameSize`
    - `bytes topicName`
    - `varint partition`
    - `varlong offset`
    - `varlong timestamp`
    - `varint numHeaders`
    - `Header[]`:
        - `varint headerKeySize`
        - `bytes headerKey`
        - `varint headerValueSize`
        - `bytes headerValue`
- `o.a.k.s.processor.api.header.Headers` (inspired on Connect Headers) implemented by a class `o.a.k.s.header.StreamHeaders`:

        - `int size()`
        - `boolean isEmpty()`
        - `Iterator<Header> allWithName(String key)`
        - `Header lastWithName(String key)`
        - `boolean hasWithName(String key)`
        - `Headers add(Header header)`
        - `Headers add(String key, byte[] value)`
        - `Headers addUtf8(String key, String value)`
        - `Headers remove(String key)`
        - `Headers retainLatest(String key)`
        - `Headers retainLatest()`
        - `Headers clear()`
        - `Headers duplicate()`
        - `Headers apply(Headers.HeaderTransform transform)`
        - `Headers apply(String key, Headers.HeaderTransform transform)`
        - `functional interface HeaderTransform { Header apply(Header header); }`
        - `o.a.k.common.header.Headers unwrap()`
- `o.a.k.s.header.Header` implemented by a class `o.a.k.s.header.StreamHeader`
    - `String key`
    - `byte[] value`

And the following existing APIs will be expanded:

- `o.a.k.s.processor.api.Record` now implements `RecordMetadata`:
    - Fields:
        - New: `String topic`, `int partition`, `long offset`
        - Change: `headers` type is now `o.a.k.s.processor.api.headers.Headers`
    - Constructors:
        - `Record(K key, V value, long timestamp, Headers headers, String topic, int partition, long offset) {`
        - `Record(K key, V value, long timestamp, org.apache.kafka.common.header.Headers headers, String topic, int partition, long offset)`
        - `Record(K key, V value, long timestamp, org.apache.kafka.common.header.Header[] headers, String topic, int partition, long offset)`
        - `Record(K key, V value, long timestamp, Headers headers, Optional<RecordMetadata> recordMetadata)`
        - `Record(K key, V value, long timestamp, org.apache.kafka.common.header.Headers headers, Optional<RecordMetadata> recordMetadata)`
        - `Record(K key, V value, long timestamp, org.apache.kafka.common.header.Header[] headers, Optional<RecordMetadata> recordMetadata)`
        - `Record(K key, V value, long timestamp, org.apache.kafka.common.header.Headers headers)`
        - `Record(K key, V value, long timestamp, org.apache.kafka.common.header.Header[] headers)`
        - `Record(K key, V value, long timestamp, Headers headers)`
    - New methods:
        - `String topic()`
        - `int partition()`
        - `long offset()`
        - `Optional<RecordMetadata> recordMetadata()`
- `o.a.k.s.processor.To`:
    - Fields:
        - New: `org.apache.kafka.common.header.Headers`
    - Constructors:
        - `To(String childName, long timestamp, Headers headers)`
    - New methods:
        - `To withHeaders(Headers headers)`

Description

1. `KStream#mapValueToRecord(Named)` operation exposes the new `Record<K, V>` type including headers, and topic metadata. The overloaded parameterless alternative `mapValueToRecord()` is also available.
2. `RecordSerde<K, V>` is a public API, and it's implicitly defined as `valueSerde` when the `KStream#mapValueToRecord` operation is called.
3. `KStream#setRecordHeaders(RecordHeaderMapper, Named)` operation will “flush” headers into the actual record's headers crossing the stream operation, to be used by consumers downstream. This mapper function receives `K` key and `V` value, and return a `o.a.k.s.header.Headers`. Users can create new Headers using the Streams' implementation `o.a.k.s.header.StreamHeaders`, or using existing ones by previously using `KStreams#mapValueToRecord()`. The overloaded parameterless alternative `setRecordHeaders(RecordHeaderMapper)` is also available.

Usage examples

- Filter records based on the topic name:

```
builder
  .stream(List.of("input","another"), Consumed.with(Serdes.String(),Serdes.String()))
  .mapValueToRecord() // 1. map record metadata
  .split() // 2. branch by topic name
  .branch((key, value) -> value.topic().equals("input"), Branched.withConsumer(b1 ->{ /*...*/ }))
  .noDefaultBranch();
```

- Filter records based on if a header exists and its value:

```
b1
  .mapValueToRecord()
  //...
  .filter((key, value) -> value.headers().hasWithName("k"))
  .filter((key, value) -> "v".equals(value.headers().lastWithName("k").valueAsUtf8()))
```

- Apply headers to underlying record:

```
b1
  .mapValueToRecord()
  //...
  .setRecordHeaders((k, v) -> v.headers().addUtf8("k1", "v1").retainLatest())
```

- Use Headers in stateful aggregations:

```
b1
  .mapValueToRecord()
  //...
  .groupByKey()
  .reduce((value1, value2) -> {
    value1.headers().forEach(header -> value2.headers().add(header));
    return value2;
  }, Materialized.with(Serdes.String(), new RecordSerde<>(Serdes.String(), Serdes.String())));
```

Proposed Changes

- `KStream`:

```
KStream<K, Record<K, V>> mapValueToRecord(final Named named);

KStream<K, Record<K, V>> mapValueToRecord();

KStream<K, V> setRecordHeaders(final RecordHeadersMapper<? super K, ? super V> action, final Named named);

KStream<K, V> setRecordHeaders(final RecordHeadersMapper<? super K, ? super V> action);
```

- `RecordHeadersMapper`:

```
package org.apache.kafka.streams.kstream;

import org.apache.kafka.streams.header.Headers;

public interface RecordHeadersMapper<K, V> {
    Headers get(final K key, final V value);
}
```

- Add the following Header interfaces with its implementations (`StreamHeader(s)`):

```
package org.apache.kafka.streams.processor.api.header;

public interface Header {
    String key();
    byte[] value();
    String valueAsUtf8();
}
```

```
package org.apache.kafka.streams.processor.api.header;

import java.util.Iterator;

public interface Headers extends Iterable<Header> {

    // State validation
    int size();
    boolean isEmpty();

    // Access headers
    Iterator<Header> allWithName(final String key);
    Header lastWithName(final String key);
    boolean hasWithName(final String key);

    // Add/delete/clean
    Headers add(final Header header);
    Headers add(final String key, final byte[] value);
    Headers addUtf8(final String key, final String value);
    Headers remove(final String key);
    Headers retainLatest(final String key);
    Headers retainLatest();
    Headers clear();
    Headers duplicate();

    // Transformations
    Headers apply(final Headers.HeaderTransform transform);
    Headers apply(final String key, final Headers.HeaderTransform transform);

    interface HeaderTransform {
        Header apply(final Header header);
    }

    // to core Headers
    org.apache.kafka.common.header.Headers unwrap();
}
```

4. Modify `Record<K, V>`

```
package org.apache.kafka.streams.processor.api;

import java.util.Optional;
import org.apache.kafka.streams.errors.StreamsException;

import java.util.Objects;
import org.apache.kafka.streams.processor.api.header.Header;
import org.apache.kafka.streams.processor.api.header.Headers;
import org.apache.kafka.streams.processor.api.header.StreamHeaders;

public class Record<K, V> implements RecordMetadata {
    public static final Header[] EMPTY_HEADERS = new Header[0];

    private final K key;
    private final V value;
    private final long timestamp;
    private final Headers headers;
```

```
private final String topic;
private final int partition;
private final long offset;

public Record(final K key, final V value,
    final long timestamp,
    final Headers headers,
    final String topic, final int partition, final long offset) {
}

public Record(final K key, final V value,
    final long timestamp,
    final org.apache.kafka.common.header.Headers headers,
    final String topic, final int partition, final long offset) {

    this(key, value, timestamp, StreamHeaders.wrap(headers), topic, partition, offset);
}

public Record(final K key, final V value,
    final long timestamp,
    final org.apache.kafka.common.header.Header[] headers,
    final String topic, final int partition, final long offset) {

    this(key, value, timestamp, StreamHeaders.wrap(headers), topic, partition, offset);
}

public Record(final K key, final V value,
    final long timestamp,
    final org.apache.kafka.common.header.Header[] headers,
    final Optional<RecordMetadata> recordMetadata) {

    this(key, value, timestamp, StreamHeaders.wrap(headers),
        recordMetadata.map(RecordMetadata::topic).orElse(null),
        recordMetadata.map(RecordMetadata::partition).orElse(-1),
        recordMetadata.map(RecordMetadata::offset).orElse(-1L));
}

public Record(final K key, final V value,
    final long timestamp,
    final org.apache.kafka.common.header.Headers headers,
    final Optional<RecordMetadata> recordMetadata) {

    this(key, value, timestamp, StreamHeaders.wrap(headers),
        recordMetadata.map(RecordMetadata::topic).orElse(null),
        recordMetadata.map(RecordMetadata::partition).orElse(-1),
        recordMetadata.map(RecordMetadata::offset).orElse(-1L));
}

public Record(final K key, final V value,
    final long timestamp,
    final Headers headers,
    final Optional<RecordMetadata> recordMetadata) {

    this(key, value, timestamp, headers,
        recordMetadata.map(RecordMetadata::topic).orElse(null),
        recordMetadata.map(RecordMetadata::partition).orElse(-1),
        recordMetadata.map(RecordMetadata::offset).orElse(-1L));
}

public Record(final K key, final V value,
    final long timestamp,
    final org.apache.kafka.common.header.Header[] headers) {

    this(key, value, timestamp, StreamHeaders.wrap(headers), Optional.empty());
}

public Record(final K key, final V value,
    final long timestamp,
    final org.apache.kafka.common.header.Headers headers) {

    this(key, value, timestamp, StreamHeaders.wrap(headers), Optional.empty());
```

```
}

public Record(final K key, final V value,
    final long timestamp,
    final Headers headers) {

    this(key, value, timestamp, headers, Optional.empty());
}

public Record(final K key, final V value, final long timestamp) {
    this(key, value, timestamp, org.apache.kafka.common.record.Record.EMPTY_HEADERS);
}

public K key() {
    return key;
}

public V value() {
    return value;
}

public long timestamp() {
    return timestamp;
}

public Headers headers() {
    return headers;
}

public <NewK> Record<NewK, V> withKey(final NewK key) {
    return new Record<>(key, value, timestamp, headers, recordMetadata());
}

public <NewV> Record<K, NewV> withValue(final NewV value) {
    return new Record<>(key, value, timestamp, headers, recordMetadata());
}

public Record<K, V> withTimestamp(final long timestamp) {
    return new Record<>(key, value, timestamp, headers, recordMetadata());
}

public Record<K, V> withHeaders(final Headers headers) {
    return new Record<>(key, value, timestamp, headers, recordMetadata());
}

public Record<K, V> withHeaders(final org.apache.kafka.common.header.Headers headers) {
    return new Record<>(key, value, timestamp, headers, recordMetadata());
}

@Override
public String topic() {
    return topic;
}

@Override
public int partition() {
    return partition;
}

@Override
public long offset() {
    return offset;
}

Optional<RecordMetadata> recordMetadata() {
    if (topic != null) {
        return Optional.of(new RecordMetadata() {
            @Override
            public String topic() {
                return topic;
            }
```

```
                @Override
                public int partition() {
                    return partition;
                }

                @Override
                public long offset() {
                    return offset;
                }
            });
        } else  {
            return Optional.empty();
        }
    }
}
```

- New `RecordSerde<K, V>`:

```
package org.apache.kafka.streams.kstream;

public class RecordSerde<K, V> implements Serde<Record<K, V>> {

    final Serde<K> keySerde;
    final Serde<V> valueSerde;

    public static <K, V> RecordSerde<K, V> with(final Serde<K> keySerde, final Serde<V> valueSerde) {
        return new RecordSerde<>(keySerde, valueSerde);
    }

    RecordSerde(final Serde<K> keySerde, final Serde<V> valueSerde) {
        this.keySerde = keySerde;
        this.valueSerde = valueSerde;
    }

    @Override
    public Serializer<Record<K, V>> serializer() {
        return new RecordSerializer<>(keySerde.serializer(), valueSerde.serializer());
    }

    @Override
    public Deserializer<Record<K, V>> deserializer() {
        return new RecordDeserializer<>(keySerde.deserializer(), valueSerde.deserializer());
    }

    static class RecordSerializer<K, V> implements Serializer<Record<K, V>> {

        final Serializer<K> keySerializer;
        final Serializer<V> valueSerializer;

        RecordSerializer(final Serializer<K> keySerializer, final Serializer<V> valueSerializer) {
            this.keySerializer = keySerializer;
            this.valueSerializer = valueSerializer;
        }

        @Override
        public byte[] serialize(final String topic, final Record<K, V> data) {
            // implementation
        }
    }

    static class RecordDeserializer<K, V> implements Deserializer<Record<K, V>> {

        final Deserializer<K> keyDeserializer;
        final Deserializer<V> valueDeserializer;

        RecordDeserializer(final Deserializer<K> keyDeserializer, final Deserializer<V> valueDeserializer) {
            this.keyDeserializer = keyDeserializer;
            this.valueDeserializer = valueDeserializer;
        }

        @Override
        public Record<K, V> deserialize(final String t, final byte[] data) {
            // implementation
        }
    }
}
```

Compatibility, Deprecation, and Migration Plan

- If users have an existing stateful operator and add `mapValueToRecord` before this operator, will change the value to `Record<K, V>` , causing an incompatible topology change.
- `To` API will be extended to support headers and be backwards compatible.

- `KStreamSetRecordHeaders` and `KStreamMapValueToRecord` are both internal APIs, not exposed to users. Both will be implemented using the latest `Processor` API from `KIP-478`.

Rejected Alternatives

- Expand `KeyValue` to support headers. This will affect all current APIs, from KStream/KTable to Stores.
- Adding `mergeHeaders` functions to join/aggregation. Although this will extend support for headers, will add complexity to existing functions.
- (initial version of this KIP) Add Header-specific methods to the DSL (e.g. `withHeaders`, `addHeader`, `removeHeaders`). Although this will allow accessing and manipulating headers from DSL, it will have a high impact on the existing KStream API (too many methods) and only specific for Headers. Also, it will require dealing with the same abstraction as Kafka Records. Will require more methods to cover other metadata.
- Include a new value (e.g. `RecordValue<V>`) to load headers and metadata to the value. Instead, leverage and extend existing `Record<K,V>` type.

References

- Draft implementation: https://github.com/apache/kafka/compare/trunk...jeqo:to-record