KIP-121: Add KStream peek method

- Status
- Motivation
- Public Interfaces
- Proposed Changes
- Compatibility, Deprecation, and Migration Plan
- Rejected Alternatives

Status

Current state: *Vote Passed*

Discussion thread: *here*

JIRA: *KAFKA-4720*

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

Motivation

KStream looks a lot like `java.util.stream.Stream`; it includes methods such as `filter` and `map`, terminal operations like `to`, and so on. One conspicuously absent method is `peek`.

Most stream operations are expected to be pure functions. `peek`, by its nature, is not. Usually mutating external state is discouraged but a number of diagnostic activities are made much easier.

For example, you might want to write:

```
KStream<String> words = someWordsWeWantToCount(); // since this is what basically every kafka application does,
right? :)
words.filter((w, v) -> w.startsWith("a"))
     .peek(w -> metrics.wordsStartingWithAProcessed.increment())
     .peek(System.out::println)
     .groupByKey()
     .count();
```

In this particular case, we increment a metric counter (understanding that in failure cases some records may be double-counted due to retried work) and print every word observed out to `System.out`. This provides useful information **about the processing of the stream** as opposed to the **contents of the stream**, which can be very useful when experimenting with stateful processors or otherwise diagnosing things that went wrong.

Additionally, the JDK is known for a high bar to include methods in the base class library. `Stream#peek` made that cut. So that is pretty good evidence that some really smart Java API designers agree that it holds its weight.

Public Interfaces

Adds `KStream#peek(ForeachAction<K,V>)`

Proposed Changes

A straightforward first pass is on GitHub PR #2493.

Compatibility, Deprecation, and Migration Plan

No compatibility issues forseen.

Rejected Alternatives

`peek` by its nature may be implemented as a simple transformation of `filter` or `map`. Unfortunately this is the wrong interface (as you need to return the same value passed in for the functional interface contract) and loses some semantic information that could be used for a potential future optimizer (that the stream is not modified by the operation).