KIP-962: Relax non-null key requirement in Kafka Streams

- Motivation
- Public Interfaces / Proposed Changes
  - Operators
  - Repartition of null-key records
- Compatibility, Deprecation, and Migration Plan
  - Keeping the old behavior
    - Examples
    - Remarks
  - Why not make this change opt in?

Current state: Merged
Voting thread: here
Discussion thread: here

JIRA: [KAFKA-12317](#) - Getting issue details... [STATUS]  [KAFKA-14748](#) - Getting issue details... [STATUS]

Motivation
Kafka Streams enforces a strict non-null-key policy in the DSL across all key-dependent operations (like aggregations and joins). For left-joins, it makes sense to still accept a ‘null’, and add the left-hand record with an empty right-hand-side to the result. Similarly, for outer-joins it makes sense to keep the record. The defined semantics for left/outer join are: keep the stream record if no matching join record was found. Thus, Kafka Streams today’s behavior is inconsistent with the defined semantics.

Public Interfaces / Proposed Changes
This KIP won’t change any public interfaces. However, it will change the behavior of the following:

Operators
- `left join` Kstream-Kstream: no longer drop left records with null-key and call ValueJoiner with ‘null’ for right value.
- `outer join` Kstream-Kstream: no longer drop left/right records with null-key and call ValueJoiner with ‘null’ for right/left value.
- `left-foreign-key join` Ktable-Ktable: no longer drop left records with null-foreign-key returned by the ForeignKeyExtractor and call ValueJoiner with ‘null’ for right value.
- `left join` KStream-Ktable: no longer drop left records with null-key and call ValueJoiner with ‘null’ for right value.
- `left join` KStream-GlobalTable: no longer drop records when KeyValueMapper returns ‘null’ and call ValueJoiner with ‘null’ for right value.

Repartition of null-key records
Currently, repartitioning causes records with ‘null’-keys to be dropped. This behavior needs to be changed. The above described change in behavior would otherwise not work for topologies with repartitioning. Going forward, they will be sent to an arbitrary partition.

Compatibility, Deprecation, and Migration Plan
Keeping the old behavior
Users who want to keep the current behavior can prepend a .filter() operator to the aforementioned operators and filter accordingly.

Examples
//left join Kstream-Kstream
leftStream
  .filter((key, value) -> key != null)
  .leftJoin(rightStream, (lv, rv) -> join(lv, rv), windows);

//outer join Kstream-Kstream
rightStream
  .filter((key, value) -> key != null);
leftStream
  .filter((key, value) -> key != null)
  .outerJoin(rightStream, (lv, rv) -> join(lv, rv), windows);

//left-foreign-key join Ktable-Ktable
Function<String, String> foreignKeyExtractor = leftValue -> ...
leftTable
  .filter((key, value) -> foreignKeyExtractor.apply(value) != null)
  .leftJoin(rightTable, foreignKeyExtractor, (lv, rv) -> join(lv, rv), Named.as("left-foreign-key-table-join"));

//left join Kstream-Ktable
leftStream
  .filter((key, value) -> key != null)
  .leftJoin(ktable, (k, lv, rv) -> join(lv, rv));

//left join KStream-GlobalTable
KeyValueMapper<String, String, String> keyValueMapper = (k, v) -> ...
leftStream
  .filter((key, value) -> keyValueMapper.apply(key,value) != null)
  .leftJoin(globalTable, keyValueMapper, (lv, rv) -> join(lv, rv));

Remarks

Note that the above list of operator examples is not exhaustive.
E.g. There is only one example with a 'ValueJoinerWithKey'. All other examples pass a 'ValueJoiner' into the operator.

We will hint to the DSL users via Java docs that they have now have the option to distinguish between the following scenarios within a KTable-KTable foreign-key-left-join operator:
"null-key" v.s. "not-null-key but null-value" by passing a 'ValueJoinerWithKey' instead of 'ValueJoiner' to the left join operator.
The remark will be made in the Java docs of all left join methods with a 'ValueJoiner' as a parameter.

As part of this KIP the above information on how to keep the old behavior will also be documented here: https://kafka.apache.org/documentation/streams/upgrade-guide

Why not make this change opt in?

By default we would like the behavior of a stream defined via DSL to be consistent with the defined semantics.