KIP-364: Remove implicit Materialized, Consumed and Produced

Status

Current state: *Subsumed by KIP-365*

Discussion thread: here

Github PR: here

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

Motivation

I've proposed when Kafka Stream's Scala API was not released yet to solve the serdes issue on functions that takes a materialized (count, reduce and aggregate) to just make the materialized implicit and have a default implicit in ImplicitConversions that had the serdes filled in.

Given the following definitions:
```
def count()(implicit materialized: Materialized[K, Long, ByteArrayKeyValueStore])
def reduce(reducer: (V, V) => V)(implicit materialized: Materialized[K, V, ByteArrayKeyValueStore])
def aggregate[VR](initializer: => VR)(aggregator: (K, V, VR) => VR)(implicit materialized: Materialized[K, VR, ByteArrayKeyValueStore])
```

We can call them without Materialized:
```
groupedStream.count()
```

Or with a materialized:
```
groupedStream.count()(Materialized.as("store-name"))
```

By doing that we solved the case when no Materialized is given, setting implicitly the serdes to avoid runtime errors but still allowing the ability to give explicitly our own Materialized.

The issue with this solution are the followings:

- Making Materialized implicit suggests that a Materialized is a context object whereas it's not.
- The user of the API will by default just not care about implicit parameters and assume that they are provided.
- The user of the API might declare it's own Materialized in an implicit val since the parameter is implicit and then be caught by other unwanted functions that also takes Materialized as implicit.
- This a long shot view but in Scala 3 giving implicits explicitly looks like:
  ```
  groupedStream.count().explicitly(Materialized.as("store-name"))
  ```
  So it's clear that implicits should not be used as a defaulting system.

All of that applies to Consumed and Produced too.

Public Interfaces

```
org.apache.kafka.streams.scala.kstream.KGroupedStream.count()
org.apache.kafka.streams.scala.kstream.KGroupedStream.reduce()
org.apache.kafka.streams.scala.kstream.KGroupedStream.aggregate()
org.apache.kafka.streams.scala.kstream.KGroupedTable.count()
org.apache.kafka.streams.scala.kstream.KGroupedTable.reduce()
org.apache.kafka.streams.scala.kstream.KGroupedTable.aggregate()
org.apache.kafka.streams.scala.ImplicitConversions.materializedFromSerde()
org.apache.kafka.streams.scala.StreamsBuilder.stream()
org.apache.kafka.streams.scala.StreamsBuilder.table()
org.apache.kafka.streams.scala.StreamsBuilder.globalTable()
```

Proposed Changes

```
def count()(implicit keySerde: Serde[K], valueSerde: Serde[V])
def count(implicit materialized: Materialized[K, Long, ByteArrayKeyValueStore])

def reduce(reducer: (V, V) => V)(implicit keySerde: Serde[K], valueSerde: Serde[V])
def reduce(reducer: (V, V) => V, materialized: Materialized[K, V, ByteArrayKeyValueStore])


def aggregate[VR](initializer: => VR)(aggregator: (K, V, VR) => VR)(implicit keySerde: Serde[K], valueSerde: Serde[V])
def aggregate[VR](materialized: Materialized[K, VR, ByteArrayKeyValueStore], initializer: => VR)(aggregator: (K, V, VR) => VR)

def stream[K, V](topic: String)(implicit keySerde: Serde[K], valueSerde: Serde[V])
def stream[K, V](topic: String, consumed: Consumed[K, V])

def table[K, V](topic: String)(implicit keySerde: Serde[K], valueSerde: Serde[V])
def table[K, V](topic: String, consumed: Consumed[K, V])

def globalTable[K, V](topic: String)(implicit keySerde: Serde[K], valueSerde: Serde[V])
def globalTable[K, V](topic: String, consumed: Consumed[K, V])

def through(topic: String)(implicit keySerde: Serde[K], valueSerde: Serde[V])
def through(topic: String, produced: Produced[K, V])

def to(topic: String)(implicit keySerde: Serde[K], valueSerde: Serde[V])
def to(topic: String, produced: Produced[K, V])

def to(extractor: TopicNameExtractor[K, V])(implicit keySerde: Serde[K], valueSerde: Serde[V])
def to(extractor: TopicNameExtractor[K, V], produced: Produced[K, V])
```

This way we require only implicit Serdes if the Materialized (or Consumed, or Produced) is not given explicitly.

Compatibility, Deprecation, and Migration Plan

TBD

Rejected Alternatives

NA