KIP-957: Support Async runtimes

Status

Current state: Withdrawn, this should have been included in KIP-944.

Discussion thread: here

JIRA: KAFKA-14972

Proposed implementation: pull request 14071

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

Design goal

The goal of this KIP is to allow consumer callbacks to call the consumer again from another thread.

Motivation

Rebalances cause a lot of message duplication. This can be prevented by doing commits in the partition-revoked callback. This KIP will make it much easier to do work in that callback when an async runtime is used.

The JVM based KafkaConsumer contains a check that rejects nested invocations from different threads (in method acquire). For programs that use an async runtime, this is an almost impossible requirement. Also, the check is more strict than is required; we only need to validate that there is no concurrent access to the consumer.

Examples of affected async runtimes are Kotlin co-routines (see KAFKA-7143) and Zio.

Here follows a condensed example of how we'd like to use ZIO in the rebalance listener callback from the zio-kafka library.

onRevoked callback

```
def onRevoked(revokedTopicPartitions: Set[TopicPartition], consumer: KafkaConsumer) = {
  for {
    _             <- ZIO.logDebug(s"${revokedTps.size} partitions are revoked")
    state         <- currentStateRef.get
    streamsToEnd = state.assignedStreams.filter(control => revokedTps.contains(control.tp)) // Note, we run 1
stream per partition.
    _             <- ZIO.foreachParDiscard(streamsToEnd)(_.end(consumer))    // <== Streams will commit not yet
committed offsets
    _             <- awaitCommitsCompleted(consumer).timeout(15.seconds)
    _             <- ZIO.logTrace("onRevoked done")
  } yield ()
}
```

This code is run using the ZIO-runtime as follows from the {{ConsumerRebalanceListener::onPartitionsRevoked}} method:

Running ZIO code from callback

```
def onPartitionsRevoked(partitions: java.util.Collection[TopicPartition]): Unit = {
  Unsafe.unsafe { implicit u =>
    runtime.unsafe
           .run(onRevoked(partitions.asScala.toSet, consumer))
           .getOrThrowFiberFailure()
   ()
  }
}
```

(Note that this code is complex on purpose, starting a ZIO workflow from scratch is not something you would normally do.)

Look at line 6 of the first code block. In method end the stream will try to call `consumer::commitAsync(offsets, callback)`. In `awaitCommitsCompleted()` (line 7) we call `consumer::commitSync(Collections.emptyMap)` to wait untill all callbacks are invoked.

Since this code is running in the rebalance listener callback, KafkaConsumer enforces that the commit methods must be invoked from the same thread as the thread that invoked `onPartitionsRevoked`. Unfortunately, the ZIO runtime is inherently multi-threaded; tasks can be executed from any thread. There is no way Zio could support this limitation without a major rewrite.

Public Interfaces

Two methods will change from private to protected: `org.apache.kafka.clients.consumer.KafkaConsumer:acquire` and in the same class method `::release`.

Proposed Changes

See section 'public interfaces' above.

The change will allow custom sub-classes to implement `acquire` and `release` any way they like.

Implementation rules for sub classes that override acquire and release

Methods `acquire` and `release` ensure that only 1 thread can invoke the consumer at a time. Similarly, they ensure that only 1 thread can invoke the consumer from code that is running in a consumer callback.

Methods `acquire` and `release` also need to make sure that memory writes from all threads involved are visible for each other.

When `acquire` and `release` are overridden, it is up to the implementation to uphold these requirements.

Compatibility, Deprecation, and Migration Plan

For existing users nothing changes. There is no need to deprecate anything. No migration is needed.

Test Plan

Since the change does not change behavior of the library, no additional tests are needed.

Rejected Alternatives

See KIP-944 for a viable alternative.