package com.twitter.unified\_user\_actions.enricher.driver

import com.twitter.unified\_user\_actions.enricher.internal.thriftscala.EnrichmentEnvelop

import com.twitter.unified\_user\_actions.enricher.internal.thriftscala.EnrichmentKey

import com.twitter.unified\_user\_actions.enricher.internal.thriftscala.EnrichmentStageType.Hydration

import com.twitter.unified\_user\_actions.enricher.internal.thriftscala.EnrichmentStageType.Repartition

import com.twitter.util.Future

import EnrichmentPlanUtils.\_

import com.twitter.unified\_user\_actions.enricher.Exceptions

import com.twitter.unified\_user\_actions.enricher.ImplementationException

import com.twitter.unified\_user\_actions.enricher.hydrator.Hydrator

import com.twitter.unified\_user\_actions.enricher.partitioner.Partitioner

/\*\*

\* A driver that will execute on a key, value tuple and produce an output to a Kafka topic.

\*

\* The output Kafka topic will depend on the current enrichment plan. In one scenario, the driver

\* will output to a partitioned Kafka topic if the output needs to be repartitioned (after it has

\* been hydrated 0 or more times as necessary). In another scenario, the driver will output to

\* the final topic if there's no more work to be done.

\*

\* @param finalOutputTopic The final output Kafka topic

\* @param partitionedTopic The intermediate Kafka topic used for repartitioning based on [[EnrichmentKey]]

\* @param hydrator A hydrator that knows how to populate the metadata based on the current plan / instruction.

\* @param partitioner A partitioner that knows how to transform the current uua event into an [[EnrichmentKey]].

\*/

class EnrichmentDriver(

finalOutputTopic: Option[String],

partitionedTopic: String,

hydrator: Hydrator,

partitioner: Partitioner) {

/\*\*

\* A driver that does the following when being executed.

\* It checks if we are done with enrichment plan, if not:

\* - is the current stage repartitioning?

\* -> remap the output key, update plan accordingly then return with the new partition key

\* - is the current stage hydration?

\* -> use the hydrator to hydrate the envelop, update the plan accordingly, then proceed

\* recursively unless the next stage is repartitioning or this is the last stage.

\*/

def execute(

key: Option[EnrichmentKey],

envelop: Future[EnrichmentEnvelop]

): Future[(Option[EnrichmentKey], EnrichmentEnvelop)] = {

envelop.flatMap { envelop =>

val plan = envelop.plan

if (plan.isEnrichmentComplete) {

val topic = finalOutputTopic.getOrElse(

throw new ImplementationException(

"A final output Kafka topic is supposed to be used but " +

"no final output topic was provided."))

Future.value((key, envelop.copy(plan = plan.markLastStageCompletedWithOutputTopic(topic))))

} else {

val currentStage = plan.getCurrentStage

currentStage.stageType match {

case Repartition =>

Exceptions.require(

currentStage.instructions.size == 1,

s"re-partitioning needs exactly 1 instruction but ${currentStage.instructions.size} was provided")

val instruction = currentStage.instructions.head

val outputKey = partitioner.repartition(instruction, envelop)

val outputValue = envelop.copy(

plan = plan.markStageCompletedWithOutputTopic(

stage = currentStage,

outputTopic = partitionedTopic)

)

Future.value((outputKey, outputValue))

case Hydration =>

Exceptions.require(

currentStage.instructions.nonEmpty,

"hydration needs at least one instruction")

// Hydration is either initialized or completed after this, failure state

// will have to be handled upstream. Any unhandled exception will abort the entire

// stage.

// This is so that if the error in unrecoverable, the hydrator can choose to return an

// un-hydrated envelop to tolerate the error.

val finalEnvelop = currentStage.instructions.foldLeft(Future.value(envelop)) {

(curEnvelop, instruction) =>

curEnvelop.flatMap(e => hydrator.hydrate(instruction, key, e))

}

val outputValue = finalEnvelop.map(e =>

e.copy(

plan = plan.markStageCompleted(stage = currentStage)

))

// continue executing other stages if it can (locally) until a terminal state

execute(key, outputValue)

case \_ =>

throw new ImplementationException(s"Invalid / unsupported stage type $currentStage")

}

}

}

}

}