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

import com.twitter.inject.Test

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

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

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

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

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

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

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

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

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

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

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

import com.twitter.util.Await

import com.twitter.util.Future

import org.scalatest.BeforeAndAfter

import org.scalatest.matchers.should.Matchers

import scala.collection.mutable

class DriverTest extends Test with Matchers with BeforeAndAfter {

object ExecutionContext {

var executionCount = 0

}

before {

ExecutionContext.executionCount = 0

}

trait Fixtures extends EnricherFixture {

val repartitionTweet = mkStage()

val repartitionNotiTweet =

mkStage(instructions = Seq(EnrichmentInstruction.NotificationTweetEnrichment))

val hydrateTweet = mkStage(stageType = EnrichmentStageType.Hydration)

val hydrateTweetMultiInstructions = mkStage(

stageType = EnrichmentStageType.Hydration,

instructions = Seq(

EnrichmentInstruction.NotificationTweetEnrichment,

EnrichmentInstruction.TweetEnrichment,

EnrichmentInstruction.NotificationTweetEnrichment,

EnrichmentInstruction.TweetEnrichment

)

)

val hydrateNotiTweet = mkStage(

stageType = EnrichmentStageType.Hydration,

instructions = Seq(EnrichmentInstruction.NotificationTweetEnrichment))

val key1 = EnrichmentKey(EnrichmentIdType.TweetId, 123L)

val tweet1 = mkUUATweetEvent(981L)

val hydrator = new MockHydrator

val partitioner = new MockPartitioner

val outputTopic = "output"

val partitionTopic = "partition"

def complete(

enrichmentStage: EnrichmentStage,

outputTopic: Option[String] = None

): EnrichmentStage = {

enrichmentStage.copy(status = EnrichmentStageStatus.Completion, outputTopic = outputTopic)

}

def mkPlan(enrichmentStages: EnrichmentStage\*): EnrichmentPlan = {

EnrichmentPlan(enrichmentStages)

}

def mkStage(

status: EnrichmentStageStatus = EnrichmentStageStatus.Initialized,

stageType: EnrichmentStageType = EnrichmentStageType.Repartition,

instructions: Seq[EnrichmentInstruction] = Seq(EnrichmentInstruction.TweetEnrichment)

): EnrichmentStage = {

EnrichmentStage(status, stageType, instructions)

}

trait ExecutionCount {

val callMap: mutable.Map[Int, (EnrichmentInstruction, EnrichmentEnvelop)] =

mutable.Map[Int, (EnrichmentInstruction, EnrichmentEnvelop)]()

def recordExecution(instruction: EnrichmentInstruction, envelop: EnrichmentEnvelop): Unit = {

ExecutionContext.executionCount = ExecutionContext.executionCount + 1

callMap.put(ExecutionContext.executionCount, (instruction, envelop))

}

}

class MockHydrator extends Hydrator with ExecutionCount {

def hydrate(

instruction: EnrichmentInstruction,

key: Option[EnrichmentKey],

envelop: EnrichmentEnvelop

): Future[EnrichmentEnvelop] = {

recordExecution(instruction, envelop)

Future(envelop.copy(envelopId = ExecutionContext.executionCount))

}

}

class MockPartitioner extends Partitioner with ExecutionCount {

def repartition(

instruction: EnrichmentInstruction,

envelop: EnrichmentEnvelop

): Option[EnrichmentKey] = {

recordExecution(instruction, envelop)

Some(EnrichmentKey(EnrichmentIdType.TweetId, ExecutionContext.executionCount))

}

}

}

test("single partitioning plan works") {

new Fixtures {

val driver = new EnrichmentDriver(Some(outputTopic), partitionTopic, hydrator, partitioner)

// given a simple plan that only repartition the input and nothing else

val plan = mkPlan(repartitionTweet)

(1L to 10).foreach(id => {

val envelop = EnrichmentEnvelop(id, tweet1, plan)

// when

val actual = Await.result(driver.execute(Some(key1), Future(envelop)))

val expectedKey = Some(key1.copy(id = id))

val expectedValue =

envelop.copy(plan = mkPlan(complete(repartitionTweet, Some(partitionTopic))))

// then the result should have a new partitioned key, with the envelop unchanged except the plan is complete

// however, the output topic is the partitionTopic (since this is only a partitioning stage)

assert((expectedKey, expectedValue) == actual)

})

}

}

test("multi-stage partitioning plan works") {

new Fixtures {

val driver = new EnrichmentDriver(Some(outputTopic), partitionTopic, hydrator, partitioner)

// given a plan that chain multiple repartition stages together

val plan = mkPlan(repartitionTweet, repartitionNotiTweet)

val envelop1 = EnrichmentEnvelop(1L, tweet1, plan)

// when 1st partitioning trip

val actual1 = Await.result(driver.execute(Some(key1), Future(envelop1)))

// then the result should have a new partitioned key, with the envelop unchanged except the

// 1st stage of the plan is complete

val expectedKey1 = key1.copy(id = 1L)

val expectedValue1 =

envelop1.copy(plan =

mkPlan(complete(repartitionTweet, Some(partitionTopic)), repartitionNotiTweet))

assert((Some(expectedKey1), expectedValue1) == actual1)

// then, we reuse the last result to exercise the logics on the driver again for the 2st trip

val actual2 = Await.result(driver.execute(Some(expectedKey1), Future(expectedValue1)))

val expectedKey2 = key1.copy(id = 2L)

val expectedValue2 =

envelop1.copy(plan = mkPlan(

complete(repartitionTweet, Some(partitionTopic)),

complete(repartitionNotiTweet, Some(partitionTopic))))

assert((Some(expectedKey2), expectedValue2) == actual2)

}

}

test("single hydration plan works") {

new Fixtures {

val driver = new EnrichmentDriver(Some(outputTopic), partitionTopic, hydrator, partitioner)

// given a simple plan that only hydrate the input and nothing else

val plan = mkPlan(hydrateTweet)

(1L to 10).foreach(id => {

val envelop = EnrichmentEnvelop(id, tweet1, plan)

// when

val actual = Await.result(driver.execute(Some(key1), Future(envelop)))

val expectedValue =

envelop.copy(envelopId = id, plan = mkPlan(complete(hydrateTweet, Some(outputTopic))))

// then the result should have the same key, with the envelop hydrated & the plan is complete

// the output topic should be the final topic since this is a hydration stage and the plan is complete

assert((Some(key1), expectedValue) == actual)

})

}

}

test("single hydration with multiple instructions plan works") {

new Fixtures {

val driver = new EnrichmentDriver(Some(outputTopic), partitionTopic, hydrator, partitioner)

// given a simple plan that only hydrate the input and nothing else

val plan = mkPlan(hydrateTweetMultiInstructions)

val envelop = EnrichmentEnvelop(0L, tweet1, plan)

// when

val actual = Await.result(driver.execute(Some(key1), Future(envelop)))

val expectedValue = envelop.copy(

envelopId = 4L, // hydrate is called 4 times for 4 instructions in 1 stage

plan = mkPlan(complete(hydrateTweetMultiInstructions, Some(outputTopic))))

// then the result should have the same key, with the envelop hydrated & the plan is complete

// the output topic should be the final topic since this is a hydration stage and the plan is complete

assert((Some(key1), expectedValue) == actual)

}

}

test("multi-stage hydration plan works") {

new Fixtures {

val driver = new EnrichmentDriver(Some(outputTopic), partitionTopic, hydrator, partitioner)

// given a plan that only hydrate twice

val plan = mkPlan(hydrateTweet, hydrateNotiTweet)

val envelop = EnrichmentEnvelop(1L, tweet1, plan)

// when

val actual = Await.result(driver.execute(Some(key1), Future(envelop)))

// then the result should have the same key, with the envelop hydrated. since there's no

// partitioning stages, the driver will just recurse until all the hydration is done,

// then output to the final topic

val expectedValue =

envelop.copy(

envelopId = 2L,

plan = mkPlan(

complete(hydrateTweet),

complete(

hydrateNotiTweet,

Some(outputTopic)

) // only the last stage has the output topic

))

assert((Some(key1), expectedValue) == actual)

}

}

test("multi-stage partition+hydration plan works") {

new Fixtures {

val driver = new EnrichmentDriver(Some(outputTopic), partitionTopic, hydrator, partitioner)

// given a plan that repartition then hydrate twice

val plan = mkPlan(repartitionTweet, hydrateTweet, repartitionNotiTweet, hydrateNotiTweet)

var curEnvelop = EnrichmentEnvelop(1L, tweet1, plan)

var curKey = key1

// stage 1, partitioning on tweet should be correct

var actual = Await.result(driver.execute(Some(curKey), Future(curEnvelop)))

var expectedKey = curKey.copy(id = 1L)

var expectedValue = curEnvelop.copy(

plan = mkPlan(

complete(repartitionTweet, Some(partitionTopic)),

hydrateTweet,

repartitionNotiTweet,

hydrateNotiTweet))

assert((Some(expectedKey), expectedValue) == actual)

curEnvelop = actual.\_2

curKey = actual.\_1.get

// stage 2-3, hydrating on tweet should be correct

// and since the next stage after hydration is a repartition, it will does so correctly

actual = Await.result(driver.execute(Some(curKey), Future(curEnvelop)))

expectedKey = curKey.copy(id = 3) // repartition is done in stage 3

expectedValue = curEnvelop.copy(

envelopId = 2L, // hydration is done in stage 2

plan = mkPlan(

complete(repartitionTweet, Some(partitionTopic)),

complete(hydrateTweet),

complete(repartitionNotiTweet, Some(partitionTopic)),

hydrateNotiTweet)

)

assert((Some(expectedKey), expectedValue) == actual)

curEnvelop = actual.\_2

curKey = actual.\_1.get

// then finally, stage 4 would output to the final topic

actual = Await.result(driver.execute(Some(curKey), Future(curEnvelop)))

expectedKey = curKey // nothing's changed in the key

expectedValue = curEnvelop.copy(

envelopId = 4L,

plan = mkPlan(

complete(repartitionTweet, Some(partitionTopic)),

complete(hydrateTweet),

complete(repartitionNotiTweet, Some(partitionTopic)),

complete(hydrateNotiTweet, Some(outputTopic))

)

)

assert((Some(expectedKey), expectedValue) == actual)

}

}

}