package com.twitter.unified\_user\_actions.service

import com.twitter.conversions.DurationOps.\_

import com.twitter.conversions.StorageUnitOps.\_

import com.twitter.dynmap.DynMap

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.finatra.kafka.domain.AckMode

import com.twitter.finatra.kafka.domain.KafkaGroupId

import com.twitter.finatra.kafka.serde.ScalaSerdes

import com.twitter.finatra.kafkastreams.config.KafkaStreamsConfig

import com.twitter.finatra.kafkastreams.config.SecureKafkaStreamsConfig

import com.twitter.finatra.kafkastreams.partitioning.StaticPartitioning

import com.twitter.finatra.mtls.modules.ServiceIdentifierModule

import com.twitter.finatra.kafkastreams.dsl.FinatraDslFlatMapAsync

import com.twitter.graphql.thriftscala.GraphqlExecutionService

import com.twitter.logging.Logging

import com.twitter.unified\_user\_actions.enricher.driver.EnrichmentDriver

import com.twitter.unified\_user\_actions.enricher.hcache.LocalCache

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

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.partitioner.DefaultPartitioner

import com.twitter.unified\_user\_actions.service.module.CacheModule

import com.twitter.unified\_user\_actions.service.module.ClientIdModule

import com.twitter.unified\_user\_actions.service.module.GraphqlClientProviderModule

import com.twitter.util.Future

import org.apache.kafka.common.record.CompressionType

import org.apache.kafka.streams.StreamsBuilder

import org.apache.kafka.streams.processor.RecordContext

import org.apache.kafka.streams.processor.TopicNameExtractor

import org.apache.kafka.streams.scala.kstream.Consumed

import org.apache.kafka.streams.scala.kstream.Produced

import com.twitter.unified\_user\_actions.enricher.driver.EnrichmentPlanUtils.\_

object EnricherServiceMain extends EnricherService

class EnricherService

extends FinatraDslFlatMapAsync

with StaticPartitioning

with SecureKafkaStreamsConfig

with Logging {

val InputTopic = "unified\_user\_actions\_keyed\_dev"

val OutputTopic = "unified\_user\_actions\_enriched"

override val modules = Seq(

CacheModule,

ClientIdModule,

GraphqlClientProviderModule,

ServiceIdentifierModule

)

override protected def configureKafkaStreams(builder: StreamsBuilder): Unit = {

val graphqlClient = injector.instance[GraphqlExecutionService.FinagledClient]

val localCache = injector.instance[LocalCache[EnrichmentKey, DynMap]]

val statsReceiver = injector.instance[StatsReceiver]

val driver = new EnrichmentDriver(

finalOutputTopic = Some(OutputTopic),

partitionedTopic = InputTopic,

hydrator = new DefaultHydrator(

cache = localCache,

graphqlClient = graphqlClient,

scopedStatsReceiver = statsReceiver.scope("DefaultHydrator")),

partitioner = new DefaultPartitioner

)

val kstream = builder.asScala

.stream(InputTopic)(

Consumed.`with`(ScalaSerdes.Thrift[EnrichmentKey], ScalaSerdes.Thrift[EnrichmentEnvelop]))

.flatMapAsync[EnrichmentKey, EnrichmentEnvelop](

commitInterval = 5.seconds,

numWorkers = 10000

) { (enrichmentKey: EnrichmentKey, enrichmentEnvelop: EnrichmentEnvelop) =>

driver

.execute(Some(enrichmentKey), Future.value(enrichmentEnvelop))

.map(tuple => tuple.\_1.map(key => (key, tuple.\_2)).seq)

}

val topicExtractor: TopicNameExtractor[EnrichmentKey, EnrichmentEnvelop] =

(\_: EnrichmentKey, envelop: EnrichmentEnvelop, \_: RecordContext) =>

envelop.plan.getLastCompletedStage.outputTopic.getOrElse(

throw new IllegalStateException("Missing output topic in the last completed stage"))

kstream.to(topicExtractor)(

Produced.`with`(ScalaSerdes.Thrift[EnrichmentKey], ScalaSerdes.Thrift[EnrichmentEnvelop]))

}

override def streamsProperties(config: KafkaStreamsConfig): KafkaStreamsConfig =

super

.streamsProperties(config)

.consumer.groupId(KafkaGroupId(applicationId()))

.consumer.clientId(s"${applicationId()}-consumer")

.consumer.requestTimeout(30.seconds)

.consumer.sessionTimeout(30.seconds)

.consumer.fetchMin(1.megabyte)

.consumer.fetchMax(5.megabytes)

.consumer.receiveBuffer(32.megabytes)

.consumer.maxPollInterval(1.minute)

.consumer.maxPollRecords(50000)

.producer.clientId(s"${applicationId()}-producer")

.producer.batchSize(16.kilobytes)

.producer.bufferMemorySize(256.megabyte)

.producer.requestTimeout(30.seconds)

.producer.compressionType(CompressionType.LZ4)

.producer.ackMode(AckMode.ALL)

}