package com.twitter.recosinjector.uua\_processors

import org.apache.kafka.clients.consumer.ConsumerRecord

import com.twitter.finatra.kafka.serde.UnKeyed

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.recos.util.Action

import com.twitter.recos.util.Action.Action

import com.twitter.recosinjector.clients.Gizmoduck

import com.twitter.recosinjector.clients.Tweetypie

import com.twitter.recosinjector.edges.UnifiedUserActionToUserVideoGraphBuilder

import com.twitter.recosinjector.edges.UnifiedUserActionToUserAdGraphBuilder

import com.twitter.recosinjector.edges.UnifiedUserActionToUserTweetGraphPlusBuilder

import com.twitter.unified\_user\_actions.thriftscala.UnifiedUserAction

import com.twitter.unified\_user\_actions.thriftscala.ActionType

import com.twitter.unified\_user\_actions.thriftscala.Item

import com.twitter.recosinjector.filters.UserFilter

import com.twitter.recosinjector.publishers.KafkaEventPublisher

import com.twitter.recosinjector.util.TweetDetails

import com.twitter.recosinjector.util.UserTweetEngagement

import com.twitter.recosinjector.util.UuaEngagementEventDetails

import com.twitter.unified\_user\_actions.thriftscala.NotificationContent

import com.twitter.unified\_user\_actions.thriftscala.NotificationInfo

import com.twitter.util.Future

class UnifiedUserActionProcessor(

gizmoduck: Gizmoduck,

tweetypie: Tweetypie,

kafkaEventPublisher: KafkaEventPublisher,

userVideoGraphTopic: String,

userVideoGraphBuilder: UnifiedUserActionToUserVideoGraphBuilder,

userAdGraphTopic: String,

userAdGraphBuilder: UnifiedUserActionToUserAdGraphBuilder,

userTweetGraphPlusTopic: String,

userTweetGraphPlusBuilder: UnifiedUserActionToUserTweetGraphPlusBuilder

)(

implicit statsReceiver: StatsReceiver) {

val messagesProcessedCount = statsReceiver.counter("messages\_processed")

val eventsByTypeCounts = statsReceiver.scope("events\_by\_type")

private val numSelfEngageCounter = statsReceiver.counter("num\_self\_engage\_event")

private val numTweetFailSafetyLevelCounter = statsReceiver.counter("num\_fail\_tweetypie\_safety")

private val numNullCastTweetCounter = statsReceiver.counter("num\_null\_cast\_tweet")

private val numEngageUserUnsafeCounter = statsReceiver.counter("num\_engage\_user\_unsafe")

private val engageUserFilter = new UserFilter(gizmoduck)(statsReceiver.scope("engage\_user"))

private val numNoProcessTweetCounter = statsReceiver.counter("num\_no\_process\_tweet")

private val numProcessTweetCounter = statsReceiver.counter("num\_process\_tweet")

private def getUuaEngagementEventDetails(

unifiedUserAction: UnifiedUserAction

): Option[Future[UuaEngagementEventDetails]] = {

val userIdOpt = unifiedUserAction.userIdentifier.userId

val tweetIdOpt = unifiedUserAction.item match {

case Item.TweetInfo(tweetInfo) => Some(tweetInfo.actionTweetId)

case Item.NotificationInfo(

NotificationInfo(\_, NotificationContent.TweetNotification(notification))) =>

Some(notification.tweetId)

case \_ => None

}

val timestamp = unifiedUserAction.eventMetadata.sourceTimestampMs

val action = getTweetAction(unifiedUserAction.actionType)

tweetIdOpt

.flatMap { tweetId =>

userIdOpt.map { engageUserId =>

val tweetFut = tweetypie.getTweet(tweetId)

tweetFut.map { tweetOpt =>

val tweetDetailsOpt = tweetOpt.map(TweetDetails)

val engagement = UserTweetEngagement(

engageUserId = engageUserId,

action = action,

engagementTimeMillis = Some(timestamp),

tweetId = tweetId,

engageUser = None,

tweetDetails = tweetDetailsOpt

)

UuaEngagementEventDetails(engagement)

}

}

}

}

private def getTweetAction(action: ActionType): Action = {

action match {

case ActionType.ClientTweetVideoPlayback50 => Action.VideoPlayback50

case ActionType.ClientTweetClick => Action.Click

case ActionType.ClientTweetVideoPlayback75 => Action.VideoPlayback75

case ActionType.ClientTweetVideoQualityView => Action.VideoQualityView

case ActionType.ServerTweetFav => Action.Favorite

case ActionType.ServerTweetReply => Action.Reply

case ActionType.ServerTweetRetweet => Action.Retweet

case ActionType.ClientTweetQuote => Action.Quote

case ActionType.ClientNotificationOpen => Action.NotificationOpen

case ActionType.ClientTweetEmailClick => Action.EmailClick

case ActionType.ClientTweetShareViaBookmark => Action.Share

case ActionType.ClientTweetShareViaCopyLink => Action.Share

case ActionType.ClientTweetSeeFewer => Action.TweetSeeFewer

case ActionType.ClientTweetNotRelevant => Action.TweetNotRelevant

case ActionType.ClientTweetNotInterestedIn => Action.TweetNotInterestedIn

case ActionType.ServerTweetReport => Action.TweetReport

case ActionType.ClientTweetMuteAuthor => Action.TweetMuteAuthor

case ActionType.ClientTweetBlockAuthor => Action.TweetBlockAuthor

case \_ => Action.UnDefined

}

}

private def shouldProcessTweetEngagement(

event: UuaEngagementEventDetails,

isAdsUseCase: Boolean = false

): Future[Boolean] = {

val engagement = event.userTweetEngagement

val engageUserId = engagement.engageUserId

val authorIdOpt = engagement.tweetDetails.flatMap(\_.authorId)

val isSelfEngage = authorIdOpt.contains(engageUserId)

val isNullCastTweet = engagement.tweetDetails.forall(\_.isNullCastTweet)

val isEngageUserSafeFut = engageUserFilter.filterByUserId(engageUserId)

val isTweetPassSafety =

engagement.tweetDetails.isDefined // Tweetypie can fetch a tweet object successfully

isEngageUserSafeFut.map { isEngageUserSafe =>

if (isSelfEngage) numSelfEngageCounter.incr()

if (isNullCastTweet) numNullCastTweetCounter.incr()

if (!isEngageUserSafe) numEngageUserUnsafeCounter.incr()

if (!isTweetPassSafety) numTweetFailSafetyLevelCounter.incr()

!isSelfEngage && (!isNullCastTweet && !isAdsUseCase || isNullCastTweet && isAdsUseCase) && isEngageUserSafe && isTweetPassSafety

}

}

def apply(record: ConsumerRecord[UnKeyed, UnifiedUserAction]): Future[Unit] = {

messagesProcessedCount.incr()

val unifiedUserAction = record.value

eventsByTypeCounts.counter(unifiedUserAction.actionType.toString).incr()

getTweetAction(unifiedUserAction.actionType) match {

case Action.UnDefined =>

numNoProcessTweetCounter.incr()

Future.Unit

case action =>

getUuaEngagementEventDetails(unifiedUserAction)

.map {

\_.flatMap { detail =>

// The following cases are set up specifically for an ads relevance demo.

val actionForAds = Set(Action.Click, Action.Favorite, Action.VideoPlayback75)

if (actionForAds.contains(action))

shouldProcessTweetEngagement(detail, isAdsUseCase = true).map {

case true =>

userAdGraphBuilder.processEvent(detail).map { edges =>

edges.foreach { edge =>

kafkaEventPublisher

.publish(edge.convertToRecosHoseMessage, userAdGraphTopic)

}

}

numProcessTweetCounter.incr()

case \_ =>

}

shouldProcessTweetEngagement(detail).map {

case true =>

userVideoGraphBuilder.processEvent(detail).map { edges =>

edges.foreach { edge =>

kafkaEventPublisher

.publish(edge.convertToRecosHoseMessage, userVideoGraphTopic)

}

}

userTweetGraphPlusBuilder.processEvent(detail).map { edges =>

edges.foreach { edge =>

kafkaEventPublisher

.publish(edge.convertToRecosHoseMessage, userTweetGraphPlusTopic)

}

}

numProcessTweetCounter.incr()

case \_ =>

}

}

}.getOrElse(Future.Unit)

}

}

}