package com.twitter.home\_mixer.product.for\_you.side\_effect

import com.twitter.home\_mixer.model.HomeFeatures.IsReadFromCacheFeature

import com.twitter.home\_mixer.model.HomeFeatures.PredictionRequestIdFeature

import com.twitter.home\_mixer.model.HomeFeatures.ServedIdFeature

import com.twitter.home\_mixer.model.HomeFeatures.ServedRequestIdFeature

import com.twitter.home\_mixer.product.for\_you.param.ForYouParam.EnableServedCandidateKafkaPublishingParam

import com.twitter.home\_mixer.service.HomeMixerAlertConfig

import com.twitter.ml.api.DataRecord

import com.twitter.ml.api.util.SRichDataRecord

import com.twitter.product\_mixer.component\_library.side\_effect.KafkaPublishingSideEffect

import com.twitter.product\_mixer.core.feature.featuremap.FeatureMap

import com.twitter.product\_mixer.core.functional\_component.side\_effect.PipelineResultSideEffect

import com.twitter.product\_mixer.core.model.common.identifier.CandidatePipelineIdentifier

import com.twitter.product\_mixer.core.model.common.identifier.SideEffectIdentifier

import com.twitter.product\_mixer.core.model.common.presentation.CandidateWithDetails

import com.twitter.product\_mixer.core.model.marshalling.response.urt.Timeline

import com.twitter.product\_mixer.core.pipeline.PipelineQuery

import com.twitter.timelines.ml.cont\_train.common.domain.non\_scalding.DataRecordLoggingRelatedFeatures.tlmServedKeysFeatureContext

import com.twitter.timelines.ml.kafka.serde.ServedCandidateKeySerde

import com.twitter.timelines.ml.kafka.serde.TBaseSerde

import com.twitter.timelines.prediction.features.common.TimelinesSharedFeatures

import com.twitter.timelines.served\_candidates\_logging.{thriftscala => sc}

import com.twitter.timelines.suggests.common.poly\_data\_record.{thriftjava => pldr}

import com.twitter.util.Time

import org.apache.kafka.clients.producer.ProducerRecord

import org.apache.kafka.common.serialization.Serializer

/\*\*

\* Pipeline side-effect that publishes candidate keys to a Kafka topic.

\*/

class ServedCandidateKeysKafkaSideEffect(

topic: String,

sourceIdentifiers: Set[CandidatePipelineIdentifier])

extends KafkaPublishingSideEffect[

sc.ServedCandidateKey,

pldr.PolyDataRecord,

PipelineQuery,

Timeline

]

with PipelineResultSideEffect.Conditionally[PipelineQuery, Timeline] {

import ServedCandidateKafkaSideEffect.\_

override val identifier: SideEffectIdentifier = SideEffectIdentifier("ServedCandidateKeys")

override def onlyIf(

query: PipelineQuery,

selectedCandidates: Seq[CandidateWithDetails],

remainingCandidates: Seq[CandidateWithDetails],

droppedCandidates: Seq[CandidateWithDetails],

response: Timeline

): Boolean = query.params.getBoolean(EnableServedCandidateKafkaPublishingParam)

override val bootstrapServer: String = "/s/kafka/timeline:kafka-tls"

override val keySerde: Serializer[sc.ServedCandidateKey] = ServedCandidateKeySerde.serializer()

override val valueSerde: Serializer[pldr.PolyDataRecord] =

TBaseSerde.Thrift[pldr.PolyDataRecord]().serializer

override val clientId: String = "home\_mixer\_served\_candidate\_keys\_producer"

override def buildRecords(

query: PipelineQuery,

selectedCandidates: Seq[CandidateWithDetails],

remainingCandidates: Seq[CandidateWithDetails],

droppedCandidates: Seq[CandidateWithDetails],

response: Timeline

): Seq[ProducerRecord[sc.ServedCandidateKey, pldr.PolyDataRecord]] = {

val servedTimestamp = Time.now.inMilliseconds

val servedRequestIdOpt =

query.features.getOrElse(FeatureMap.empty).getOrElse(ServedRequestIdFeature, None)

extractCandidates(query, selectedCandidates, sourceIdentifiers).collect {

// Only publish non-cached tweets to the ServedCandidateKey topic

case candidate if !candidate.features.getOrElse(IsReadFromCacheFeature, false) =>

val key = sc.ServedCandidateKey(

tweetId = candidate.candidateIdLong,

viewerId = query.getRequiredUserId,

servedId = -1L

)

val record = SRichDataRecord(new DataRecord, tlmServedKeysFeatureContext)

record.setFeatureValueFromOption(

TimelinesSharedFeatures.PREDICTION\_REQUEST\_ID,

candidate.features.getOrElse(PredictionRequestIdFeature, None)

)

record

.setFeatureValueFromOption(TimelinesSharedFeatures.SERVED\_REQUEST\_ID, servedRequestIdOpt)

record.setFeatureValueFromOption(

TimelinesSharedFeatures.SERVED\_ID,

candidate.features.getOrElse(ServedIdFeature, None)

)

record.setFeatureValueFromOption(

TimelinesSharedFeatures.INJECTION\_TYPE,

record.getFeatureValueOpt(TimelinesSharedFeatures.INJECTION\_TYPE))

record.setFeatureValue(

TimelinesSharedFeatures.SERVED\_TIMESTAMP,

servedTimestamp

)

record.record.dropUnknownFeatures()

new ProducerRecord(topic, key, pldr.PolyDataRecord.dataRecord(record.getRecord))

}

}

override val alerts = Seq(

HomeMixerAlertConfig.BusinessHours.defaultSuccessRateAlert(98.5)

)

}