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

import com.google.inject.Provides

import com.twitter.decider.Decider

import com.twitter.finagle.stats.StatsReceiver

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

import com.twitter.finatra.kafka.serde.UnKeyedSerde

import com.twitter.ibis.thriftscala.NotificationScribe

import com.twitter.inject.TwitterModule

import com.twitter.inject.annotations.Flag

import com.twitter.kafka.client.processor.AtLeastOnceProcessor

import com.twitter.unified\_user\_actions.kafka.CompressionTypeFlag

import com.twitter.unified\_user\_actions.kafka.serde.NullableScalaSerdes

import com.twitter.unified\_user\_actions.adapter.email\_notification\_event.EmailNotificationEventAdapter

import com.twitter.util.Duration

import com.twitter.util.StorageUnit

import com.twitter.util.logging.Logging

import javax.inject.Singleton

object KafkaProcessorEmailNotificationEventModule extends TwitterModule with Logging {

override def modules = Seq(FlagsModule)

private val notificationEventAdapter = new EmailNotificationEventAdapter

// NOTE: This is a shared processor name in order to simplify monviz stat computation.

private final val processorName = "uuaProcessor"

@Provides

@Singleton

def providesKafkaProcessor(

decider: Decider,

@Flag(FlagsModule.cluster) cluster: String,

@Flag(FlagsModule.kafkaSourceCluster) kafkaSourceCluster: String,

@Flag(FlagsModule.kafkaDestCluster) kafkaDestCluster: String,

@Flag(FlagsModule.kafkaSourceTopic) kafkaSourceTopic: String,

@Flag(FlagsModule.kafkaSinkTopics) kafkaSinkTopics: Seq[String],

@Flag(FlagsModule.kafkaGroupId) kafkaGroupId: String,

@Flag(FlagsModule.kafkaProducerClientId) kafkaProducerClientId: String,

@Flag(FlagsModule.kafkaMaxPendingRequests) kafkaMaxPendingRequests: Int,

@Flag(FlagsModule.kafkaWorkerThreads) kafkaWorkerThreads: Int,

@Flag(FlagsModule.commitInterval) commitInterval: Duration,

@Flag(FlagsModule.maxPollRecords) maxPollRecords: Int,

@Flag(FlagsModule.maxPollInterval) maxPollInterval: Duration,

@Flag(FlagsModule.sessionTimeout) sessionTimeout: Duration,

@Flag(FlagsModule.fetchMax) fetchMax: StorageUnit,

@Flag(FlagsModule.batchSize) batchSize: StorageUnit,

@Flag(FlagsModule.linger) linger: Duration,

@Flag(FlagsModule.bufferMem) bufferMem: StorageUnit,

@Flag(FlagsModule.compressionType) compressionTypeFlag: CompressionTypeFlag,

@Flag(FlagsModule.retries) retries: Int,

@Flag(FlagsModule.retryBackoff) retryBackoff: Duration,

@Flag(FlagsModule.requestTimeout) requestTimeout: Duration,

@Flag(FlagsModule.enableTrustStore) enableTrustStore: Boolean,

@Flag(FlagsModule.trustStoreLocation) trustStoreLocation: String,

statsReceiver: StatsReceiver,

): AtLeastOnceProcessor[UnKeyed, NotificationScribe] = {

KafkaProcessorProvider.provideDefaultAtLeastOnceProcessor(

name = processorName,

kafkaSourceCluster = kafkaSourceCluster,

kafkaGroupId = kafkaGroupId,

kafkaSourceTopic = kafkaSourceTopic,

sourceKeyDeserializer = UnKeyedSerde.deserializer,

sourceValueDeserializer = NullableScalaSerdes

.Thrift[NotificationScribe](statsReceiver.counter("deserializerErrors")).deserializer,

commitInterval = commitInterval,

maxPollRecords = maxPollRecords,

maxPollInterval = maxPollInterval,

sessionTimeout = sessionTimeout,

fetchMax = fetchMax,

processorMaxPendingRequests = kafkaMaxPendingRequests,

processorWorkerThreads = kafkaWorkerThreads,

adapter = notificationEventAdapter,

kafkaSinkTopics = kafkaSinkTopics,

kafkaDestCluster = kafkaDestCluster,

kafkaProducerClientId = kafkaProducerClientId,

batchSize = batchSize,

linger = linger,

bufferMem = bufferMem,

compressionType = compressionTypeFlag.compressionType,

retries = retries,

retryBackoff = retryBackoff,

requestTimeout = requestTimeout,

statsReceiver = statsReceiver,

trustStoreLocationOpt = if (enableTrustStore) Some(trustStoreLocation) else None,

decider = decider,

zone = ZoneFiltering.zoneMapping(cluster),

maybeProcess = ZoneFiltering.localDCFiltering

)

}

}