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

import com.google.inject.Stage

import com.twitter.adserver.thriftscala.DisplayLocation

import com.twitter.app.GlobalFlag

import com.twitter.finatra.kafka.consumers.FinagleKafkaConsumerBuilder

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

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

import com.twitter.finatra.kafka.domain.KafkaTopic

import com.twitter.finatra.kafka.domain.SeekStrategy

import com.twitter.finatra.kafka.producers.FinagleKafkaProducerBuilder

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

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

import com.twitter.finatra.kafka.test.KafkaFeatureTest

import com.twitter.iesource.thriftscala.ClientEventContext

import com.twitter.iesource.thriftscala.TweetImpression

import com.twitter.iesource.thriftscala.ClientType

import com.twitter.iesource.thriftscala.ContextualEventNamespace

import com.twitter.iesource.thriftscala.EngagingContext

import com.twitter.iesource.thriftscala.EventSource

import com.twitter.iesource.thriftscala.InteractionDetails

import com.twitter.iesource.thriftscala.InteractionEvent

import com.twitter.iesource.thriftscala.InteractionType

import com.twitter.iesource.thriftscala.InteractionTargetType

import com.twitter.iesource.thriftscala.UserIdentifier

import com.twitter.inject.server.EmbeddedTwitterServer

import com.twitter.kafka.client.processor.KafkaConsumerClient

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

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

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

import com.twitter.util.Duration

import com.twitter.util.StorageUnit

class RekeyUuaIesourceServiceStartupTest extends KafkaFeatureTest {

private val inputTopic =

kafkaTopic(ScalaSerdes.Long, ScalaSerdes.CompactThrift[InteractionEvent], name = "source")

private val outputTopic =

kafkaTopic(ScalaSerdes.Long, ScalaSerdes.Thrift[KeyedUuaTweet], name = "sink")

val startupFlags = Map(

"kafka.group.id" -> "client-event",

"kafka.producer.client.id" -> "uua",

"kafka.source.topic" -> inputTopic.topic,

"kafka.sink.topics" -> outputTopic.topic,

"kafka.consumer.fetch.min" -> "6.megabytes",

"kafka.max.pending.requests" -> "100",

"kafka.worker.threads" -> "1",

"kafka.trust.store.enable" -> "false",

"kafka.producer.batch.size" -> "0.byte",

"cluster" -> "atla",

)

val deciderFlags = Map(

"decider.base" -> "/decider.yml"

)

override protected def kafkaBootstrapFlag: Map[String, String] = {

Map(

ClientConfigs.kafkaBootstrapServerConfig -> kafkaCluster.bootstrapServers(),

ClientConfigs.kafkaBootstrapServerRemoteDestConfig -> kafkaCluster.bootstrapServers(),

)

}

override val server: EmbeddedTwitterServer = new EmbeddedTwitterServer(

twitterServer = new RekeyUuaIesourceService() {

override def warmup(): Unit = {

// noop

}

override val overrideModules = Seq(

KafkaProcessorRekeyUuaIesourceModule

)

},

globalFlags = Map[GlobalFlag[\_], String](

com.twitter.finatra.kafka.consumers.enableTlsAndKerberos -> "false",

),

flags = startupFlags ++ kafkaBootstrapFlag ++ deciderFlags,

stage = Stage.PRODUCTION

)

private def getConsumer(

seekStrategy: SeekStrategy = SeekStrategy.BEGINNING,

) = {

val builder = FinagleKafkaConsumerBuilder()

.dest(brokers.map(\_.brokerList()).mkString(","))

.clientId("consumer")

.groupId(KafkaGroupId("validator"))

.keyDeserializer(ScalaSerdes.Long.deserializer)

.valueDeserializer(ScalaSerdes.CompactThrift[InteractionEvent].deserializer)

.requestTimeout(Duration.fromSeconds(1))

.enableAutoCommit(false)

.seekStrategy(seekStrategy)

new KafkaConsumerClient(builder.config)

}

private def getUUAConsumer(

seekStrategy: SeekStrategy = SeekStrategy.BEGINNING,

) = {

val builder = FinagleKafkaConsumerBuilder()

.dest(brokers.map(\_.brokerList()).mkString(","))

.clientId("consumer\_uua")

.groupId(KafkaGroupId("validator\_uua"))

.keyDeserializer(UnKeyedSerde.deserializer)

.valueDeserializer(ScalaSerdes.Thrift[KeyedUuaTweet].deserializer)

.requestTimeout(Duration.fromSeconds(1))

.enableAutoCommit(false)

.seekStrategy(seekStrategy)

new KafkaConsumerClient(builder.config)

}

private def getProducer(clientId: String = "producer") = {

FinagleKafkaProducerBuilder()

.dest(brokers.map(\_.brokerList()).mkString(","))

.clientId(clientId)

.ackMode(AckMode.ALL)

.batchSize(StorageUnit.zero)

.keySerializer(ScalaSerdes.Long.serializer)

.valueSerializer(ScalaSerdes.CompactThrift[InteractionEvent].serializer)

.build()

}

test("RekeyUuaIesourceService starts") {

server.assertHealthy()

}

test("RekeyUuaIesourceService should process input events") {

val producer = getProducer()

val inputConsumer = getConsumer()

val uuaConsumer = getUUAConsumer()

val value: InteractionEvent = InteractionEvent(

targetId = 1L,

targetType = InteractionTargetType.Tweet,

engagingUserId = 11L,

eventSource = EventSource.ClientEvent,

timestampMillis = 123456L,

interactionType = Some(InteractionType.TweetRenderImpression),

details = InteractionDetails.TweetRenderImpression(TweetImpression()),

additionalEngagingUserIdentifiers = UserIdentifier(),

engagingContext = EngagingContext.ClientEventContext(

ClientEventContext(

clientEventNamespace = ContextualEventNamespace(),

clientType = ClientType.Iphone,

displayLocation = DisplayLocation(1)))

)

try {

server.assertHealthy()

// before, should be empty

inputConsumer.subscribe(Set(KafkaTopic(inputTopic.topic)))

assert(inputConsumer.poll().count() == 0)

// after, should contain at least a message

await(producer.send(inputTopic.topic, value.targetId, value, System.currentTimeMillis))

producer.flush()

assert(inputConsumer.poll().count() == 1)

uuaConsumer.subscribe(Set(KafkaTopic(outputTopic.topic)))

// This is tricky: it is not guaranteed that the srvice can process and output the

// event to output topic faster than the below consumer. So we'd use a timer here which may

// not be the best practice.

// If someone finds the below test is flaky, please just remove the below test completely.

Thread.sleep(5000L)

assert(uuaConsumer.poll().count() == 1)

} finally {

await(producer.close())

inputConsumer.close()

}

}

}