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

import com.google.inject.Stage

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.UnKeyed

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

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

import com.twitter.inject.server.EmbeddedTwitterServer

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

import com.twitter.timelineservice.thriftscala.ContextualizedFavoriteEvent

import com.twitter.timelineservice.thriftscala.FavoriteEvent

import com.twitter.timelineservice.thriftscala.FavoriteEventUnion

import com.twitter.timelineservice.thriftscala.LogEventContext

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

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

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

import com.twitter.util.Duration

import com.twitter.util.StorageUnit

class TlsFavServiceStartupTest extends KafkaFeatureTest {

private val inputTopic =

kafkaTopic(UnKeyedSerde, ScalaSerdes.Thrift[ContextualizedFavoriteEvent], name = "source")

private val outputTopic =

kafkaTopic(UnKeyedSerde, ScalaSerdes.Thrift[UnifiedUserAction], name = "sink")

val startupFlags = Map(

"kafka.group.id" -> "tls",

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

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

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

"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 TlsFavsService() {

override def warmup(): Unit = {

// noop

}

override val overrideModules = Seq(

KafkaProcessorTlsFavsModule

)

},

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(UnKeyedSerde.deserializer)

.valueDeserializer(ScalaSerdes.Thrift[ContextualizedFavoriteEvent].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(UnKeyedSerde.serializer)

.valueSerializer(ScalaSerdes.Thrift[ContextualizedFavoriteEvent].serializer)

.build()

}

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[UnifiedUserAction].deserializer)

.requestTimeout(Duration.fromSeconds(1))

.enableAutoCommit(false)

.seekStrategy(seekStrategy)

new KafkaConsumerClient(builder.config)

}

test("TlsFavService starts") {

server.assertHealthy()

}

test("TlsFavService should process input events") {

val producer = getProducer()

val inputConsumer = getConsumer()

val uuaConsumer = getUUAConsumer()

val favoriteEvent = FavoriteEventUnion.Favorite(FavoriteEvent(123L, 123L, 123L, 123L))

val value =

ContextualizedFavoriteEvent(favoriteEvent, LogEventContext("localhost", 123L))

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, new UnKeyed, 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 TlsFavsService 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()

}

}

}