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

import com.twitter.conversions.StorageUnitOps.\_

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.producers.BlockingFinagleKafkaProducer

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

import com.twitter.kafka.client.processor.ThreadSafeKafkaConsumerClient

import com.twitter.util.logging.Logging

import com.twitter.util.Duration

import com.twitter.util.StorageUnit

import org.apache.kafka.clients.CommonClientConfigs

import org.apache.kafka.clients.producer.ProducerConfig

import org.apache.kafka.common.config.SaslConfigs

import org.apache.kafka.common.config.SslConfigs

import org.apache.kafka.common.record.CompressionType

import org.apache.kafka.common.security.auth.SecurityProtocol

import org.apache.kafka.common.serialization.Deserializer

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

/\*\*

\* A Utility class mainly provides raw Kafka producer/consumer supports

\*/

object ClientProviders extends Logging {

/\*\*

\* Provide a Finagle-thread-safe-and-compatible Kafka consumer.

\* For the params and their significance, please see [[ClientConfigs]]

\*/

def mkConsumer[CK, CV](

bootstrapServer: String,

keySerde: Deserializer[CK],

valueSerde: Deserializer[CV],

groupId: String,

autoCommit: Boolean = false,

maxPollRecords: Int = ClientConfigs.consumerMaxPollRecordsDefault,

maxPollInterval: Duration = ClientConfigs.consumerMaxPollIntervalDefault,

autoCommitInterval: Duration = ClientConfigs.kafkaCommitIntervalDefault,

sessionTimeout: Duration = ClientConfigs.consumerSessionTimeoutDefault,

fetchMax: StorageUnit = ClientConfigs.consumerFetchMaxDefault,

fetchMin: StorageUnit = ClientConfigs.consumerFetchMinDefault,

receiveBuffer: StorageUnit = ClientConfigs.consumerReceiveBufferSizeDefault,

trustStoreLocationOpt: Option[String] = Some(ClientConfigs.trustStoreLocationDefault)

): ThreadSafeKafkaConsumerClient[CK, CV] = {

val baseBuilder =

FinagleKafkaConsumerBuilder[CK, CV]()

.keyDeserializer(keySerde)

.valueDeserializer(valueSerde)

.dest(bootstrapServer)

.groupId(KafkaGroupId(groupId))

.enableAutoCommit(autoCommit)

.maxPollRecords(maxPollRecords)

.maxPollInterval(maxPollInterval)

.autoCommitInterval(autoCommitInterval)

.receiveBuffer(receiveBuffer)

.sessionTimeout(sessionTimeout)

.fetchMax(fetchMax)

.fetchMin(fetchMin)

.withConfig(

CommonClientConfigs.SECURITY\_PROTOCOL\_CONFIG,

SecurityProtocol.PLAINTEXT.toString)

trustStoreLocationOpt

.map { trustStoreLocation =>

new ThreadSafeKafkaConsumerClient[CK, CV](

baseBuilder

.withConfig(

CommonClientConfigs.SECURITY\_PROTOCOL\_CONFIG,

SecurityProtocol.SASL\_SSL.toString)

.withConfig(SslConfigs.SSL\_TRUSTSTORE\_LOCATION\_CONFIG, trustStoreLocation)

.withConfig(SaslConfigs.SASL\_MECHANISM, SaslConfigs.GSSAPI\_MECHANISM)

.withConfig(SaslConfigs.SASL\_KERBEROS\_SERVICE\_NAME, "kafka")

.withConfig(SaslConfigs.SASL\_KERBEROS\_SERVER\_NAME, "kafka")

.config)

}.getOrElse {

new ThreadSafeKafkaConsumerClient[CK, CV](

baseBuilder

.withConfig(

CommonClientConfigs.SECURITY\_PROTOCOL\_CONFIG,

SecurityProtocol.PLAINTEXT.toString)

.config)

}

}

/\*\*

\* Provide a Finagle-compatible Kafka producer.

\* For the params and their significance, please see [[ClientConfigs]]

\*/

def mkProducer[PK, PV](

bootstrapServer: String,

keySerde: Serializer[PK],

valueSerde: Serializer[PV],

clientId: String,

idempotence: Boolean = ClientConfigs.producerIdempotenceDefault,

batchSize: StorageUnit = ClientConfigs.producerBatchSizeDefault,

linger: Duration = ClientConfigs.producerLingerDefault,

bufferMem: StorageUnit = ClientConfigs.producerBufferMemDefault,

compressionType: CompressionType = ClientConfigs.compressionDefault.compressionType,

retries: Int = ClientConfigs.retriesDefault,

retryBackoff: Duration = ClientConfigs.retryBackoffDefault,

requestTimeout: Duration = ClientConfigs.producerRequestTimeoutDefault,

trustStoreLocationOpt: Option[String] = Some(ClientConfigs.trustStoreLocationDefault)

): BlockingFinagleKafkaProducer[PK, PV] = {

val baseBuilder = FinagleKafkaProducerBuilder[PK, PV]()

.keySerializer(keySerde)

.valueSerializer(valueSerde)

.dest(bootstrapServer)

.clientId(clientId)

.batchSize(batchSize)

.linger(linger)

.bufferMemorySize(bufferMem)

.maxRequestSize(4.megabytes)

.compressionType(compressionType)

.enableIdempotence(idempotence)

.ackMode(AckMode.ALL)

.maxInFlightRequestsPerConnection(5)

.retries(retries)

.retryBackoff(retryBackoff)

.requestTimeout(requestTimeout)

.withConfig(ProducerConfig.DELIVERY\_TIMEOUT\_MS\_CONFIG, requestTimeout + linger)

trustStoreLocationOpt

.map { trustStoreLocation =>

baseBuilder

.withConfig(

CommonClientConfigs.SECURITY\_PROTOCOL\_CONFIG,

SecurityProtocol.SASL\_SSL.toString)

.withConfig(SslConfigs.SSL\_TRUSTSTORE\_LOCATION\_CONFIG, trustStoreLocation)

.withConfig(SaslConfigs.SASL\_MECHANISM, SaslConfigs.GSSAPI\_MECHANISM)

.withConfig(SaslConfigs.SASL\_KERBEROS\_SERVICE\_NAME, "kafka")

.withConfig(SaslConfigs.SASL\_KERBEROS\_SERVER\_NAME, "kafka")

.build()

}.getOrElse {

baseBuilder

.withConfig(

CommonClientConfigs.SECURITY\_PROTOCOL\_CONFIG,

SecurityProtocol.PLAINTEXT.toString)

.build()

}

}

}