package com.twitter.simclusters\_v2.summingbird.stores

import com.twitter.bijection.Injection

import com.twitter.finagle.stats.NullStatsReceiver

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.io.Buf

import com.twitter.scrooge.ThriftStruct

import com.twitter.simclusters\_v2.common.TweetId

import com.twitter.simclusters\_v2.summingbird.stores.PersistentTweetEmbeddingStore.Timestamp

import com.twitter.simclusters\_v2.thriftscala.PersistentSimClustersEmbedding

import com.twitter.storage.client.manhattan.kv.Guarantee

import com.twitter.storage.client.manhattan.kv.ManhattanKVClient

import com.twitter.storage.client.manhattan.kv.ManhattanKVClientMtlsParams

import com.twitter.storage.client.manhattan.kv.ManhattanKVEndpointBuilder

import com.twitter.storage.client.manhattan.kv.impl.FullBufKey

import com.twitter.storage.client.manhattan.kv.impl.ValueDescriptor

import com.twitter.storehaus.ReadableStore

import com.twitter.storehaus\_internal.manhattan\_kv.ManhattanEndpointStore

import com.twitter.strato.catalog.Version

import com.twitter.strato.config.MValEncoding

import com.twitter.strato.config.NativeEncoding

import com.twitter.strato.config.PkeyLkey2

import com.twitter.strato.data.Conv

import com.twitter.strato.data.Type

import com.twitter.strato.mh.ManhattanInjections

import com.twitter.strato.thrift.ScroogeConv

import com.twitter.strato.thrift.ScroogeConvImplicits.\_

object ManhattanFromStratoStore {

/\* This enables reading from a MH store where the data is written by Strato. Strato uses a unique

encoding (Conv) which needs to be reconstructed for each MH store based on the type of data that

is written to it. Once that encoding is generated on start-up, we can read from the store like

any other ReadableStore.

\*/

def createPersistentTweetStore(

dataset: String,

mhMtlsParams: ManhattanKVClientMtlsParams,

statsReceiver: StatsReceiver = NullStatsReceiver

): ReadableStore[(TweetId, Timestamp), PersistentSimClustersEmbedding] = {

val appId = "simclusters\_embeddings\_prod"

val dest = "/s/manhattan/omega.native-thrift"

val endpoint = createMhEndpoint(

appId = appId,

dest = dest,

mhMtlsParams = mhMtlsParams,

statsReceiver = statsReceiver)

val (

keyInj: Injection[(TweetId, Timestamp), FullBufKey],

valueDesc: ValueDescriptor.EmptyValue[PersistentSimClustersEmbedding]) =

injectionsFromPkeyLkeyValueStruct[TweetId, Timestamp, PersistentSimClustersEmbedding](

dataset = dataset,

pkType = Type.Long,

lkType = Type.Long)

ManhattanEndpointStore

.readable[(TweetId, Timestamp), PersistentSimClustersEmbedding, FullBufKey](

endpoint = endpoint,

keyDescBuilder = keyInj,

emptyValDesc = valueDesc)

}

private def createMhEndpoint(

appId: String,

dest: String,

mhMtlsParams: ManhattanKVClientMtlsParams,

statsReceiver: StatsReceiver = NullStatsReceiver

) = {

val mhc = ManhattanKVClient.memoizedByDest(

appId = appId,

dest = dest,

mtlsParams = mhMtlsParams

)

ManhattanKVEndpointBuilder(mhc)

.defaultGuarantee(Guarantee.SoftDcReadMyWrites)

.statsReceiver(statsReceiver)

.build()

}

private def injectionsFromPkeyLkeyValueStruct[PK: Conv, LK: Conv, V <: ThriftStruct: Manifest](

dataset: String,

pkType: Type,

lkType: Type

): (Injection[(PK, LK), FullBufKey], ValueDescriptor.EmptyValue[V]) = {

// Strato uses a unique encoding (Conv) so we need to rebuild that based on the pkey, lkey and

// value type before converting it to the Manhattan injections for key -> FullBufKey and

// value -> Buf

val valueConv: Conv[V] = ScroogeConv.fromStruct[V]

val mhEncodingMapping = PkeyLkey2(

pkey = pkType,

lkey = lkType,

value = valueConv.t,

pkeyEncoding = NativeEncoding,

lkeyEncoding = NativeEncoding,

valueEncoding = MValEncoding()

)

val (keyInj: Injection[(PK, LK), FullBufKey], valueInj: Injection[V, Buf], \_, \_) =

ManhattanInjections.fromPkeyLkey[PK, LK, V](mhEncodingMapping, dataset, Version.Default)

val valDesc: ValueDescriptor.EmptyValue[V] = ValueDescriptor.EmptyValue(valueInj)

(keyInj, valDesc)

}

}