package com.twitter.tweetypie.storage

import com.google.common.base.CaseFormat

import com.twitter.conversions.DurationOps.\_

import com.twitter.finagle.mtls.authentication.ServiceIdentifier

import com.twitter.scrooge.TFieldBlob

import com.twitter.scrooge.ThriftStructFieldInfo

import com.twitter.stitch.Stitch

import com.twitter.storage.client.manhattan.kv.\_

import com.twitter.tweetypie.additionalfields.AdditionalFields

import com.twitter.tweetypie.storage.ManhattanOperations.Read

import com.twitter.tweetypie.storage.TweetUtils.\_

import com.twitter.tweetypie.storage\_internal.thriftscala.StoredTweet

import com.twitter.tweetypie.thriftscala.{Tweet => TweetypieTweet}

import com.twitter.util.Duration

import com.twitter.util.Future

import com.twitter.util.Return

import com.twitter.util.Throw

import diffshow.Container

import diffshow.DiffShow

import diffshow.Expr

import org.apache.commons.codec.binary.Base64

import scala.util.Try

import shapeless.Cached

import shapeless.Strict

// This class is used by the Tweetypie Console to inspect tweet field content in Manhattan

class InspectFields(svcIdentifier: ServiceIdentifier) {

val mhApplicationId = "tbird\_mh"

val mhDatasetName = "tbird\_mh"

val mhDestinationName = "/s/manhattan/cylon.native-thrift"

val mhTimeout: Duration = 5000.milliseconds

val localMhEndpoint: ManhattanKVEndpoint =

ManhattanKVEndpointBuilder(

ManhattanKVClient(

mhApplicationId,

mhDestinationName,

ManhattanKVClientMtlsParams(svcIdentifier)))

.defaultGuarantee(Guarantee.SoftDcReadMyWrites)

.defaultMaxTimeout(mhTimeout)

.build()

val readOperation: Read = (new ManhattanOperations(mhDatasetName, localMhEndpoint)).read

def lookup(tweetId: Long): Future[String] = {

val result = readOperation(tweetId).liftToTry.map {

case Return(mhRecords) =>

prettyPrintManhattanRecords(tweetId, TweetKey.padTweetIdStr(tweetId), mhRecords)

case Throw(e) => e.toString

}

Stitch.run(result)

}

def storedTweet(tweetId: Long): Future[StoredTweet] = {

val result = readOperation(tweetId).liftToTry.map {

case Return(mhRecords) =>

buildStoredTweet(tweetId, mhRecords)

case Throw(e) =>

throw e

}

Stitch.run(result)

}

private[this] def prettyPrintManhattanRecords(

tweetId: Long,

pkey: String,

mhRecords: Seq[TweetManhattanRecord]

): String = {

if (mhRecords.isEmpty) {

"Not Found"

} else {

val formattedRecords = getFormattedManhattanRecords(tweetId, mhRecords)

val keyFieldWidth = formattedRecords.map(\_.key.length).max + 2

val fieldNameFieldWidth = formattedRecords.map(\_.fieldName.length).max + 2

val formatString = s" %-${keyFieldWidth}s %-${fieldNameFieldWidth}s %s"

val recordsString =

formattedRecords

.map { record =>

val content = record.content.replaceAll("\n", "\n" + formatString.format("", "", ""))

formatString.format(record.key, record.fieldName, content)

}

.mkString("\n")

"/tbird\_mh/" + pkey + "/" + "\n" + recordsString

}

}

private[this] def getFormattedManhattanRecords(

tweetId: Long,

mhRecords: Seq[TweetManhattanRecord]

): Seq[FormattedManhattanRecord] = {

val storedTweet = buildStoredTweet(tweetId, mhRecords).copy(updatedAt = None)

val tweetypieTweet: Option[TweetypieTweet] =

Try(StorageConversions.fromStoredTweet(storedTweet)).toOption

val blobMap: Map[String, TFieldBlob] = getStoredTweetBlobs(mhRecords).map { blob =>

getFieldName(blob.field.id) -> blob

}.toMap

mhRecords

.map {

case TweetManhattanRecord(fullKey, mhValue) =>

FormattedManhattanRecord(

key = fullKey.lKey.toString,

fieldName = getFieldName(fullKey.lKey),

content = prettyPrintManhattanValue(

fullKey.lKey,

mhValue,

storedTweet,

tweetypieTweet,

tweetId,

blobMap

)

)

}

.sortBy(\_.key.replace("external", "xternal")) // sort by key, with internal first

}

private[this] def getFieldNameFromThrift(

fieldId: Short,

fieldInfos: List[ThriftStructFieldInfo]

): String =

fieldInfos

.find(info => info.tfield.id == fieldId)

.map(\_.tfield.name)

.getOrElse("<UNKNOWN FIELD>")

private[this] def isLkeyScrubbedField(lkey: String): Boolean =

lkey.split("/")(1) == "scrubbed\_fields"

private[this] def getFieldName(lkey: TweetKey.LKey): String =

lkey match {

case fieldKey: TweetKey.LKey.FieldKey => getFieldName(fieldKey.fieldId)

case \_ => ""

}

private[this] def getFieldName(fieldId: Short): String =

if (fieldId == 1) {

"core\_fields"

} else if (AdditionalFields.isAdditionalFieldId(fieldId)) {

getFieldNameFromThrift(fieldId, TweetypieTweet.fieldInfos)

} else {

getFieldNameFromThrift(fieldId, StoredTweet.fieldInfos)

}

private[this] def prettyPrintManhattanValue(

lkey: TweetKey.LKey,

mhValue: TweetManhattanValue,

storedTweet: StoredTweet,

tweetypieTweet: Option[TweetypieTweet],

tweetId: Long,

tfieldBlobs: Map[String, TFieldBlob]

): String = {

val decoded = lkey match {

case \_: TweetKey.LKey.MetadataKey =>

decodeMetadata(mhValue)

case fieldKey: TweetKey.LKey.FieldKey =>

tfieldBlobs

.get(getFieldName(fieldKey.fieldId))

.map(blob => decodeField(tweetId, blob, storedTweet, tweetypieTweet))

case \_ =>

None

}

decoded.getOrElse { // If all else fails, encode the data as a base64 string

val contents = mhValue.contents.array

if (contents.isEmpty) {

"<NO DATA>"

} else {

Base64.encodeBase64String(contents)

}

}

}

private[this] def decodeMetadata(mhValue: TweetManhattanValue): Option[String] = {

val byteArray = ByteArrayCodec.fromByteBuffer(mhValue.contents)

Try(Json.decode(byteArray).toString).toOption

}

private[this] def decodeField(

tweetId: Long,

blob: TFieldBlob,

storedTweet: StoredTweet,

tweetypieTweet: Option[TweetypieTweet]

): String = {

val fieldId = blob.field.id

if (fieldId == 1) {

coreFields(storedTweet)

} else if (AdditionalFields.isAdditionalFieldId(fieldId)) {

decodeTweetWithOneField(TweetypieTweet(tweetId).setField(blob))

} else {

decodeTweetWithOneField(StoredTweet(tweetId).setField(blob))

}

}

// Takes a Tweet or StoredTweet with a single field set and returns the value of that field

private[this] def decodeTweetWithOneField[T](

tweetWithOneField: T

)(

implicit ev: Cached[Strict[DiffShow[T]]]

): String = {

val config = diffshow.Config(hideFieldWithEmptyVal = true)

val tree: Expr = config.transform(DiffShow.show(tweetWithOneField))

// matches a Tweet or StoredTweet with two values, the first being the id

val value = tree.transform {

case Container(\_, List(diffshow.Field("id", \_), diffshow.Field(\_, value))) => value

}

config.exprPrinter.apply(value, width = 80).render

}

private[this] def coreFields(storedTweet: StoredTweet): String =

diffshow.show(CoreFieldsCodec.fromTweet(storedTweet), hideFieldWithEmptyVal = true)

private[this] def toCamelCase(s: String): String =

CaseFormat.LOWER\_UNDERSCORE.to(CaseFormat.LOWER\_CAMEL, s)

}

case class FormattedManhattanRecord(key: String, fieldName: String, content: String)