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

import com.twitter.bijection.{Bufferable, Injection}

import com.twitter.bijection.scrooge.CompactScalaCodec

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

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

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

import com.twitter.storehaus.ReadableStore

import com.twitter.storehaus\_internal.manhattan.{Athena, ManhattanRO, ManhattanROConfig}

import com.twitter.storehaus\_internal.util.{ApplicationID, DatasetName, HDFSPath}

import com.twitter.util.{Future, Memoize}

object ClusterDetailsReadableStore {

val modelVersionToDatasetMap: Map[String, String] = Map(

ModelVersions.Model20M145KDec11 -> "simclusters\_v2\_cluster\_details",

ModelVersions.Model20M145KUpdated -> "simclusters\_v2\_cluster\_details\_20m\_145k\_updated",

ModelVersions.Model20M145K2020 -> "simclusters\_v2\_cluster\_details\_20m\_145k\_2020"

)

val knownModelVersions: String = modelVersionToDatasetMap.keys.mkString(",")

private val clusterDetailsStores =

Memoize[(ManhattanKVClientMtlsParams, String), ReadableStore[(String, Int), ClusterDetails]] {

case (mhMtlsParams: ManhattanKVClientMtlsParams, datasetName: String) =>

getForDatasetName(mhMtlsParams, datasetName)

}

def getForDatasetName(

mhMtlsParams: ManhattanKVClientMtlsParams,

datasetName: String

): ReadableStore[(String, Int), ClusterDetails] = {

implicit val keyInjection: Injection[(String, Int), Array[Byte]] =

Bufferable.injectionOf[(String, Int)]

implicit val valueInjection: Injection[ClusterDetails, Array[Byte]] =

CompactScalaCodec(ClusterDetails)

ManhattanRO.getReadableStoreWithMtls[(String, Int), ClusterDetails](

ManhattanROConfig(

HDFSPath(""), // not needed

ApplicationID("simclusters\_v2"),

DatasetName(datasetName), // this should be correct

Athena

),

mhMtlsParams

)

}

def apply(

mhMtlsParams: ManhattanKVClientMtlsParams

): ReadableStore[(String, Int), ClusterDetails] = {

new ReadableStore[(String, Int), ClusterDetails] {

override def get(modelVersionAndClusterId: (String, Int)): Future[Option[ClusterDetails]] = {

val (modelVersion, \_) = modelVersionAndClusterId

modelVersionToDatasetMap.get(modelVersion) match {

case Some(datasetName) =>

clusterDetailsStores((mhMtlsParams, datasetName)).get(modelVersionAndClusterId)

case None =>

Future.exception(

new IllegalArgumentException(

"Unknown model version " + modelVersion + ". Known modelVersions: " + knownModelVersions)

)

}

}

}

}

}