package com.twitter.cr\_mixer.module

import com.google.inject.Provides

import com.twitter.bijection.Injection

import com.twitter.bijection.scrooge.BinaryScalaCodec

import com.twitter.bijection.scrooge.CompactScalaCodec

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

import com.twitter.inject.TwitterModule

import com.twitter.ml.api.{thriftscala => api}

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

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

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

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

import com.twitter.storehaus.ReadableStore

import com.twitter.storehaus\_internal.manhattan.Apollo

import com.twitter.storehaus\_internal.manhattan.ManhattanRO

import com.twitter.storehaus\_internal.manhattan.ManhattanROConfig

import com.twitter.storehaus\_internal.util.ApplicationID

import com.twitter.storehaus\_internal.util.DatasetName

import com.twitter.storehaus\_internal.util.HDFSPath

import javax.inject.Named

import javax.inject.Singleton

object EmbeddingStoreModule extends TwitterModule {

type UserId = Long

implicit val mbcgUserEmbeddingInjection: Injection[api.Embedding, Array[Byte]] =

CompactScalaCodec(api.Embedding)

implicit val tweetCandidatesInjection: Injection[CandidateTweetsList, Array[Byte]] =

CompactScalaCodec(CandidateTweetsList)

final val TwHINEmbeddingRegularUpdateMhStoreName = "TwHINEmbeddingRegularUpdateMhStore"

@Provides

@Singleton

@Named(TwHINEmbeddingRegularUpdateMhStoreName)

def twHINEmbeddingRegularUpdateMhStore(

serviceIdentifier: ServiceIdentifier

): ReadableStore[InternalId, api.Embedding] = {

val binaryEmbeddingInjection: Injection[api.Embedding, Array[Byte]] =

BinaryScalaCodec(api.Embedding)

val longCodec = implicitly[Injection[Long, Array[Byte]]]

ManhattanRO

.getReadableStoreWithMtls[TweetId, api.Embedding](

ManhattanROConfig(

HDFSPath(""), // not needed

ApplicationID("cr\_mixer\_apollo"),

DatasetName("twhin\_regular\_update\_tweet\_embedding\_apollo"),

Apollo

),

ManhattanKVClientMtlsParams(serviceIdentifier)

)(longCodec, binaryEmbeddingInjection).composeKeyMapping[InternalId] {

case InternalId.TweetId(tweetId) =>

tweetId

case \_ =>

throw new UnsupportedOperationException("Invalid Internal Id")

}

}

final val ConsumerBasedTwHINEmbeddingRegularUpdateMhStoreName =

"ConsumerBasedTwHINEmbeddingRegularUpdateMhStore"

@Provides

@Singleton

@Named(ConsumerBasedTwHINEmbeddingRegularUpdateMhStoreName)

def consumerBasedTwHINEmbeddingRegularUpdateMhStore(

serviceIdentifier: ServiceIdentifier

): ReadableStore[InternalId, api.Embedding] = {

val binaryEmbeddingInjection: Injection[api.Embedding, Array[Byte]] =

BinaryScalaCodec(api.Embedding)

val longCodec = implicitly[Injection[Long, Array[Byte]]]

ManhattanRO

.getReadableStoreWithMtls[UserId, api.Embedding](

ManhattanROConfig(

HDFSPath(""), // not needed

ApplicationID("cr\_mixer\_apollo"),

DatasetName("twhin\_user\_embedding\_regular\_update\_apollo"),

Apollo

),

ManhattanKVClientMtlsParams(serviceIdentifier)

)(longCodec, binaryEmbeddingInjection).composeKeyMapping[InternalId] {

case InternalId.UserId(userId) =>

userId

case \_ =>

throw new UnsupportedOperationException("Invalid Internal Id")

}

}

final val TwoTowerFavConsumerEmbeddingMhStoreName = "TwoTowerFavConsumerEmbeddingMhStore"

@Provides

@Singleton

@Named(TwoTowerFavConsumerEmbeddingMhStoreName)

def twoTowerFavConsumerEmbeddingMhStore(

serviceIdentifier: ServiceIdentifier

): ReadableStore[InternalId, api.Embedding] = {

val binaryEmbeddingInjection: Injection[api.Embedding, Array[Byte]] =

BinaryScalaCodec(api.Embedding)

val longCodec = implicitly[Injection[Long, Array[Byte]]]

ManhattanRO

.getReadableStoreWithMtls[UserId, api.Embedding](

ManhattanROConfig(

HDFSPath(""), // not needed

ApplicationID("cr\_mixer\_apollo"),

DatasetName("two\_tower\_fav\_user\_embedding\_apollo"),

Apollo

),

ManhattanKVClientMtlsParams(serviceIdentifier)

)(longCodec, binaryEmbeddingInjection).composeKeyMapping[InternalId] {

case InternalId.UserId(userId) =>

userId

case \_ =>

throw new UnsupportedOperationException("Invalid Internal Id")

}

}

final val DebuggerDemoUserEmbeddingMhStoreName = "DebuggerDemoUserEmbeddingMhStoreName"

@Provides

@Singleton

@Named(DebuggerDemoUserEmbeddingMhStoreName)

def debuggerDemoUserEmbeddingStore(

serviceIdentifier: ServiceIdentifier

): ReadableStore[InternalId, api.Embedding] = {

// This dataset is from src/scala/com/twitter/wtf/beam/bq\_embedding\_export/sql/MlfExperimentalUserEmbeddingScalaDataset.sql

// Change the above sql if you want to use a diff embedding

val manhattanROConfig = ManhattanROConfig(

HDFSPath(""), // not needed

ApplicationID("cr\_mixer\_apollo"),

DatasetName("experimental\_user\_embedding"),

Apollo

)

buildUserEmbeddingStore(serviceIdentifier, manhattanROConfig)

}

final val DebuggerDemoTweetEmbeddingMhStoreName = "DebuggerDemoTweetEmbeddingMhStore"

@Provides

@Singleton

@Named(DebuggerDemoTweetEmbeddingMhStoreName)

def debuggerDemoTweetEmbeddingStore(

serviceIdentifier: ServiceIdentifier

): ReadableStore[InternalId, api.Embedding] = {

// This dataset is from src/scala/com/twitter/wtf/beam/bq\_embedding\_export/sql/MlfExperimentalTweetEmbeddingScalaDataset.sql

// Change the above sql if you want to use a diff embedding

val manhattanROConfig = ManhattanROConfig(

HDFSPath(""), // not needed

ApplicationID("cr\_mixer\_apollo"),

DatasetName("experimental\_tweet\_embedding"),

Apollo

)

buildTweetEmbeddingStore(serviceIdentifier, manhattanROConfig)

}

private def buildUserEmbeddingStore(

serviceIdentifier: ServiceIdentifier,

manhattanROConfig: ManhattanROConfig

): ReadableStore[InternalId, api.Embedding] = {

val binaryEmbeddingInjection: Injection[api.Embedding, Array[Byte]] =

BinaryScalaCodec(api.Embedding)

val longCodec = implicitly[Injection[Long, Array[Byte]]]

ManhattanRO

.getReadableStoreWithMtls[UserId, api.Embedding](

manhattanROConfig,

ManhattanKVClientMtlsParams(serviceIdentifier)

)(longCodec, binaryEmbeddingInjection).composeKeyMapping[InternalId] {

case InternalId.UserId(userId) =>

userId

case \_ =>

throw new UnsupportedOperationException("Invalid Internal Id")

}

}

private def buildTweetEmbeddingStore(

serviceIdentifier: ServiceIdentifier,

manhattanROConfig: ManhattanROConfig

): ReadableStore[InternalId, api.Embedding] = {

val binaryEmbeddingInjection: Injection[api.Embedding, Array[Byte]] =

BinaryScalaCodec(api.Embedding)

val longCodec = implicitly[Injection[Long, Array[Byte]]]

ManhattanRO

.getReadableStoreWithMtls[TweetId, api.Embedding](

manhattanROConfig,

ManhattanKVClientMtlsParams(serviceIdentifier)

)(longCodec, binaryEmbeddingInjection).composeKeyMapping[InternalId] {

case InternalId.TweetId(tweetId) =>

tweetId

case \_ =>

throw new UnsupportedOperationException("Invalid Internal Id")

}

}

}