package com.twitter.tsp.modules

import com.google.inject.Module

import com.google.inject.Provides

import com.google.inject.Singleton

import com.twitter.bijection.scrooge.BinaryScalaCodec

import com.twitter.conversions.DurationOps.\_

import com.twitter.finagle.memcached.{Client => MemClient}

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

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.store.health.TweetHealthModelStore

import com.twitter.frigate.common.store.health.TweetHealthModelStore.TweetHealthModelStoreConfig

import com.twitter.frigate.common.store.health.UserHealthModelStore

import com.twitter.frigate.common.store.interests.UserId

import com.twitter.frigate.thriftscala.TweetHealthScores

import com.twitter.frigate.thriftscala.UserAgathaScores

import com.twitter.hermit.store.common.DeciderableReadableStore

import com.twitter.hermit.store.common.ObservedCachedReadableStore

import com.twitter.hermit.store.common.ObservedMemcachedReadableStore

import com.twitter.inject.TwitterModule

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

import com.twitter.stitch.tweetypie.TweetyPie

import com.twitter.storehaus.ReadableStore

import com.twitter.strato.client.{Client => StratoClient}

import com.twitter.tsp.common.DeciderKey

import com.twitter.tsp.common.TopicSocialProofDecider

import com.twitter.tsp.stores.TweetInfoStore

import com.twitter.tsp.stores.TweetyPieFieldsStore

import com.twitter.tweetypie.thriftscala.TweetService

import com.twitter.tsp.thriftscala.TspTweetInfo

import com.twitter.util.JavaTimer

import com.twitter.util.Timer

object TweetInfoStoreModule extends TwitterModule {

override def modules: Seq[Module] = Seq(UnifiedCacheClient)

implicit val timer: Timer = new JavaTimer(true)

@Provides

@Singleton

def providesTweetInfoStore(

decider: TopicSocialProofDecider,

serviceIdentifier: ServiceIdentifier,

statsReceiver: StatsReceiver,

stratoClient: StratoClient,

tspUnifiedCacheClient: MemClient,

tweetyPieService: TweetService.MethodPerEndpoint

): ReadableStore[TweetId, TspTweetInfo] = {

val tweetHealthModelStore: ReadableStore[TweetId, TweetHealthScores] = {

val underlyingStore = TweetHealthModelStore.buildReadableStore(

stratoClient,

Some(

TweetHealthModelStoreConfig(

enablePBlock = true,

enableToxicity = true,

enablePSpammy = true,

enablePReported = true,

enableSpammyTweetContent = true,

enablePNegMultimodal = false))

)(statsReceiver.scope("UnderlyingTweetHealthModelStore"))

DeciderableReadableStore(

ObservedMemcachedReadableStore.fromCacheClient(

backingStore = underlyingStore,

cacheClient = tspUnifiedCacheClient,

ttl = 2.hours

)(

valueInjection = BinaryScalaCodec(TweetHealthScores),

statsReceiver = statsReceiver.scope("TweetHealthModelStore"),

keyToString = { k: TweetId => s"tHMS/$k" }

),

decider.deciderGateBuilder.idGate(DeciderKey.enableHealthSignalsScoreDeciderKey),

statsReceiver.scope("TweetHealthModelStore")

)

}

val userHealthModelStore: ReadableStore[UserId, UserAgathaScores] = {

val underlyingStore =

UserHealthModelStore.buildReadableStore(stratoClient)(

statsReceiver.scope("UnderlyingUserHealthModelStore"))

DeciderableReadableStore(

ObservedMemcachedReadableStore.fromCacheClient(

backingStore = underlyingStore,

cacheClient = tspUnifiedCacheClient,

ttl = 18.hours

)(

valueInjection = BinaryScalaCodec(UserAgathaScores),

statsReceiver = statsReceiver.scope("UserHealthModelStore"),

keyToString = { k: UserId => s"uHMS/$k" }

),

decider.deciderGateBuilder.idGate(DeciderKey.enableUserAgathaScoreDeciderKey),

statsReceiver.scope("UserHealthModelStore")

)

}

val tweetInfoStore: ReadableStore[TweetId, TspTweetInfo] = {

val underlyingStore = TweetInfoStore(

TweetyPieFieldsStore.getStoreFromTweetyPie(TweetyPie(tweetyPieService, statsReceiver)),

tweetHealthModelStore: ReadableStore[TweetId, TweetHealthScores],

userHealthModelStore: ReadableStore[UserId, UserAgathaScores],

timer: Timer

)(statsReceiver.scope("tweetInfoStore"))

val memcachedStore = ObservedMemcachedReadableStore.fromCacheClient(

backingStore = underlyingStore,

cacheClient = tspUnifiedCacheClient,

ttl = 15.minutes,

// Hydrating tweetInfo is now a required step for all candidates,

// hence we needed to tune these thresholds.

asyncUpdate = serviceIdentifier.environment == "prod"

)(

valueInjection = BinaryScalaCodec(TspTweetInfo),

statsReceiver = statsReceiver.scope("memCachedTweetInfoStore"),

keyToString = { k: TweetId => s"tIS/$k" }

)

val inMemoryStore = ObservedCachedReadableStore.from(

memcachedStore,

ttl = 15.minutes,

maxKeys = 8388607, // Check TweetInfo definition. size~92b. Around 736 MB

windowSize = 10000L,

cacheName = "tweet\_info\_cache",

maxMultiGetSize = 20

)(statsReceiver.scope("inMemoryCachedTweetInfoStore"))

inMemoryStore

}

tweetInfoStore

}

}