package com.twitter.home\_mixer.module

import com.google.inject.Provides

import com.twitter.conversions.DurationOps.\_

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

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.inject.TwitterModule

import com.twitter.inject.annotations.Flag

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

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

import com.twitter.timelines.clients.manhattan.store.\_

import com.twitter.timelines.impressionbloomfilter.{thriftscala => blm}

import com.twitter.timelines.impressionstore.impressionbloomfilter.ImpressionBloomFilterManhattanKeyValueDescriptor

import com.twitter.util.Duration

import javax.inject.Singleton

object ImpressionBloomFilterModule extends TwitterModule {

private val ProdAppId = "impression\_bloom\_filter\_store"

private val ProdDataset = "impression\_bloom\_filter"

private val StagingAppId = "impression\_bloom\_filter\_store\_staging"

private val StagingDataset = "impression\_bloom\_filter\_staging"

private val ClientStatsScope = "tweetBloomFilterImpressionManhattanClient"

private val DefaultTTL = 7.days

private final val Timeout = "mh\_impression\_store\_bloom\_filter.timeout"

flag[Duration](Timeout, 150.millis, "Timeout per request")

@Provides

@Singleton

def providesImpressionBloomFilter(

@Flag(Timeout) timeout: Duration,

serviceIdentifier: ServiceIdentifier,

statsReceiver: StatsReceiver

): ManhattanStoreClient[blm.ImpressionBloomFilterKey, blm.ImpressionBloomFilterSeq] = {

val (appId, dataset) = serviceIdentifier.environment.toLowerCase match {

case "prod" => (ProdAppId, ProdDataset)

case \_ => (StagingAppId, StagingDataset)

}

implicit val manhattanKeyValueDescriptor: ImpressionBloomFilterManhattanKeyValueDescriptor =

ImpressionBloomFilterManhattanKeyValueDescriptor(

dataset = dataset,

ttl = DefaultTTL

)

ManhattanStoreClientBuilder.buildManhattanClient(

serviceIdentifier = serviceIdentifier,

cluster = ManhattanClusters.nash,

appId = appId,

defaultMaxTimeout = timeout,

maxRetryCount = 2,

defaultGuarantee = Some(Guarantee.SoftDcReadMyWrites),

isReadOnly = false,

statsScope = ClientStatsScope,

statsReceiver = statsReceiver

)

}

}