package com.twitter.visibility.interfaces.push\_service

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.stitch.Stitch

import com.twitter.stitch.tweetypie.TweetyPie.TweetyPieResult

import com.twitter.storehaus.ReadableStore

import com.twitter.logging.Logger

import com.twitter.visibility.models.SafetyLevel

class PushServiceVisibilityLibraryParity(

magicRecsV2tweetyPieStore: ReadableStore[Long, TweetyPieResult],

magicRecsAggressiveV2tweetyPieStore: ReadableStore[Long, TweetyPieResult]

)(

implicit statsReceiver: StatsReceiver) {

private val stats = statsReceiver.scope("push\_service\_vf\_parity")

private val requests = stats.counter("requests")

private val equal = stats.counter("equal")

private val notEqual = stats.counter("notEqual")

private val failures = stats.counter("failures")

private val bothAllow = stats.counter("bothAllow")

private val bothReject = stats.counter("bothReject")

private val onlyTweetypieRejects = stats.counter("onlyTweetypieRejects")

private val onlyPushServiceRejects = stats.counter("onlyPushServiceRejects")

val log = Logger.get("pushservice\_vf\_parity")

def runParityTest(

req: PushServiceVisibilityRequest,

resp: PushServiceVisibilityResponse

): Stitch[Unit] = {

requests.incr()

getTweetypieResult(req).map { tweetypieResult =>

val isSameVerdict = (tweetypieResult == resp.shouldAllow)

isSameVerdict match {

case true => equal.incr()

case false => notEqual.incr()

}

(tweetypieResult, resp.shouldAllow) match {

case (true, true) => bothAllow.incr()

case (true, false) => onlyPushServiceRejects.incr()

case (false, true) => onlyTweetypieRejects.incr()

case (false, false) => bothReject.incr()

}

resp.getDropRules.foreach { dropRule =>

stats.counter(s"rules/${dropRule.name}/requests").incr()

stats

.counter(

s"rules/${dropRule.name}/" ++ (if (isSameVerdict) "equal" else "notEqual")).incr()

}

if (!isSameVerdict) {

val dropRuleNames = resp.getDropRules.map("<<" ++ \_.name ++ ">>").mkString(",")

val safetyLevelStr = req.safetyLevel match {

case SafetyLevel.MagicRecsAggressiveV2 => "aggr"

case \_ => " "

}

log.info(

s"ttweetId:${req.tweet.id} () push:${resp.shouldAllow}, tweety:${tweetypieResult}, rules=[${dropRuleNames}] lvl=${safetyLevelStr}")

}

}

}

def getTweetypieResult(request: PushServiceVisibilityRequest): Stitch[Boolean] = {

val tweetypieStore = request.safetyLevel match {

case SafetyLevel.MagicRecsAggressiveV2 => magicRecsAggressiveV2tweetyPieStore

case \_ => magicRecsV2tweetyPieStore

}

Stitch.callFuture(

tweetypieStore.get(request.tweet.id).onFailure(\_ => failures.incr()).map(x => x.isDefined))

}

}