package com.twitter.frigate.pushservice.store

import com.twitter.finagle.stats.BroadcastStatsReceiver

import com.twitter.finagle.stats.NullStatsReceiver

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.logger.MRLogger

import com.twitter.frigate.common.store

import com.twitter.frigate.common.store.Fail

import com.twitter.frigate.common.store.IbisRequestInfo

import com.twitter.frigate.common.store.IbisResponse

import com.twitter.frigate.common.store.Sent

import com.twitter.frigate.pushservice.model.PushTypes.PushCandidate

import com.twitter.frigate.thriftscala.CommonRecommendationType

import com.twitter.ibis2.service.thriftscala.Flags

import com.twitter.ibis2.service.thriftscala.FlowControl

import com.twitter.ibis2.service.thriftscala.Ibis2Request

import com.twitter.ibis2.service.thriftscala.Ibis2Response

import com.twitter.ibis2.service.thriftscala.Ibis2ResponseStatus

import com.twitter.ibis2.service.thriftscala.Ibis2Service

import com.twitter.ibis2.service.thriftscala.NotificationNotSentCode

import com.twitter.ibis2.service.thriftscala.TargetFanoutResult.NotSentReason

import com.twitter.util.Future

trait Ibis2Store extends store.Ibis2Store {

def send(ibis2Request: Ibis2Request, candidate: PushCandidate): Future[IbisResponse]

}

case class PushIbis2Store(

ibisClient: Ibis2Service.MethodPerEndpoint

)(

implicit val statsReceiver: StatsReceiver = NullStatsReceiver)

extends Ibis2Store {

private val log = MRLogger(this.getClass.getSimpleName)

private val stats = statsReceiver.scope("ibis\_v2\_store")

private val statsByCrt = stats.scope("byCrt")

private val requestsByCrt = statsByCrt.scope("requests")

private val failuresByCrt = statsByCrt.scope("failures")

private val successByCrt = statsByCrt.scope("success")

private val statsByIbisModel = stats.scope("byIbisModel")

private val requestsByIbisModel = statsByIbisModel.scope("requests")

private val failuresByIbisModel = statsByIbisModel.scope("failures")

private val successByIbisModel = statsByIbisModel.scope("success")

private[this] def ibisSend(

ibis2Request: Ibis2Request,

commonRecommendationType: CommonRecommendationType

): Future[IbisResponse] = {

val ibisModel = ibis2Request.modelName

val bStats = if (ibis2Request.flags.getOrElse(Flags()).darkWrite.contains(true)) {

BroadcastStatsReceiver(

Seq(

stats,

stats.scope("dark\_write")

)

)

} else BroadcastStatsReceiver(Seq(stats))

bStats.counter("requests").incr()

requestsByCrt.counter(commonRecommendationType.name).incr()

requestsByIbisModel.counter(ibisModel).incr()

retry(ibisClient, ibis2Request, 3, bStats)

.map { response =>

bStats.counter(response.status.status.name).incr()

successByCrt.counter(response.status.status.name, commonRecommendationType.name).incr()

successByIbisModel.counter(response.status.status.name, ibisModel).incr()

response.status.status match {

case Ibis2ResponseStatus.SuccessWithDeliveries |

Ibis2ResponseStatus.SuccessNoDeliveries =>

IbisResponse(Sent, Some(response))

case \_ =>

IbisResponse(Fail, Some(response))

}

}

.onFailure { ex =>

bStats.counter("failures").incr()

val exceptionName = ex.getClass.getCanonicalName

bStats.scope("failures").counter(exceptionName).incr()

failuresByCrt.counter(exceptionName, commonRecommendationType.name).incr()

failuresByIbisModel.counter(exceptionName, ibisModel).incr()

}

}

private def getNotifNotSentReason(

ibis2Response: Ibis2Response

): Option[NotificationNotSentCode] = {

ibis2Response.status.fanoutResults match {

case Some(fanoutResult) =>

fanoutResult.pushResult.flatMap { pushResult =>

pushResult.results.headOption match {

case Some(NotSentReason(notSentInfo)) => Some(notSentInfo.notSentCode)

case \_ => None

}

}

case \_ => None

}

}

def send(ibis2Request: Ibis2Request, candidate: PushCandidate): Future[IbisResponse] = {

val requestWithIID = if (ibis2Request.flowControl.exists(\_.externalIid.isDefined)) {

ibis2Request

} else {

ibis2Request.copy(

flowControl = Some(

ibis2Request.flowControl

.getOrElse(FlowControl())

.copy(externalIid = Some(candidate.impressionId))

)

)

}

val commonRecommendationType = candidate.frigateNotification.commonRecommendationType

ibisSend(requestWithIID, commonRecommendationType)

.onSuccess { response =>

response.ibis2Response.foreach { ibis2Response =>

getNotifNotSentReason(ibis2Response).foreach { notifNotSentCode =>

stats.scope(ibis2Response.status.status.name).counter(s"$notifNotSentCode").incr()

}

if (ibis2Response.status.status != Ibis2ResponseStatus.SuccessWithDeliveries) {

log.warning(

s"Request dropped on ibis for ${ibis2Request.recipientSelector.recipientId}: $ibis2Response")

}

}

}

.onFailure { ex =>

log.warning(

s"Ibis Request failure: ${ex.getClass.getCanonicalName} \n For IbisRequest: $ibis2Request")

log.error(ex, ex.getMessage)

}

}

// retry request when Ibis2ResponseStatus is PreFanoutError

def retry(

ibisClient: Ibis2Service.MethodPerEndpoint,

request: Ibis2Request,

retryCount: Int,

bStats: StatsReceiver

): Future[Ibis2Response] = {

ibisClient.sendNotification(request).flatMap { response =>

response.status.status match {

case Ibis2ResponseStatus.PreFanoutError if retryCount > 0 =>

bStats.scope("requests").counter("retry").incr()

bStats.counter(response.status.status.name).incr()

retry(ibisClient, request, retryCount - 1, bStats)

case \_ =>

Future.value(response)

}

}

}

override def send(

ibis2Request: Ibis2Request,

requestInfo: IbisRequestInfo

): Future[IbisResponse] = {

ibisSend(ibis2Request, requestInfo.commonRecommendationType)

}

}

case class StagingIbis2Store(remoteIbis2Store: PushIbis2Store) extends Ibis2Store {

final def addDarkWriteFlagIbis2Request(

isTeamMember: Boolean,

ibis2Request: Ibis2Request

): Ibis2Request = {

val flags =

ibis2Request.flags.getOrElse(Flags())

val darkWrite: Boolean = !isTeamMember || flags.darkWrite.getOrElse(false)

ibis2Request.copy(flags = Some(flags.copy(darkWrite = Some(darkWrite))))

}

override def send(ibis2Request: Ibis2Request, candidate: PushCandidate): Future[IbisResponse] = {

candidate.target.isTeamMember.flatMap { isTeamMember =>

val ibis2Req = addDarkWriteFlagIbis2Request(isTeamMember, ibis2Request)

remoteIbis2Store.send(ibis2Req, candidate)

}

}

override def send(

ibis2Request: Ibis2Request,

requestInfo: IbisRequestInfo

): Future[IbisResponse] = {

requestInfo.isTeamMember.flatMap { isTeamMember =>

val ibis2Req = addDarkWriteFlagIbis2Request(isTeamMember, ibis2Request)

remoteIbis2Store.send(ibis2Req, requestInfo)

}

}

}