package com.twitter.frigate.pushservice.take

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.base.Stats.track

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.pushservice.take.sender.Ibis2Sender

import com.twitter.frigate.pushservice.take.sender.NtabSender

import com.twitter.frigate.scribe.thriftscala.NotificationScribe

import com.twitter.util.Future

import com.twitter.frigate.thriftscala.ChannelName

/\*\*

\* NotificationSender wraps up all the notification infra send logic, and serves as an abstract layer

\* between CandidateNotifier and the respective senders including ntab, ibis, which is being

\* gated with both a decider/feature switch

\*/

class NotificationSender(

ibis2Sender: Ibis2Sender,

ntabSender: NtabSender,

statsReceiver: StatsReceiver,

notificationScribe: NotificationScribe => Unit) {

private val notificationNotifierStats = statsReceiver.scope(this.getClass.getSimpleName)

private val ibis2SendLatency = notificationNotifierStats.scope("ibis2\_send")

private val loggedOutIbis2SendLatency = notificationNotifierStats.scope("logged\_out\_ibis2\_send")

private val ntabSendLatency = notificationNotifierStats.scope("ntab\_send")

private val ntabWriteThenSkipPushCounter =

notificationNotifierStats.counter("ntab\_write\_then\_skip\_push")

private val ntabWriteThenIbisSendCounter =

notificationNotifierStats.counter("ntab\_write\_then\_ibis\_send")

notificationNotifierStats.counter("ins\_dark\_traffic\_send")

private val ntabOnlyChannelSenderV3Counter =

notificationNotifierStats.counter("ntab\_only\_channel\_send\_v3")

def sendIbisDarkWrite(candidate: PushCandidate): Future[IbisResponse] = {

ibis2Sender.sendAsDarkWrite(candidate)

}

private def isNtabOnlySend(

channels: Seq[ChannelName]

): Future[Boolean] = {

val isNtabOnlyChannel = channels.contains(ChannelName.NtabOnly)

if (isNtabOnlyChannel) ntabOnlyChannelSenderV3Counter.incr()

Future.value(isNtabOnlyChannel)

}

private def isPushOnly(channels: Seq[ChannelName], candidate: PushCandidate): Future[Boolean] = {

Future.value(channels.contains(ChannelName.PushOnly))

}

def notify(

channels: Seq[ChannelName],

candidate: PushCandidate

): Future[IbisResponse] = {

Future

.join(isPushOnly(channels, candidate), isNtabOnlySend(channels)).map {

case (isPushOnly, isNtabOnly) =>

if (isPushOnly) {

track(ibis2SendLatency)(ibis2Sender.send(channels, candidate, notificationScribe, None))

} else {

track(ntabSendLatency)(

ntabSender

.send(candidate, isNtabOnly))

.flatMap { ntabResponse =>

if (isNtabOnly) {

ntabWriteThenSkipPushCounter.incr()

candidate

.scribeData(channels = channels).foreach(notificationScribe).map(\_ =>

IbisResponse(Sent))

} else {

ntabWriteThenIbisSendCounter.incr()

track(ibis2SendLatency)(

ibis2Sender.send(channels, candidate, notificationScribe, ntabResponse))

}

}

}

}.flatten

}

def loggedOutNotify(

candidate: PushCandidate

): Future[IbisResponse] = {

val ibisResponse = {

track(loggedOutIbis2SendLatency)(

ibis2Sender.send(Seq(ChannelName.PushNtab), candidate, notificationScribe, None))

}

ibisResponse

}

}