package com.twitter.cr\_mixer.logging

import com.twitter.cr\_mixer.logging.ScribeLoggerUtils.\_

import com.twitter.cr\_mixer.model.UtegTweetCandidateGeneratorQuery

import com.twitter.cr\_mixer.model.ModuleNames

import com.twitter.cr\_mixer.model.TweetWithScoreAndSocialProof

import com.twitter.cr\_mixer.param.decider.CrMixerDecider

import com.twitter.cr\_mixer.param.decider.DeciderConstants

import com.twitter.cr\_mixer.thriftscala.UtegTweetRequest

import com.twitter.cr\_mixer.thriftscala.UtegTweetResponse

import com.twitter.cr\_mixer.thriftscala.FetchCandidatesResult

import com.twitter.cr\_mixer.thriftscala.GetUtegTweetsScribe

import com.twitter.cr\_mixer.thriftscala.PerformanceMetrics

import com.twitter.cr\_mixer.thriftscala.UtegTweetResult

import com.twitter.cr\_mixer.thriftscala.UtegTweetTopLevelApiResult

import com.twitter.cr\_mixer.thriftscala.TweetCandidateWithMetadata

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.finagle.tracing.Trace

import com.twitter.logging.Logger

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

import com.twitter.util.Future

import com.twitter.util.Stopwatch

import javax.inject.Inject

import javax.inject.Named

import javax.inject.Singleton

@Singleton

case class UtegTweetScribeLogger @Inject() (

decider: CrMixerDecider,

statsReceiver: StatsReceiver,

@Named(ModuleNames.UtegTweetsLogger) utegTweetScribeLogger: Logger) {

private val scopedStats = statsReceiver.scope("UtegTweetScribeLogger")

private val topLevelApiStats = scopedStats.scope("TopLevelApi")

private val upperFunnelsStats = scopedStats.scope("UpperFunnels")

def scribeInitialCandidates(

query: UtegTweetCandidateGeneratorQuery,

getResultFn: => Future[Seq[TweetWithScoreAndSocialProof]]

): Future[Seq[TweetWithScoreAndSocialProof]] = {

scribeResultsAndPerformanceMetrics(

ScribeMetadata.from(query),

getResultFn,

convertToResultFn = convertFetchCandidatesResult

)

}

/\*\*

\* Scribe Top Level API Request / Response and performance metrics

\* for the GetUtegTweetRecommendations() endpoint.

\*/

def scribeGetUtegTweetRecommendations(

request: UtegTweetRequest,

startTime: Long,

scribeMetadata: ScribeMetadata,

getResultFn: => Future[UtegTweetResponse]

): Future[UtegTweetResponse] = {

val timer = Stopwatch.start()

getResultFn.onSuccess { response =>

if (decider.isAvailableForId(

scribeMetadata.userId,

DeciderConstants.upperFunnelPerStepScribeRate)) {

topLevelApiStats.counter(scribeMetadata.product.originalName).incr()

val latencyMs = timer().inMilliseconds

val result = convertTopLevelAPIResult(request, response, startTime)

val traceId = Trace.id.traceId.toLong

val scribeMsg =

buildScribeMessage(result, scribeMetadata, latencyMs, traceId)

scribeResult(scribeMsg)

}

}

}

private def convertTopLevelAPIResult(

request: UtegTweetRequest,

response: UtegTweetResponse,

startTime: Long

): UtegTweetResult = {

UtegTweetResult.UtegTweetTopLevelApiResult(

UtegTweetTopLevelApiResult(

timestamp = startTime,

request = request,

response = response

))

}

private def buildScribeMessage(

utegTweetResult: UtegTweetResult,

scribeMetadata: ScribeMetadata,

latencyMs: Long,

traceId: Long

): GetUtegTweetsScribe = {

GetUtegTweetsScribe(

uuid = scribeMetadata.requestUUID,

userId = scribeMetadata.userId,

utegTweetResult = utegTweetResult,

traceId = Some(traceId),

performanceMetrics = Some(PerformanceMetrics(Some(latencyMs))),

impressedBuckets = getImpressedBuckets(scopedStats)

)

}

private def scribeResult(

scribeMsg: GetUtegTweetsScribe

): Unit = {

publish(logger = utegTweetScribeLogger, codec = GetUtegTweetsScribe, message = scribeMsg)

}

private def convertFetchCandidatesResult(

candidates: Seq[TweetWithScoreAndSocialProof],

requestUserId: UserId

): UtegTweetResult = {

val tweetCandidatesWithMetadata = candidates.map { candidate =>

TweetCandidateWithMetadata(

tweetId = candidate.tweetId,

candidateGenerationKey = None

) // do not hydrate candidateGenerationKey to save cost

}

UtegTweetResult.FetchCandidatesResult(FetchCandidatesResult(Some(tweetCandidatesWithMetadata)))

}

/\*\*

\* Scribe Per-step intermediate results and performance metrics

\* for each step: fetch candidates, filters.

\*/

private def scribeResultsAndPerformanceMetrics[T](

scribeMetadata: ScribeMetadata,

getResultFn: => Future[T],

convertToResultFn: (T, UserId) => UtegTweetResult

): Future[T] = {

val timer = Stopwatch.start()

getResultFn.onSuccess { input =>

if (decider.isAvailableForId(

scribeMetadata.userId,

DeciderConstants.upperFunnelPerStepScribeRate)) {

upperFunnelsStats.counter(scribeMetadata.product.originalName).incr()

val latencyMs = timer().inMilliseconds

val result = convertToResultFn(input, scribeMetadata.userId)

val traceId = Trace.id.traceId.toLong

val scribeMsg =

buildScribeMessage(result, scribeMetadata, latencyMs, traceId)

scribeResult(scribeMsg)

}

}

}

}