package com.twitter.cr\_mixer.logging

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

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

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

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

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

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

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

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

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

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

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

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

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

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

import com.twitter.cr\_mixer.util.CandidateGenerationKeyUtil

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 AdsRecommendationsScribeLogger @Inject() (

@Named(ModuleNames.AdsRecommendationsLogger) adsRecommendationsScribeLogger: Logger,

decider: CrMixerDecider,

statsReceiver: StatsReceiver) {

private val scopedStats = statsReceiver.scope(this.getClass.getCanonicalName)

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

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

/\*

\* Scribe first step results after fetching initial ads candidate

\* \*/

def scribeInitialAdsCandidates(

query: AdsCandidateGeneratorQuery,

getResultFn: => Future[Seq[Seq[InitialAdsCandidate]]],

enableScribe: Boolean // controlled by feature switch so that we can scribe for certain DDG

): Future[Seq[Seq[InitialAdsCandidate]]] = {

val scribeMetadata = ScribeMetadata.from(query)

val timer = Stopwatch.start()

getResultFn.onSuccess { input =>

val latencyMs = timer().inMilliseconds

val result = convertFetchCandidatesResult(input, scribeMetadata.userId)

val traceId = Trace.id.traceId.toLong

val scribeMsg = buildScribeMessage(result, scribeMetadata, latencyMs, traceId)

if (enableScribe && decider.isAvailableForId(

scribeMetadata.userId,

DeciderConstants.adsRecommendationsPerExperimentScribeRate)) {

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

scribeResult(scribeMsg)

}

}

}

/\*

\* Scribe top level API results

\* \*/

def scribeGetAdsRecommendations(

request: AdsRequest,

startTime: Long,

scribeMetadata: ScribeMetadata,

getResultFn: => Future[AdsResponse],

enableScribe: Boolean

): Future[AdsResponse] = {

val timer = Stopwatch.start()

getResultFn.onSuccess { response =>

val latencyMs = timer().inMilliseconds

val result = AdsRecommendationsResult.AdsRecommendationTopLevelApiResult(

AdsRecommendationTopLevelApiResult(

timestamp = startTime,

request = request,

response = response

))

val traceId = Trace.id.traceId.toLong

val scribeMsg = buildScribeMessage(result, scribeMetadata, latencyMs, traceId)

if (enableScribe && decider.isAvailableForId(

scribeMetadata.userId,

DeciderConstants.adsRecommendationsPerExperimentScribeRate)) {

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

scribeResult(scribeMsg)

}

}

}

private def convertFetchCandidatesResult(

candidatesSeq: Seq[Seq[InitialAdsCandidate]],

requestUserId: UserId

): AdsRecommendationsResult = {

val tweetCandidatesWithMetadata = candidatesSeq.flatMap { candidates =>

candidates.map { candidate =>

TweetCandidateWithMetadata(

tweetId = candidate.tweetId,

candidateGenerationKey = Some(

CandidateGenerationKeyUtil.toThrift(candidate.candidateGenerationInfo, requestUserId)),

score = Some(candidate.getSimilarityScore),

numCandidateGenerationKeys = None // not populated yet

)

}

}

AdsRecommendationsResult.FetchCandidatesResult(

FetchCandidatesResult(Some(tweetCandidatesWithMetadata)))

}

private def buildScribeMessage(

result: AdsRecommendationsResult,

scribeMetadata: ScribeMetadata,

latencyMs: Long,

traceId: Long

): GetAdsRecommendationsScribe = {

GetAdsRecommendationsScribe(

uuid = scribeMetadata.requestUUID,

userId = scribeMetadata.userId,

result = result,

traceId = Some(traceId),

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

impressedBuckets = getImpressedBuckets(scopedStats)

)

}

private def scribeResult(

scribeMsg: GetAdsRecommendationsScribe

): Unit = {

publish(

logger = adsRecommendationsScribeLogger,

codec = GetAdsRecommendationsScribe,

message = scribeMsg)

}

}