package com.twitter.timelineranker.config

import com.twitter.conversions.DurationOps.\_

import com.twitter.conversions.PercentOps.\_

import com.twitter.cortex\_tweet\_annotate.thriftscala.CortexTweetQueryService

import com.twitter.finagle.ssl.OpportunisticTls

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.finagle.thrift.ClientId

import com.twitter.finagle.util.DefaultTimer

import com.twitter.gizmoduck.thriftscala.{UserService => Gizmoduck}

import com.twitter.manhattan.v1.thriftscala.{ManhattanCoordinator => ManhattanV1}

import com.twitter.merlin.thriftscala.UserRolesService

import com.twitter.recos.user\_tweet\_entity\_graph.thriftscala.UserTweetEntityGraph

import com.twitter.socialgraph.thriftscala.SocialGraphService

import com.twitter.storehaus.Store

import com.twitter.strato.client.Strato

import com.twitter.strato.client.{Client => StratoClient}

import com.twitter.timelineranker.clients.content\_features\_cache.ContentFeaturesMemcacheBuilder

import com.twitter.timelineranker.recap.model.ContentFeatures

import com.twitter.timelineranker.thriftscala.TimelineRanker

import com.twitter.timelines.clients.memcache\_common.StorehausMemcacheConfig

import com.twitter.timelines.model.TweetId

import com.twitter.timelineservice.thriftscala.TimelineService

import com.twitter.tweetypie.thriftscala.{TweetService => TweetyPie}

import com.twitter.util.Timer

import org.apache.thrift.protocol.TCompactProtocol

class DefaultUnderlyingClientConfiguration(flags: TimelineRankerFlags, statsReceiver: StatsReceiver)

extends UnderlyingClientConfiguration(flags, statsReceiver) { top =>

val timer: Timer = DefaultTimer

val twCachePrefix = "/srv#/prod/local/cache"

override val cortexTweetQueryServiceClient: CortexTweetQueryService.MethodPerEndpoint = {

methodPerEndpointClient[

CortexTweetQueryService.ServicePerEndpoint,

CortexTweetQueryService.MethodPerEndpoint](

thriftMuxClientBuilder("cortex-tweet-query", requireMutualTls = true)

.dest("/s/cortex-tweet-annotate/cortex-tweet-query")

.requestTimeout(200.milliseconds)

.timeout(500.milliseconds)

)

}

override val gizmoduckClient: Gizmoduck.MethodPerEndpoint = {

methodPerEndpointClient[Gizmoduck.ServicePerEndpoint, Gizmoduck.MethodPerEndpoint](

thriftMuxClientBuilder("gizmoduck", requireMutualTls = true)

.dest("/s/gizmoduck/gizmoduck")

.requestTimeout(400.milliseconds)

.timeout(900.milliseconds)

)

}

override lazy val manhattanStarbuckClient: ManhattanV1.MethodPerEndpoint = {

methodPerEndpointClient[ManhattanV1.ServicePerEndpoint, ManhattanV1.MethodPerEndpoint](

thriftMuxClientBuilder("manhattan.starbuck", requireMutualTls = true)

.dest("/s/manhattan/starbuck.native-thrift")

.requestTimeout(600.millis)

)

}

override val sgsClient: SocialGraphService.MethodPerEndpoint = {

methodPerEndpointClient[

SocialGraphService.ServicePerEndpoint,

SocialGraphService.MethodPerEndpoint](

thriftMuxClientBuilder("socialgraph", requireMutualTls = true)

.dest("/s/socialgraph/socialgraph")

.requestTimeout(350.milliseconds)

.timeout(700.milliseconds)

)

}

override lazy val timelineRankerForwardingClient: TimelineRanker.FinagledClient =

new TimelineRanker.FinagledClient(

thriftMuxClientBuilder(

TimelineRankerConstants.ForwardedClientName,

ClientId(TimelineRankerConstants.ForwardedClientName),

protocolFactoryOption = Some(new TCompactProtocol.Factory()),

requireMutualTls = true

).dest("/s/timelineranker/timelineranker:compactthrift").build(),

protocolFactory = new TCompactProtocol.Factory()

)

override val timelineServiceClient: TimelineService.MethodPerEndpoint = {

methodPerEndpointClient[TimelineService.ServicePerEndpoint, TimelineService.MethodPerEndpoint](

thriftMuxClientBuilder("timelineservice", requireMutualTls = true)

.dest("/s/timelineservice/timelineservice")

.requestTimeout(600.milliseconds)

.timeout(800.milliseconds)

)

}

override val tweetyPieHighQoSClient: TweetyPie.MethodPerEndpoint = {

methodPerEndpointClient[TweetyPie.ServicePerEndpoint, TweetyPie.MethodPerEndpoint](

thriftMuxClientBuilder("tweetypieHighQoS", requireMutualTls = true)

.dest("/s/tweetypie/tweetypie")

.requestTimeout(450.milliseconds)

.timeout(800.milliseconds),

maxExtraLoadPercent = Some(1.percent)

)

}

/\*\*

\* Provide less costly TweetPie client with the trade-off of reduced quality. Intended for use cases

\* which are not essential for successful completion of timeline requests. Currently this client differs

\* from the highQoS endpoint by having retries count set to 1 instead of 2.

\*/

override val tweetyPieLowQoSClient: TweetyPie.MethodPerEndpoint = {

methodPerEndpointClient[TweetyPie.ServicePerEndpoint, TweetyPie.MethodPerEndpoint](

thriftMuxClientBuilder("tweetypieLowQoS", requireMutualTls = true)

.dest("/s/tweetypie/tweetypie")

.retryPolicy(mkRetryPolicy(1)) // override default value

.requestTimeout(450.milliseconds)

.timeout(800.milliseconds),

maxExtraLoadPercent = Some(1.percent)

)

}

override val userRolesServiceClient: UserRolesService.MethodPerEndpoint = {

methodPerEndpointClient[

UserRolesService.ServicePerEndpoint,

UserRolesService.MethodPerEndpoint](

thriftMuxClientBuilder("merlin", requireMutualTls = true)

.dest("/s/merlin/merlin")

.requestTimeout(1.second)

)

}

lazy val contentFeaturesCache: Store[TweetId, ContentFeatures] =

new ContentFeaturesMemcacheBuilder(

config = new StorehausMemcacheConfig(

destName = s"$twCachePrefix/timelines\_content\_features:twemcaches",

keyPrefix = "",

requestTimeout = 50.milliseconds,

numTries = 1,

globalTimeout = 60.milliseconds,

tcpConnectTimeout = 50.milliseconds,

connectionAcquisitionTimeout = 25.milliseconds,

numPendingRequests = 100,

isReadOnly = false,

serviceIdentifier = serviceIdentifier

),

ttl = 48.hours,

statsReceiver

).build

override val userTweetEntityGraphClient: UserTweetEntityGraph.FinagledClient =

new UserTweetEntityGraph.FinagledClient(

thriftMuxClientBuilder("user\_tweet\_entity\_graph", requireMutualTls = true)

.dest("/s/cassowary/user\_tweet\_entity\_graph")

.retryPolicy(mkRetryPolicy(2))

.requestTimeout(300.milliseconds)

.build()

)

override val stratoClient: StratoClient =

Strato.client.withMutualTls(serviceIdentifier, OpportunisticTls.Required).build()

}