package com.twitter.tweetypie.backends

import com.twitter.container.{thriftscala => ccs}

import com.twitter.finagle.Backoff

import com.twitter.finagle.service.RetryPolicy

import com.twitter.finatra.thrift.thriftscala.ServerError

import com.twitter.finatra.thrift.thriftscala.ServerErrorCause

import com.twitter.servo.util.FutureArrow

import com.twitter.tweetypie.Duration

import com.twitter.tweetypie.Future

import com.twitter.tweetypie.Try

import com.twitter.tweetypie.util.RetryPolicyBuilder

import com.twitter.tweetypie.{thriftscala => tp}

import com.twitter.util.Throw

object CreativesContainerService {

import Backend.\_

type MaterializeAsTweet = FutureArrow[ccs.MaterializeAsTweetRequests, Seq[tp.GetTweetResult]]

type MaterializeAsTweetFields =

FutureArrow[ccs.MaterializeAsTweetFieldsRequests, Seq[tp.GetTweetFieldsResult]]

def fromClient(

client: ccs.CreativesContainerService.MethodPerEndpoint

): CreativesContainerService =

new CreativesContainerService {

val materializeAsTweet: MaterializeAsTweet = FutureArrow(client.materializeAsTweets)

val materializeAsTweetFields: MaterializeAsTweetFields = FutureArrow(

client.materializeAsTweetFields)

def ping(): Future[Unit] = client.materializeAsTweets(ccs.MaterializeAsTweetRequests()).unit

}

case class Config(

requestTimeout: Duration,

timeoutBackoffs: Stream[Duration],

serverErrorBackoffs: Stream[Duration]) {

def apply(svc: CreativesContainerService, ctx: Backend.Context): CreativesContainerService =

new CreativesContainerService {

override val materializeAsTweet: MaterializeAsTweet =

policy("materializeAsTweets", ctx)(svc.materializeAsTweet)

override val materializeAsTweetFields: MaterializeAsTweetFields =

policy("materializeAsTweetFields", ctx)(svc.materializeAsTweetFields)

override def ping(): Future[Unit] = svc.ping()

}

private[this] def policy[A, B](name: String, ctx: Context): Builder[A, B] =

defaultPolicy(name, requestTimeout, retryPolicy, ctx)

private[this] def retryPolicy[B]: RetryPolicy[Try[B]] =

RetryPolicy.combine[Try[B]](

RetryPolicyBuilder.timeouts[B](timeoutBackoffs),

RetryPolicy.backoff(Backoff.fromStream(serverErrorBackoffs)) {

case Throw(ex: ServerError) if ex.errorCause != ServerErrorCause.NotImplemented => true

}

)

implicit val warmup: Warmup[CreativesContainerService] =

Warmup[CreativesContainerService]("creativesContainerService")(\_.ping())

}

}

trait CreativesContainerService {

import CreativesContainerService.\_

val materializeAsTweet: MaterializeAsTweet

val materializeAsTweetFields: MaterializeAsTweetFields

def ping(): Future[Unit]

}