package com.twitter.recosinjector.config

import com.twitter.bijection.scrooge.BinaryScalaCodec

import com.twitter.conversions.DurationOps.\_

import com.twitter.finagle.client.ClientRegistry

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.store.TweetCreationTimeMHStore

import com.twitter.frigate.common.util.Finagle.\_

import com.twitter.frigate.common.util.{UrlInfo, UrlInfoInjection, UrlResolver}

import com.twitter.gizmoduck.thriftscala.{LookupContext, QueryFields, User, UserService}

import com.twitter.hermit.store.common.{ObservedCachedReadableStore, ObservedMemcachedReadableStore}

import com.twitter.hermit.store.gizmoduck.GizmoduckUserStore

import com.twitter.hermit.store.tweetypie.TweetyPieStore

import com.twitter.logging.Logger

import com.twitter.pink\_floyd.thriftscala.{ClientIdentifier, Storer}

import com.twitter.socialgraph.thriftscala.{IdsRequest, SocialGraphService}

import com.twitter.spam.rtf.thriftscala.SafetyLevel

import com.twitter.stitch.socialgraph.SocialGraph

import com.twitter.stitch.storehaus.ReadableStoreOfStitch

import com.twitter.stitch.tweetypie.TweetyPie.TweetyPieResult

import com.twitter.storage.client.manhattan.kv.{

ManhattanKVClient,

ManhattanKVClientMtlsParams,

ManhattanKVEndpointBuilder

}

import com.twitter.storehaus.ReadableStore

import com.twitter.tweetypie.thriftscala.{GetTweetOptions, TweetService}

import com.twitter.util.Future

/\*

\* Any finagle clients should not be defined as lazy. If defined lazy,

\* ClientRegistry.expAllRegisteredClientsResolved() call in init will not ensure that the clients

\* are active before thrift endpoint is active. We want the clients to be active, because zookeeper

\* resolution triggered by first request(s) might result in the request(s) failing.

\*/

trait DeployConfig extends Config with CacheConfig {

implicit def statsReceiver: StatsReceiver

def log: Logger

// Clients

val gizmoduckClient = new UserService.FinagledClient(

readOnlyThriftService(

"gizmoduck",

"/s/gizmoduck/gizmoduck",

statsReceiver,

recosInjectorThriftClientId,

requestTimeout = 450.milliseconds,

mTLSServiceIdentifier = Some(serviceIdentifier)

)

)

val tweetyPieClient = new TweetService.FinagledClient(

readOnlyThriftService(

"tweetypie",

"/s/tweetypie/tweetypie",

statsReceiver,

recosInjectorThriftClientId,

requestTimeout = 450.milliseconds,

mTLSServiceIdentifier = Some(serviceIdentifier)

)

)

val sgsClient = new SocialGraphService.FinagledClient(

readOnlyThriftService(

"socialgraph",

"/s/socialgraph/socialgraph",

statsReceiver,

recosInjectorThriftClientId,

requestTimeout = 450.milliseconds,

mTLSServiceIdentifier = Some(serviceIdentifier)

)

)

val pinkStoreClient = new Storer.FinagledClient(

readOnlyThriftService(

"pink\_store",

"/s/spiderduck/pink-store",

statsReceiver,

recosInjectorThriftClientId,

requestTimeout = 450.milliseconds,

mTLSServiceIdentifier = Some(serviceIdentifier)

)

)

// Stores

private val \_gizmoduckStore = {

val queryFields: Set[QueryFields] = Set(

QueryFields.Discoverability,

QueryFields.Labels,

QueryFields.Safety

)

val context: LookupContext = LookupContext(

includeDeactivated = true,

safetyLevel = Some(SafetyLevel.Recommendations)

)

GizmoduckUserStore(

client = gizmoduckClient,

queryFields = queryFields,

context = context,

statsReceiver = statsReceiver

)

}

override val userStore: ReadableStore[Long, User] = {

// memcache based cache

ObservedMemcachedReadableStore.fromCacheClient(

backingStore = \_gizmoduckStore,

cacheClient = recosInjectorCoreSvcsCacheClient,

ttl = 2.hours

)(

valueInjection = BinaryScalaCodec(User),

statsReceiver = statsReceiver.scope("UserStore"),

keyToString = { k: Long =>

s"usri/$k"

}

)

}

/\*\*

\* TweetyPie store, used to fetch tweet objects when unavailable, and also as a source of

\* tweet SafetyLevel filtering.

\* Note: we do NOT cache TweetyPie calls, as it makes tweet SafetyLevel filtering less accurate.

\* TweetyPie QPS is < 20K/cluster.

\* More info is here:

\* https://cgit.twitter.biz/source/tree/src/thrift/com/twitter/spam/rtf/safety\_level.thrift

\*/

override val tweetyPieStore: ReadableStore[Long, TweetyPieResult] = {

val getTweetOptions = Some(

GetTweetOptions(

includeCards = true,

safetyLevel = Some(SafetyLevel.RecosWritePath)

)

)

TweetyPieStore(

tweetyPieClient,

getTweetOptions,

convertExceptionsToNotFound = false // Do not suppress TweetyPie errors. Leave it to caller

)

}

private val \_urlInfoStore = {

//Initialize pink store client, for parsing url

UrlResolver(

pinkStoreClient,

statsReceiver.scope("urlFetcher"),

clientId = ClientIdentifier.Recoshose

)

}

override val urlInfoStore: ReadableStore[String, UrlInfo] = {

// memcache based cache

val memcachedStore = ObservedMemcachedReadableStore.fromCacheClient(

backingStore = \_urlInfoStore,

cacheClient = recosInjectorCoreSvcsCacheClient,

ttl = 2.hours

)(

valueInjection = UrlInfoInjection,

statsReceiver = statsReceiver.scope("UrlInfoStore"),

keyToString = { k: String =>

s"uisri/$k"

}

)

ObservedCachedReadableStore.from(

memcachedStore,

ttl = 1.minutes,

maxKeys = 1e5.toInt,

windowSize = 10000L,

cacheName = "url\_store\_in\_proc\_cache"

)(statsReceiver.scope("url\_store\_in\_proc\_cache"))

}

override val socialGraphIdStore = ReadableStoreOfStitch { idsRequest: IdsRequest =>

SocialGraph(sgsClient).ids(idsRequest)

}

/\*\*

\* MH Store for updating the last time user created a tweet

\*/

val tweetCreationStore: TweetCreationTimeMHStore = {

val client = ManhattanKVClient(

appId = "recos\_tweet\_creation\_info",

dest = "/s/manhattan/omega.native-thrift",

mtlsParams = ManhattanKVClientMtlsParams(serviceIdentifier)

)

val endpoint = ManhattanKVEndpointBuilder(client)

.defaultMaxTimeout(700.milliseconds)

.statsReceiver(

statsReceiver

.scope(serviceIdentifier.zone)

.scope(serviceIdentifier.environment)

.scope("recos\_injector\_tweet\_creation\_info\_store")

)

.build()

val dataset = if (serviceIdentifier.environment == "prod") {

"recos\_injector\_tweet\_creation\_info"

} else {

"recos\_injector\_tweet\_creation\_info\_staging"

}

new TweetCreationTimeMHStore(

cluster = serviceIdentifier.zone,

endpoint = endpoint,

dataset = dataset,

writeTtl = Some(14.days),

statsReceiver.scope("recos\_injector\_tweet\_creation\_info\_store")

)

}

// wait for all serversets to populate

override def init(): Future[Unit] = ClientRegistry.expAllRegisteredClientsResolved().unit

}