package com.twitter.frigate.pushservice.config

import com.twitter.abdecider.LoggingABDecider

import com.twitter.bijection.scrooge.BinaryScalaCodec

import com.twitter.bijection.Base64String

import com.twitter.bijection.Injection

import com.twitter.conversions.DurationOps.\_

import com.twitter.decider.Decider

import com.twitter.featureswitches.v2.FeatureSwitches

import com.twitter.finagle.mtls.authentication.ServiceIdentifier

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.finagle.thrift.ClientId

import com.twitter.finagle.thrift.RichClientParam

import com.twitter.finagle.util.DefaultTimer

import com.twitter.frigate.common.config.RateLimiterGenerator

import com.twitter.frigate.common.filter.DynamicRequestMeterFilter

import com.twitter.frigate.common.history.ManhattanHistoryStore

import com.twitter.frigate.common.history.InvalidatingAfterWritesPushServiceHistoryStore

import com.twitter.frigate.common.history.ManhattanKVHistoryStore

import com.twitter.frigate.common.history.PushServiceHistoryStore

import com.twitter.frigate.common.history.SimplePushServiceHistoryStore

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

import com.twitter.frigate.data\_pipeline.features\_common.FeatureStoreUtil

import com.twitter.frigate.data\_pipeline.features\_common.TargetLevelFeaturesConfig

import com.twitter.frigate.pushservice.model.PushTypes.Target

import com.twitter.frigate.pushservice.params.DeciderKey

import com.twitter.frigate.pushservice.params.PushQPSLimitConstants

import com.twitter.frigate.pushservice.params.PushServiceTunableKeys

import com.twitter.frigate.pushservice.params.ShardParams

import com.twitter.frigate.pushservice.store.PushIbis2Store

import com.twitter.frigate.pushservice.thriftscala.PushRequestScribe

import com.twitter.frigate.scribe.thriftscala.NotificationScribe

import com.twitter.ibis2.service.thriftscala.Ibis2Service

import com.twitter.logging.Logger

import com.twitter.notificationservice.api.thriftscala.DeleteCurrentTimelineForUserRequest

import com.twitter.notificationservice.api.thriftscala.NotificationApi

import com.twitter.notificationservice.api.thriftscala.NotificationApi$FinagleClient

import com.twitter.notificationservice.thriftscala.CreateGenericNotificationRequest

import com.twitter.notificationservice.thriftscala.CreateGenericNotificationResponse

import com.twitter.notificationservice.thriftscala.DeleteGenericNotificationRequest

import com.twitter.notificationservice.thriftscala.NotificationService

import com.twitter.notificationservice.thriftscala.NotificationService$FinagleClient

import com.twitter.servo.decider.DeciderGateBuilder

import com.twitter.util.tunable.TunableMap

import com.twitter.util.Future

import com.twitter.util.Timer

case class ProdConfig(

override val isServiceLocal: Boolean,

override val localConfigRepoPath: String,

override val inMemCacheOff: Boolean,

override val decider: Decider,

override val abDecider: LoggingABDecider,

override val featureSwitches: FeatureSwitches,

override val shardParams: ShardParams,

override val serviceIdentifier: ServiceIdentifier,

override val tunableMap: TunableMap,

)(

implicit val statsReceiver: StatsReceiver)

extends {

// Due to trait initialization logic in Scala, any abstract members declared in Config or

// DeployConfig should be declared in this block. Otherwise the abstract member might initialize to

// null if invoked before object creation finishing.

val log = Logger("ProdConfig")

// Deciders

val isPushserviceCanaryDeepbirdv2CanaryClusterEnabled = decider

.feature(DeciderKey.enablePushserviceDeepbirdv2CanaryClusterDeciderKey.toString).isAvailable

// Client ids

val notifierThriftClientId = ClientId("frigate-notifier.prod")

val loggedOutNotifierThriftClientId = ClientId("frigate-logged-out-notifier.prod")

val pushserviceThriftClientId: ClientId = ClientId("frigate-pushservice.prod")

// Dests

val frigateHistoryCacheDest = "/s/cache/frigate\_history"

val memcacheCASDest = "/s/cache/magic\_recs\_cas:twemcaches"

val historyStoreMemcacheDest =

"/srv#/prod/local/cache/magic\_recs\_history:twemcaches"

val deepbirdv2PredictionServiceDest =

if (serviceIdentifier.service.equals("frigate-pushservice-canary") &&

isPushserviceCanaryDeepbirdv2CanaryClusterEnabled)

"/s/frigate/deepbirdv2-magicrecs-canary"

else "/s/frigate/deepbirdv2-magicrecs"

override val fanoutMetadataColumn = "frigate/magicfanout/prod/mh/fanoutMetadata"

override val timer: Timer = DefaultTimer

override val featureStoreUtil = FeatureStoreUtil.withParams(Some(serviceIdentifier))

override val targetLevelFeaturesConfig = TargetLevelFeaturesConfig()

val pushServiceMHCacheDest = "/s/cache/pushservice\_mh"

val pushServiceCoreSvcsCacheDest = "/srv#/prod/local/cache/pushservice\_core\_svcs"

val userTweetEntityGraphDest = "/s/cassowary/user\_tweet\_entity\_graph"

val userUserGraphDest = "/s/cassowary/user\_user\_graph"

val lexServiceDest = "/s/live-video/timeline-thrift"

val entityGraphCacheDest = "/s/cache/pushservice\_entity\_graph"

override val pushIbisV2Store = {

val service = Finagle.readOnlyThriftService(

"ibis-v2-service",

"/s/ibis2/ibis2",

statsReceiver,

notifierThriftClientId,

requestTimeout = 3.seconds,

tries = 3,

mTLSServiceIdentifier = Some(serviceIdentifier)

)

// according to ibis team, it is safe to retry on timeout, write & channel closed exceptions.

val pushIbisClient = new Ibis2Service.FinagledClient(

new DynamicRequestMeterFilter(

tunableMap(PushServiceTunableKeys.IbisQpsLimitTunableKey),

RateLimiterGenerator.asTuple(\_, shardParams.numShards, 20),

PushQPSLimitConstants.IbisOrNTabQPSForRFPH

)(timer).andThen(service),

RichClientParam(serviceName = "ibis-v2-service")

)

PushIbis2Store(pushIbisClient)

}

val notificationServiceClient: NotificationService$FinagleClient = {

val service = Finagle.readWriteThriftService(

"notificationservice",

"/s/notificationservice/notificationservice",

statsReceiver,

pushserviceThriftClientId,

requestTimeout = 10.seconds,

mTLSServiceIdentifier = Some(serviceIdentifier)

)

new NotificationService.FinagledClient(

new DynamicRequestMeterFilter(

tunableMap(PushServiceTunableKeys.NtabQpsLimitTunableKey),

RateLimiterGenerator.asTuple(\_, shardParams.numShards, 20),

PushQPSLimitConstants.IbisOrNTabQPSForRFPH)(timer).andThen(service),

RichClientParam(serviceName = "notificationservice")

)

}

val notificationServiceApiClient: NotificationApi$FinagleClient = {

val service = Finagle.readWriteThriftService(

"notificationservice-api",

"/s/notificationservice/notificationservice-api:thrift",

statsReceiver,

pushserviceThriftClientId,

requestTimeout = 10.seconds,

mTLSServiceIdentifier = Some(serviceIdentifier)

)

new NotificationApi.FinagledClient(

new DynamicRequestMeterFilter(

tunableMap(PushServiceTunableKeys.NtabQpsLimitTunableKey),

RateLimiterGenerator.asTuple(\_, shardParams.numShards, 20),

PushQPSLimitConstants.IbisOrNTabQPSForRFPH)(timer).andThen(service),

RichClientParam(serviceName = "notificationservice-api")

)

}

val mrRequestScriberNode = "mr\_request\_scribe"

val loggedOutMrRequestScriberNode = "lo\_mr\_request\_scribe"

override val pushSendEventStreamName = "frigate\_pushservice\_send\_event\_prod"

} with DeployConfig {

// Scribe

private val notificationScribeLog = Logger("notification\_scribe")

private val notificationScribeInjection: Injection[NotificationScribe, String] = BinaryScalaCodec(

NotificationScribe

) andThen Injection.connect[Array[Byte], Base64String, String]

override def notificationScribe(data: NotificationScribe): Unit = {

val logEntry: String = notificationScribeInjection(data)

notificationScribeLog.info(logEntry)

}

// History Store - Invalidates cached history after writes

override val historyStore = new InvalidatingAfterWritesPushServiceHistoryStore(

ManhattanHistoryStore(notificationHistoryStore, statsReceiver),

recentHistoryCacheClient,

new DeciderGateBuilder(decider)

.idGate(DeciderKey.enableInvalidatingCachedHistoryStoreAfterWrites)

)

override val emailHistoryStore: PushServiceHistoryStore = {

statsReceiver.scope("frigate\_email\_history").counter("request").incr()

new SimplePushServiceHistoryStore(emailNotificationHistoryStore)

}

override val loggedOutHistoryStore =

new InvalidatingAfterWritesPushServiceHistoryStore(

ManhattanKVHistoryStore(

manhattanKVLoggedOutHistoryStoreEndpoint,

"frigate\_notification\_logged\_out\_history"),

recentHistoryCacheClient,

new DeciderGateBuilder(decider)

.idGate(DeciderKey.enableInvalidatingCachedLoggedOutHistoryStoreAfterWrites)

)

private val requestScribeLog = Logger("request\_scribe")

private val requestScribeInjection: Injection[PushRequestScribe, String] = BinaryScalaCodec(

PushRequestScribe

) andThen Injection.connect[Array[Byte], Base64String, String]

override def requestScribe(data: PushRequestScribe): Unit = {

val logEntry: String = requestScribeInjection(data)

requestScribeLog.info(logEntry)

}

// generic notification server

override def notificationServiceSend(

target: Target,

request: CreateGenericNotificationRequest

): Future[CreateGenericNotificationResponse] =

notificationServiceClient.createGenericNotification(request)

// generic notification server

override def notificationServiceDelete(

request: DeleteGenericNotificationRequest

): Future[Unit] = notificationServiceClient.deleteGenericNotification(request)

// NTab-api

override def notificationServiceDeleteTimeline(

request: DeleteCurrentTimelineForUserRequest

): Future[Unit] = notificationServiceApiClient.deleteCurrentTimelineForUser(request)

}