package com.twitter.frigate.pushservice.config

import com.twitter.abdecider.LoggingABDecider

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.InvalidatingAfterWritesPushServiceHistoryStore

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

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

import com.twitter.frigate.common.history.ReadOnlyHistoryStore

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

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

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

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.\_

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.thriftscala.CreateGenericNotificationRequest

import com.twitter.notificationservice.thriftscala.CreateGenericNotificationResponse

import com.twitter.notificationservice.thriftscala.CreateGenericNotificationResponseType

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 StagingConfig(

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("StagingConfig")

// Client ids

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

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

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

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

// dest

val frigateHistoryCacheDest = "/srv#/test/local/cache/twemcache\_frigate\_history"

val memcacheCASDest = "/srv#/test/local/cache/twemcache\_magic\_recs\_cas\_dev:twemcaches"

val pushServiceMHCacheDest = "/srv#/test/local/cache/twemcache\_pushservice\_test"

val entityGraphCacheDest = "/srv#/test/local/cache/twemcache\_pushservice\_test"

val pushServiceCoreSvcsCacheDest = "/srv#/test/local/cache/twemcache\_pushservice\_core\_svcs\_test"

val historyStoreMemcacheDest = "/srv#/test/local/cache/twemcache\_eventstream\_test:twemcaches"

val userTweetEntityGraphDest = "/cluster/local/cassowary/staging/user\_tweet\_entity\_graph"

val userUserGraphDest = "/cluster/local/cassowary/staging/user\_user\_graph"

val lexServiceDest = "/srv#/staging/local/live-video/timeline-thrift"

val deepbirdv2PredictionServiceDest = "/cluster/local/frigate/staging/deepbirdv2-magicrecs"

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

override val targetLevelFeaturesConfig = TargetLevelFeaturesConfig()

val mrRequestScriberNode = "validation\_mr\_request\_scribe"

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

override val timer: Timer = DefaultTimer

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

override val pushIbisV2Store = {

val service = Finagle.readWriteThriftService(

"ibis-v2-service",

"/s/ibis2/ibis2",

statsReceiver,

notifierThriftClientId,

requestTimeout = 6.seconds,

mTLSServiceIdentifier = Some(serviceIdentifier)

)

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")

)

StagingIbis2Store(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")

)

}

} with DeployConfig {

// Scribe

private val notificationScribeLog = Logger("StagingNotificationScribe")

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

notificationScribeLog.info(data.toString)

}

private val requestScribeLog = Logger("StagingRequestScribe")

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

requestScribeLog.info(data.toString)

}

// history store

override val historyStore = new InvalidatingAfterWritesPushServiceHistoryStore(

ReadOnlyHistoryStore(

ManhattanHistoryStore(notificationHistoryStore, statsReceiver)

),

recentHistoryCacheClient,

new DeciderGateBuilder(decider)

.idGate(DeciderKey.enableInvalidatingCachedHistoryStoreAfterWrites)

)

override val emailHistoryStore: PushServiceHistoryStore = new SimplePushServiceHistoryStore(

emailNotificationHistoryStore)

// history store

override val loggedOutHistoryStore =

new InvalidatingAfterWritesPushServiceHistoryStore(

ReadOnlyHistoryStore(

ManhattanKVHistoryStore(

manhattanKVLoggedOutHistoryStoreEndpoint,

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

recentHistoryCacheClient,

new DeciderGateBuilder(decider)

.idGate(DeciderKey.enableInvalidatingCachedLoggedOutHistoryStoreAfterWrites)

)

override def notificationServiceSend(

target: Target,

request: CreateGenericNotificationRequest

): Future[CreateGenericNotificationResponse] =

target.isTeamMember.flatMap { isTeamMember =>

if (isTeamMember) {

notificationServiceClient.createGenericNotification(request)

} else {

log.info(s"Mock creating generic notification $request for user: ${target.targetId}")

Future.value(

CreateGenericNotificationResponse(CreateGenericNotificationResponseType.Success)

)

}

}

override def notificationServiceDelete(

request: DeleteGenericNotificationRequest

): Future[Unit] = Future.Unit

override def notificationServiceDeleteTimeline(

request: DeleteCurrentTimelineForUserRequest

): Future[Unit] = Future.Unit

}