package com.twitter.recosinjector

import com.twitter.app.Flag

import com.twitter.finagle.http.HttpMuxer

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

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.util.ElfOwlFilter

import com.twitter.recosinjector.clients.Gizmoduck

import com.twitter.recosinjector.clients.RecosHoseEntitiesCache

import com.twitter.recosinjector.clients.SocialGraph

import com.twitter.recosinjector.clients.Tweetypie

import com.twitter.recosinjector.clients.UrlResolver

import com.twitter.recosinjector.config.\_

import com.twitter.recosinjector.edges.SocialWriteEventToUserUserGraphBuilder

import com.twitter.recosinjector.edges.TimelineEventToUserTweetEntityGraphBuilder

import com.twitter.recosinjector.edges.TweetEventToUserTweetEntityGraphBuilder

import com.twitter.recosinjector.edges.TweetEventToUserUserGraphBuilder

import com.twitter.recosinjector.edges.UnifiedUserActionToUserVideoGraphBuilder

import com.twitter.recosinjector.edges.UnifiedUserActionToUserAdGraphBuilder

import com.twitter.recosinjector.edges.UnifiedUserActionToUserTweetGraphPlusBuilder

import com.twitter.recosinjector.edges.UserTweetEntityEdgeBuilder

import com.twitter.recosinjector.event\_processors.SocialWriteEventProcessor

import com.twitter.recosinjector.event\_processors.TimelineEventProcessor

import com.twitter.recosinjector.event\_processors.TweetEventProcessor

import com.twitter.recosinjector.publishers.KafkaEventPublisher

import com.twitter.recosinjector.uua\_processors.UnifiedUserActionProcessor

import com.twitter.recosinjector.uua\_processors.UnifiedUserActionsConsumer

import com.twitter.server.logging.{Logging => JDK14Logging}

import com.twitter.server.Deciderable

import com.twitter.server.TwitterServer

import com.twitter.socialgraph.thriftscala.WriteEvent

import com.twitter.timelineservice.thriftscala.{Event => TimelineEvent}

import com.twitter.tweetypie.thriftscala.TweetEvent

import com.twitter.util.Await

import com.twitter.util.Duration

import java.util.concurrent.TimeUnit

object Main extends TwitterServer with JDK14Logging with Deciderable { self =>

implicit val stats: StatsReceiver = statsReceiver

private val dataCenter: Flag[String] = flag("service.cluster", "atla", "Data Center")

private val serviceRole: Flag[String] = flag("service.role", "Service Role")

private val serviceEnv: Flag[String] = flag("service.env", "Service Env")

private val serviceName: Flag[String] = flag("service.name", "Service Name")

private val shardId = flag("shardId", 0, "Shard ID")

private val numShards = flag("numShards", 1, "Number of shards for this service")

private val truststoreLocation =

flag[String]("truststore\_location", "", "Truststore file location")

def main(): Unit = {

val serviceIdentifier = ServiceIdentifier(

role = serviceRole(),

service = serviceName(),

environment = serviceEnv(),

zone = dataCenter()

)

println("ServiceIdentifier = " + serviceIdentifier.toString)

log.info("ServiceIdentifier = " + serviceIdentifier.toString)

val shard = shardId()

val numOfShards = numShards()

val environment = serviceEnv()

implicit val config: DeployConfig = {

environment match {

case "prod" => ProdConfig(serviceIdentifier)(stats)

case "staging" | "devel" => StagingConfig(serviceIdentifier)

case env => throw new Exception(s"Unknown environment $env")

}

}

// Initialize the config and wait for initialization to finish

Await.ready(config.init())

log.info(

"Starting Recos Injector: environment %s, clientId %s",

environment,

config.recosInjectorThriftClientId

)

log.info("Starting shard Id: %d of %d shards...".format(shard, numOfShards))

// Client wrappers

val cache = new RecosHoseEntitiesCache(config.recosInjectorCoreSvcsCacheClient)

val gizmoduck = new Gizmoduck(config.userStore)

val socialGraph = new SocialGraph(config.socialGraphIdStore)

val tweetypie = new Tweetypie(config.tweetyPieStore)

val urlResolver = new UrlResolver(config.urlInfoStore)

// Edge builders

val userTweetEntityEdgeBuilder = new UserTweetEntityEdgeBuilder(cache, urlResolver)

// Publishers

val kafkaEventPublisher = KafkaEventPublisher(

"/s/kafka/recommendations:kafka-tls",

config.outputKafkaTopicPrefix,

config.recosInjectorThriftClientId,

truststoreLocation())

// Message Builders

val socialWriteToUserUserMessageBuilder =

new SocialWriteEventToUserUserGraphBuilder()(

statsReceiver.scope("SocialWriteEventToUserUserGraphBuilder")

)

val timelineToUserTweetEntityMessageBuilder = new TimelineEventToUserTweetEntityGraphBuilder(

userTweetEntityEdgeBuilder = userTweetEntityEdgeBuilder

)(statsReceiver.scope("TimelineEventToUserTweetEntityGraphBuilder"))

val tweetEventToUserTweetEntityGraphBuilder = new TweetEventToUserTweetEntityGraphBuilder(

userTweetEntityEdgeBuilder = userTweetEntityEdgeBuilder,

tweetCreationStore = config.tweetCreationStore,

decider = config.recosInjectorDecider

)(statsReceiver.scope("TweetEventToUserTweetEntityGraphBuilder"))

val socialWriteEventProcessor = new SocialWriteEventProcessor(

eventBusStreamName = s"recos\_injector\_social\_write\_event\_$environment",

thriftStruct = WriteEvent,

serviceIdentifier = serviceIdentifier,

kafkaEventPublisher = kafkaEventPublisher,

userUserGraphTopic = KafkaEventPublisher.UserUserTopic,

userUserGraphMessageBuilder = socialWriteToUserUserMessageBuilder

)(statsReceiver.scope("SocialWriteEventProcessor"))

val tweetToUserUserMessageBuilder = new TweetEventToUserUserGraphBuilder()(

statsReceiver.scope("TweetEventToUserUserGraphBuilder")

)

val unifiedUserActionToUserVideoGraphBuilder = new UnifiedUserActionToUserVideoGraphBuilder(

userTweetEntityEdgeBuilder = userTweetEntityEdgeBuilder

)(statsReceiver.scope("UnifiedUserActionToUserVideoGraphBuilder"))

val unifiedUserActionToUserAdGraphBuilder = new UnifiedUserActionToUserAdGraphBuilder(

userTweetEntityEdgeBuilder = userTweetEntityEdgeBuilder

)(statsReceiver.scope("UnifiedUserActionToUserAdGraphBuilder"))

val unifiedUserActionToUserTweetGraphPlusBuilder =

new UnifiedUserActionToUserTweetGraphPlusBuilder(

userTweetEntityEdgeBuilder = userTweetEntityEdgeBuilder

)(statsReceiver.scope("UnifiedUserActionToUserTweetGraphPlusBuilder"))

// Processors

val tweetEventProcessor = new TweetEventProcessor(

eventBusStreamName = s"recos\_injector\_tweet\_events\_$environment",

thriftStruct = TweetEvent,

serviceIdentifier = serviceIdentifier,

userUserGraphMessageBuilder = tweetToUserUserMessageBuilder,

userUserGraphTopic = KafkaEventPublisher.UserUserTopic,

userTweetEntityGraphMessageBuilder = tweetEventToUserTweetEntityGraphBuilder,

userTweetEntityGraphTopic = KafkaEventPublisher.UserTweetEntityTopic,

kafkaEventPublisher = kafkaEventPublisher,

socialGraph = socialGraph,

tweetypie = tweetypie,

gizmoduck = gizmoduck

)(statsReceiver.scope("TweetEventProcessor"))

val timelineEventProcessor = new TimelineEventProcessor(

eventBusStreamName = s"recos\_injector\_timeline\_events\_prototype\_$environment",

thriftStruct = TimelineEvent,

serviceIdentifier = serviceIdentifier,

kafkaEventPublisher = kafkaEventPublisher,

userTweetEntityGraphTopic = KafkaEventPublisher.UserTweetEntityTopic,

userTweetEntityGraphMessageBuilder = timelineToUserTweetEntityMessageBuilder,

decider = config.recosInjectorDecider,

gizmoduck = gizmoduck,

tweetypie = tweetypie

)(statsReceiver.scope("TimelineEventProcessor"))

val eventBusProcessors = Seq(

timelineEventProcessor,

socialWriteEventProcessor,

tweetEventProcessor

)

val uuaProcessor = new UnifiedUserActionProcessor(

gizmoduck = gizmoduck,

tweetypie = tweetypie,

kafkaEventPublisher = kafkaEventPublisher,

userVideoGraphTopic = KafkaEventPublisher.UserVideoTopic,

userVideoGraphBuilder = unifiedUserActionToUserVideoGraphBuilder,

userAdGraphTopic = KafkaEventPublisher.UserAdTopic,

userAdGraphBuilder = unifiedUserActionToUserAdGraphBuilder,

userTweetGraphPlusTopic = KafkaEventPublisher.UserTweetPlusTopic,

userTweetGraphPlusBuilder = unifiedUserActionToUserTweetGraphPlusBuilder)(

statsReceiver.scope("UnifiedUserActionProcessor"))

val uuaConsumer = new UnifiedUserActionsConsumer(uuaProcessor, truststoreLocation())

// Start-up init and graceful shutdown setup

// wait a bit for services to be ready

Thread.sleep(5000L)

log.info("Starting the event processors")

eventBusProcessors.foreach(\_.start())

log.info("Starting the uua processors")

uuaConsumer.atLeastOnceProcessor.start()

this.addAdminRoute(ElfOwlFilter.getPostbackRoute())

onExit {

log.info("Shutting down the event processors")

eventBusProcessors.foreach(\_.stop())

log.info("Shutting down the uua processors")

uuaConsumer.atLeastOnceProcessor.close()

log.info("done exit")

}

// Wait on the thriftServer so that shutdownTimeout is respected.

Await.result(adminHttpServer)

}

}