package com.twitter.timelineranker.server

import com.twitter.conversions.DurationOps.\_

import com.twitter.finagle.mtls.server.MtlsStackServer.\_

import com.twitter.finagle.mux

import com.twitter.finagle.param.Reporter

import com.twitter.finagle.stats.DefaultStatsReceiver

import com.twitter.finagle.util.NullReporterFactory

import com.twitter.finagle.ListeningServer

import com.twitter.finagle.ServiceFactory

import com.twitter.finagle.ThriftMux

import com.twitter.finagle.mtls.authorization.server.MtlsServerSessionTrackerFilter

import com.twitter.finagle.ssl.OpportunisticTls

import com.twitter.finatra.thrift.filters.LoggingMDCFilter

import com.twitter.finatra.thrift.filters.ThriftMDCFilter

import com.twitter.finatra.thrift.filters.TraceIdMDCFilter

import com.twitter.logging.Logger

import com.twitter.server.TwitterServer

import com.twitter.servo.util.MemoizingStatsReceiver

import com.twitter.thriftwebforms.MethodOptions

import com.twitter.thriftwebforms.TwitterServerThriftWebForms

import com.twitter.timelineranker.config.RuntimeConfigurationImpl

import com.twitter.timelineranker.config.TimelineRankerFlags

import com.twitter.timelineranker.thriftscala

import com.twitter.timelines.config.DefaultDynamicConfigStoreFactory

import com.twitter.timelines.config.EmptyDynamicConfigStoreFactory

import com.twitter.timelines.config.Env

import com.twitter.timelines.features.app.ForcibleFeatureValues

import com.twitter.timelines.server.AdminMutableDeciders

import com.twitter.timelines.warmup.NoWarmup

import com.twitter.timelines.warmup.WarmupFlag

import com.twitter.util.Await

import java.net.SocketAddress

import org.apache.thrift.protocol.TBinaryProtocol

import org.apache.thrift.protocol.TCompactProtocol

import org.apache.thrift.protocol.TProtocolFactory

object Main extends TimelineRankerServer

class TimelineRankerServer extends {

override val statsReceiver: MemoizingStatsReceiver = new MemoizingStatsReceiver(

DefaultStatsReceiver)

} with TwitterServer with AdminMutableDeciders with ForcibleFeatureValues with WarmupFlag {

val timelineRankerFlags: TimelineRankerFlags = new TimelineRankerFlags(flag)

lazy val mainLogger: Logger = Logger.get("Main")

private[this] lazy val thriftWebFormsAccess = if (timelineRankerFlags.getEnv == Env.local) {

MethodOptions.Access.Default

} else {

MethodOptions.Access.ByLdapGroup(Seq("timeline-team", "timelineranker-twf-read"))

}

private[this] def mkThriftWebFormsRoutes() =

TwitterServerThriftWebForms[thriftscala.TimelineRanker.MethodPerEndpoint](

thriftServicePort = timelineRankerFlags.servicePort().getPort,

defaultMethodAccess = thriftWebFormsAccess,

methodOptions = TimelineRankerThriftWebForms.methodOptions,

serviceIdentifier = timelineRankerFlags.serviceIdentifier(),

opportunisticTlsLevel = OpportunisticTls.Required,

)

override protected def failfastOnFlagsNotParsed = true

override val defaultCloseGracePeriod = 10.seconds

private[this] def mkServer(

labelSuffix: String,

socketAddress: SocketAddress,

protocolFactory: TProtocolFactory,

serviceFactory: ServiceFactory[Array[Byte], Array[Byte]],

opportunisticTlsLevel: OpportunisticTls.Level,

): ListeningServer = {

val compressor = Seq(mux.transport.Compression.lz4Compressor(highCompression = false))

val decompressor = Seq(mux.transport.Compression.lz4Decompressor())

val compressionLevel =

if (timelineRankerFlags.enableThriftmuxCompression()) {

mux.transport.CompressionLevel.Desired

} else {

mux.transport.CompressionLevel.Off

}

val mtlsSessionTrackerFilter =

new MtlsServerSessionTrackerFilter[mux.Request, mux.Response](statsReceiver)

val loggingMDCFilter = { new LoggingMDCFilter }.toFilter[mux.Request, mux.Response]

val traceIdMDCFilter = { new TraceIdMDCFilter }.toFilter[mux.Request, mux.Response]

val thriftMDCFilter = { new ThriftMDCFilter }.toFilter[mux.Request, mux.Response]

val filters = mtlsSessionTrackerFilter

.andThen(loggingMDCFilter)

.andThen(traceIdMDCFilter)

.andThen(thriftMDCFilter)

ThriftMux.server

// By default, finagle logs exceptions to chickadee, which is deprecated and

// basically unused. To avoid wasted overhead, we explicitly disable the reporter.

.configured(Reporter(NullReporterFactory))

.withLabel("timelineranker." + labelSuffix)

.withMutualTls(timelineRankerFlags.getServiceIdentifier)

.withOpportunisticTls(opportunisticTlsLevel)

.withProtocolFactory(protocolFactory)

.withCompressionPreferences.compression(compressionLevel, compressor)

.withCompressionPreferences.decompression(compressionLevel, decompressor)

.filtered(filters)

.serve(socketAddress, serviceFactory)

}

def main(): Unit = {

try {

val parsedOpportunisticTlsLevel = OpportunisticTls.Values

.find(

\_.value.toLowerCase == timelineRankerFlags.opportunisticTlsLevel().toLowerCase).getOrElse(

OpportunisticTls.Desired)

TwitterServerThriftWebForms.addAdminRoutes(this, mkThriftWebFormsRoutes())

addAdminMutableDeciderRoutes(timelineRankerFlags.getEnv)

val configStoreFactory = if (timelineRankerFlags.getEnv == Env.local) {

EmptyDynamicConfigStoreFactory

} else {

new DefaultDynamicConfigStoreFactory

}

val runtimeConfiguration = new RuntimeConfigurationImpl(

timelineRankerFlags,

configStoreFactory,

decider,

forcedFeatureValues = getFeatureSwitchOverrides,

statsReceiver

)

val timelineRankerBuilder = new TimelineRankerBuilder(runtimeConfiguration)

val warmup = if (shouldWarmup) {

new Warmup(

timelineRankerBuilder.timelineRanker,

runtimeConfiguration.underlyingClients.timelineRankerForwardingClient,

mainLogger

)

} else {

new NoWarmup()

}

warmup.prebindWarmup()

// Create Thrift services that use the binary Thrift protocol, and the compact one.

val server =

mkServer(

"binary",

timelineRankerFlags.servicePort(),

new TBinaryProtocol.Factory(),

timelineRankerBuilder.serviceFactory,

parsedOpportunisticTlsLevel,

)

val compactServer =

mkServer(

"compact",

timelineRankerFlags.serviceCompactPort(),

new TCompactProtocol.Factory(),

timelineRankerBuilder.compactProtocolServiceFactory,

parsedOpportunisticTlsLevel,

)

mainLogger.info(

s"Thrift binary server bound to service port [${timelineRankerFlags.servicePort()}]")

closeOnExit(server)

mainLogger.info(

s"Thrift compact server bound to service port [${timelineRankerFlags.serviceCompactPort()}]")

closeOnExit(compactServer)

warmup.warmupComplete()

mainLogger.info("ready: server")

Await.ready(server)

Await.ready(compactServer)

} catch {

case e: Throwable =>

e.printStackTrace()

mainLogger.error(e, s"failure in main")

System.exit(1)

}

}

}