package com.twitter.timelineranker.config

import com.twitter.abdecider.ABDeciderFactory

import com.twitter.abdecider.LoggingABDecider

import com.twitter.decider.Decider

import com.twitter.featureswitches.Value

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.servo.decider.DeciderGateBuilder

import com.twitter.servo.util.Effect

import com.twitter.timelineranker.decider.DeciderKey

import com.twitter.timelines.authorization.TimelinesClientRequestAuthorizer

import com.twitter.timelines.config.\_

import com.twitter.timelines.config.configapi.\_

import com.twitter.timelines.features.\_

import com.twitter.timelines.util.ImpressionCountingABDecider

import com.twitter.timelines.util.logging.Scribe

import com.twitter.util.Await

import com.twitter.servo.util.Gate

import com.twitter.timelines.model.UserId

trait ClientProvider {

def clientWrappers: ClientWrappers

def underlyingClients: UnderlyingClientConfiguration

}

trait UtilityProvider {

def abdecider: LoggingABDecider

def clientRequestAuthorizer: TimelinesClientRequestAuthorizer

def configStore: ConfigStore

def decider: Decider

def deciderGateBuilder: DeciderGateBuilder

def employeeGate: UserRolesGate.EmployeeGate

def configApiConfiguration: ConfigApiConfiguration

def statsReceiver: StatsReceiver

def whitelist: UserList

}

trait RuntimeConfiguration extends ClientProvider with UtilityProvider with ConfigUtils {

def isProd: Boolean

def maxConcurrency: Int

def clientEventScribe: Effect[String]

def clientWrapperFactories: ClientWrapperFactories

}

class RuntimeConfigurationImpl(

flags: TimelineRankerFlags,

configStoreFactory: DynamicConfigStoreFactory,

val decider: Decider,

val forcedFeatureValues: Map[String, Value] = Map.empty[String, Value],

val statsReceiver: StatsReceiver)

extends RuntimeConfiguration {

// Creates and initialize config store as early as possible so other parts could have a dependency on it for settings.

override val configStore: DynamicConfigStore =

configStoreFactory.createDcEnvAwareFileBasedConfigStore(

relativeConfigFilePath = "timelines/timelineranker/service\_settings.yml",

dc = flags.getDatacenter,

env = flags.getEnv,

configBusConfig = ConfigBusProdConfig,

onUpdate = ConfigStore.NullOnUpdateCallback,

statsReceiver = statsReceiver

)

Await.result(configStore.init)

val environment: Env.Value = flags.getEnv

override val isProd: Boolean = isProdEnv(environment)

val datacenter: Datacenter.Value = flags.getDatacenter

val abDeciderPath = "/usr/local/config/abdecider/abdecider.yml"

override val maxConcurrency: Int = flags.maxConcurrency()

val deciderGateBuilder: DeciderGateBuilder = new DeciderGateBuilder(decider)

val clientRequestAuthorizer: TimelinesClientRequestAuthorizer =

new TimelinesClientRequestAuthorizer(

deciderGateBuilder = deciderGateBuilder,

clientDetails = ClientAccessPermissions.All,

unknownClientDetails = ClientAccessPermissions.unknown,

clientAuthorizationGate =

deciderGateBuilder.linearGate(DeciderKey.ClientRequestAuthorization),

clientWriteWhitelistGate = deciderGateBuilder.linearGate(DeciderKey.ClientWriteWhitelist),

globalCapacityQPS = flags.requestRateLimit(),

statsReceiver = statsReceiver

)

override val clientEventScribe = Scribe.clientEvent(isProd, statsReceiver)

val abdecider: LoggingABDecider = new ImpressionCountingABDecider(

abdecider = ABDeciderFactory.withScribeEffect(

abDeciderYmlPath = abDeciderPath,

scribeEffect = clientEventScribe,

decider = None,

environment = Some("production"),

).buildWithLogging(),

statsReceiver = statsReceiver

)

val underlyingClients: UnderlyingClientConfiguration = buildUnderlyingClientConfiguration

val clientWrappers: ClientWrappers = new ClientWrappers(this)

override val clientWrapperFactories: ClientWrapperFactories = new ClientWrapperFactories(this)

private[this] val userRolesCacheFactory = new UserRolesCacheFactory(

userRolesService = underlyingClients.userRolesServiceClient,

statsReceiver = statsReceiver

)

override val whitelist: Whitelist = Whitelist(

configStoreFactory = configStoreFactory,

userRolesCacheFactory = userRolesCacheFactory,

statsReceiver = statsReceiver

)

override val employeeGate: Gate[UserId] = UserRolesGate(

userRolesCacheFactory.create(UserRoles.EmployeesRoleName)

)

private[this] val featureRecipientFactory =

new UserRolesCachingFeatureRecipientFactory(userRolesCacheFactory, statsReceiver)

override val configApiConfiguration: FeatureSwitchesV2ConfigApiConfiguration =

FeatureSwitchesV2ConfigApiConfiguration(

datacenter = flags.getDatacenter,

serviceName = ServiceName.TimelineRanker,

abdecider = abdecider,

featureRecipientFactory = featureRecipientFactory,

forcedValues = forcedFeatureValues,

statsReceiver = statsReceiver

)

private[this] def buildUnderlyingClientConfiguration: UnderlyingClientConfiguration = {

environment match {

case Env.prod => new DefaultUnderlyingClientConfiguration(flags, statsReceiver)

case \_ => new StagingUnderlyingClientConfiguration(flags, statsReceiver)

}

}

}