package com.twitter.tweetypie.config

import com.fasterxml.jackson.databind.ObjectMapper

import com.fasterxml.jackson.dataformat.yaml.YAMLFactory

import com.fasterxml.jackson.module.scala.DefaultScalaModule

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

import com.twitter.util.Try

case object EmptyConfigException extends Exception

case class ServiceIdentifierPattern(

role: Option[String],

service: Option[String],

environment: Option[String],

) {

// Service identifier matches if the fields of service identifier

// match all the defined fields of pattern.

def matches(id: ServiceIdentifier): Boolean =

Seq(

role.map(\_ == id.role),

service.map(\_ == id.service),

environment.map(\_ == id.environment),

)

.flatten

.forall(identity)

// True if this is the kind of pattern that only specifies environment.

// This should be used in rare cases, for example letting all devel clients

// use permitted methods - like get\_tweet\_fields.

def onlyEnv: Boolean =

role.isEmpty && service.isEmpty && environment.isDefined

}

case class Client(

clientId: String,

serviceIdentifiers: Seq[ServiceIdentifierPattern],

tpsLimit: Option[Int],

environments: Seq[String],

loadShedEnvs: Seq[String],

permittedMethods: Set[String],

accessAllMethods: Boolean,

bypassVisibilityFiltering: Boolean,

enforceRateLimit: Boolean) {

// Client matches a service identifier if any of its patterns

// match.

def matches(id: ServiceIdentifier): Boolean =

serviceIdentifiers.exists(\_.matches(id))

}

object ClientsParser {

// Case classes for parsing yaml - should match the structure of clients.yml

private case class YamlServiceIdentifier(

role: Option[String],

service: Option[String],

environment: Option[String],

)

private case class YamlClient(

client\_id: String,

service\_identifiers: Option[Seq[YamlServiceIdentifier]],

service\_name: String,

tps\_quota: String,

contact\_email: String,

environments: Seq[String],

load\_shed\_envs: Option[

Seq[String]

], // list of environments we can rejects requests from if load shedding

comment: Option[String],

permitted\_methods: Option[Seq[String]],

access\_all\_methods: Boolean,

bypass\_visibility\_filtering: Boolean,

bypass\_visibility\_filtering\_reason: Option[String],

rate\_limit: Boolean) {

def toClient: Client = {

// we provision tps\_quota for both DCs during white-listing, to account for full fail-over.

val tpsLimit: Option[Int] = Try(tps\_quota.replaceAll("[^0-9]", "").toInt \* 1000).toOption

Client(

clientId = client\_id,

serviceIdentifiers = service\_identifiers.getOrElse(Nil).flatMap { id =>

if (id.role.isDefined || id.service.isDefined || id.environment.isDefined) {

Seq(ServiceIdentifierPattern(

role = id.role,

service = id.service,

environment = id.environment,

))

} else {

Seq()

}

},

tpsLimit = tpsLimit,

environments = environments,

loadShedEnvs = load\_shed\_envs.getOrElse(Nil),

permittedMethods = permitted\_methods.getOrElse(Nil).toSet,

accessAllMethods = access\_all\_methods,

bypassVisibilityFiltering = bypass\_visibility\_filtering,

enforceRateLimit = rate\_limit

)

}

}

private val mapper: ObjectMapper = new ObjectMapper(new YAMLFactory())

mapper.registerModule(DefaultScalaModule)

private val yamlClientTypeFactory =

mapper

.getTypeFactory()

.constructCollectionLikeType(

classOf[Seq[YamlClient]],

classOf[YamlClient]

)

def apply(yamlString: String): Seq[Client] = {

val parsed =

mapper

.readValue[Seq[YamlClient]](yamlString, yamlClientTypeFactory)

.map(\_.toClient)

if (parsed.isEmpty)

throw EmptyConfigException

else

parsed

}

}