package com.twitter.simclusters\_v2.summingbird.common

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

import com.twitter.simclusters\_v2.common.ModelVersions.\_

import com.twitter.simclusters\_v2.summingbird.common.ClientConfigs.\_

import com.twitter.simclusters\_v2.summingbird.common.SimClustersProfile.AltSetting.AltSetting

import com.twitter.simclusters\_v2.summingbird.common.SimClustersProfile.Environment.Environment

import com.twitter.simclusters\_v2.summingbird.common.SimClustersProfile.JobType.JobType

import com.twitter.simclusters\_v2.summingbird.common.SimClustersProfile.AltSetting

import com.twitter.simclusters\_v2.summingbird.common.SimClustersProfile.JobType

import com.twitter.simclusters\_v2.thriftscala.EmbeddingType

import com.twitter.simclusters\_v2.thriftscala.ModelVersion

sealed trait SimClustersProfile {

val env: Environment

val alt: AltSetting

val modelVersionStr: String

lazy val modelVersion: ModelVersion = modelVersionStr

}

sealed trait SimClustersJobProfile extends SimClustersProfile {

val jobType: JobType

final lazy val jobName: String = {

alt match {

case AltSetting.Alt =>

s"simclusters\_v2\_${jobType}\_alt\_job\_$env"

case AltSetting.Esc =>

s"simclusters\_v2\_${jobType}\_esc\_job\_$env"

case \_ =>

s"simclusters\_v2\_${jobType}\_job\_$env"

}

}

// Build the serviceIdentifier by jobType, env and zone(dc)

final lazy val serviceIdentifier: String => ServiceIdentifier = { zone =>

ServiceIdentifier(Configs.role, s"summingbird\_$jobName", env.toString, zone)

}

final lazy val favScoreThresholdForUserInterest: Double =

Configs.favScoreThresholdForUserInterest(modelVersionStr)

lazy val timelineEventSourceSubscriberId: String = {

val jobTypeStr = jobType match {

case JobType.MultiModelTweet => "multi\_model\_tweet\_"

case JobType.PersistentTweet => "persistent\_tweet\_"

case JobType.Tweet => ""

}

val prefix = alt match {

case AltSetting.Alt =>

"alt\_"

case AltSetting.Esc =>

"esc\_"

case \_ =>

""

}

s"simclusters\_v2\_${jobTypeStr}summingbird\_$prefix$env"

}

}

object SimClustersProfile {

object JobType extends Enumeration {

type JobType = Value

val Tweet: JobType = Value("tweet")

val PersistentTweet: JobType = Value("persistent\_tweet")

val MultiModelTweet: JobType = Value("multimodel\_tweet")

}

object Environment extends Enumeration {

type Environment = Value

val Prod: Environment = Value("prod")

val Devel: Environment = Value("devel")

def apply(setting: String): Environment = {

if (setting == Prod.toString) {

Prod

} else {

Devel

}

}

}

object AltSetting extends Enumeration {

type AltSetting = Value

val Normal: AltSetting = Value("normal")

val Alt: AltSetting = Value("alt")

val Esc: AltSetting = Value("esc")

def apply(setting: String): AltSetting = {

setting match {

case "alt" => Alt

case "esc" => Esc

case \_ => Normal

}

}

}

case class SimClustersTweetProfile(

env: Environment,

alt: AltSetting,

modelVersionStr: String,

entityClusterScorePath: String,

tweetTopKClustersPath: String,

clusterTopKTweetsPath: String,

coreEmbeddingType: EmbeddingType,

clusterTopKTweetsLightPath: Option[String] = None)

extends SimClustersJobProfile {

final val jobType: JobType = JobType.Tweet

}

case class PersistentTweetProfile(

env: Environment,

alt: AltSetting,

modelVersionStr: String,

persistentTweetStratoPath: String,

coreEmbeddingType: EmbeddingType)

extends SimClustersJobProfile {

final val jobType: JobType = JobType.PersistentTweet

}

final val AltProdTweetJobProfile = SimClustersTweetProfile(

env = Environment.Prod,

alt = AltSetting.Alt,

modelVersionStr = Model20M145K2020,

entityClusterScorePath = simClustersCoreAltCachePath,

tweetTopKClustersPath = simClustersCoreAltCachePath,

clusterTopKTweetsPath = simClustersCoreAltCachePath,

clusterTopKTweetsLightPath = Some(simClustersCoreAltLightCachePath),

coreEmbeddingType = EmbeddingType.LogFavBasedTweet

)

final val AltDevelTweetJobProfile = SimClustersTweetProfile(

env = Environment.Devel,

alt = AltSetting.Alt,

modelVersionStr = Model20M145K2020,

// using the same devel cache with job

entityClusterScorePath = develSimClustersCoreCachePath,

tweetTopKClustersPath = develSimClustersCoreCachePath,

clusterTopKTweetsPath = develSimClustersCoreCachePath,

clusterTopKTweetsLightPath = Some(develSimClustersCoreLightCachePath),

coreEmbeddingType = EmbeddingType.LogFavBasedTweet,

)

final val ProdPersistentTweetProfile = PersistentTweetProfile(

env = Environment.Prod,

alt = AltSetting.Normal,

modelVersionStr = Model20M145K2020,

// This profile is used by the persistent tweet embedding job to update the embedding. We

// use the uncached column to avoid reading stale data

persistentTweetStratoPath = logFavBasedTweet20M145K2020UncachedStratoPath,

coreEmbeddingType = EmbeddingType.LogFavBasedTweet

)

final val DevelPersistentTweetProfile = PersistentTweetProfile(

env = Environment.Devel,

alt = AltSetting.Normal,

modelVersionStr = Model20M145K2020,

persistentTweetStratoPath = develLogFavBasedTweet20M145K2020StratoPath,

coreEmbeddingType = EmbeddingType.LogFavBasedTweet

)

def fetchTweetJobProfile(

env: Environment,

alt: AltSetting = AltSetting.Normal

): SimClustersTweetProfile = {

(env, alt) match {

case (Environment.Prod, AltSetting.Alt) => AltProdTweetJobProfile

case (Environment.Devel, AltSetting.Alt) => AltDevelTweetJobProfile

case \_ => throw new IllegalArgumentException("Invalid env or alt setting")

}

}

def fetchPersistentJobProfile(

env: Environment,

alt: AltSetting = AltSetting.Normal

): PersistentTweetProfile = {

(env, alt) match {

case (Environment.Prod, AltSetting.Normal) => ProdPersistentTweetProfile

case (Environment.Devel, AltSetting.Normal) => DevelPersistentTweetProfile

case \_ => throw new IllegalArgumentException("Invalid env or alt setting")

}

}

/\*\*

\* For short term, fav based tweet embedding and log fav based tweets embedding exists at the

\* same time. We want to move to log fav based tweet embedding eventually.

\* Follow based tweet embeddings exists in both environment.

\* A uniform tweet embedding API is the future to replace the existing use case.

\*/

final lazy val tweetJobProfileMap: Environment => Map[

(EmbeddingType, String),

SimClustersTweetProfile

] = {

case Environment.Prod =>

Map(

(EmbeddingType.LogFavBasedTweet, Model20M145K2020) -> AltProdTweetJobProfile

)

case Environment.Devel =>

Map(

(EmbeddingType.LogFavBasedTweet, Model20M145K2020) -> AltDevelTweetJobProfile

)

}

}