package com.twitter.timelines.data\_processing.ml\_util.aggregation\_framework.heron

import com.twitter.finagle.mtls.authentication.EmptyServiceIdentifier

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

import com.twitter.ml.api.DataRecord

import com.twitter.summingbird.Options

import com.twitter.timelines.data\_processing.ml\_util.transforms.OneToSomeTransform

/\*\*

\*

\* @param appId application id for topology job

\* @param topologyWorkers number of workers/containers of topology

\* @param sourceCount number of parallel sprouts of topology

\* @param summerCount number of Summer of topology

\* @param cacheSize number of tuples a Summer awaits before aggregation.

\* @param flatMapCount number of parallel FlatMap of topology

\* @param containerRamGigaBytes total RAM of each worker/container has

\* @param name name of topology job

\* @param teamName name of team who owns topology job

\* @param teamEmail email of team who owns topology job

\* @param componentsToKerberize component of topology job (eg. Tail-FlatMap-Source) which enables kerberization

\* @param componentToMetaSpaceSizeMap MetaSpaceSize settings for components of topology job

\* @param topologyNamedOptions Sets spout allocations for named topology components

\* @param serviceIdentifier represents the identifier used for Service to Service Authentication

\* @param onlinePreTransforms sequential data record transforms applied to Producer of DataRecord before creating AggregateGroup.

\* While preTransforms defined at AggregateGroup are applied to each aggregate group, onlinePreTransforms are applied to the whole producer source.

\* @param keyedByUserEnabled boolean value to enable/disable merging user-level features from Feature Store

\* @param keyedByAuthorEnabled boolean value to enable/disable merging author-level features from Feature Store

\* @param enableUserReindexingNighthawkBtreeStore boolean value to enable reindexing RTAs on user id with btree backed nighthawk

\* @param enableUserReindexingNighthawkHashStore boolean value to enable reindexing RTAs on user id with hash backed nighthawk

\* @param userReindexingNighthawkBtreeStoreConfig NH btree store config used in reindexing user RTAs

\* @param userReindexingNighthawkHashStoreConfig NH hash store config used in reindexing user RTAs

\*/

case class RealTimeAggregatesJobConfig(

appId: String,

topologyWorkers: Int,

sourceCount: Int,

summerCount: Int,

cacheSize: Int,

flatMapCount: Int,

containerRamGigaBytes: Int,

name: String,

teamName: String,

teamEmail: String,

componentsToKerberize: Seq[String] = Seq.empty,

componentToMetaSpaceSizeMap: Map[String, String] = Map.empty,

componentToRamGigaBytesMap: Map[String, Int] = Map("Tail" -> 4),

topologyNamedOptions: Map[String, Options] = Map.empty,

serviceIdentifier: ServiceIdentifier = EmptyServiceIdentifier,

onlinePreTransforms: Seq[OneToSomeTransform] = Seq.empty,

keyedByUserEnabled: Boolean = false,

keyedByAuthorEnabled: Boolean = false,

keyedByTweetEnabled: Boolean = false,

enableUserReindexingNighthawkBtreeStore: Boolean = false,

enableUserReindexingNighthawkHashStore: Boolean = false,

userReindexingNighthawkBtreeStoreConfig: NighthawkUnderlyingStoreConfig =

NighthawkUnderlyingStoreConfig(),

userReindexingNighthawkHashStoreConfig: NighthawkUnderlyingStoreConfig =

NighthawkUnderlyingStoreConfig()) {

/\*\*

\* Apply transforms sequentially. If any transform results in a dropped (None)

\* DataRecord, then entire transform sequence will result in a dropped DataRecord.

\* Note that transforms are order-dependent.

\*/

def sequentiallyTransform(dataRecord: DataRecord): Option[DataRecord] = {

val recordOpt = Option(new DataRecord(dataRecord))

onlinePreTransforms.foldLeft(recordOpt) {

case (Some(previousRecord), preTransform) =>

preTransform(previousRecord)

case \_ => Option.empty[DataRecord]

}

}

}

trait RealTimeAggregatesJobConfigs {

def Prod: RealTimeAggregatesJobConfig

def Devel: RealTimeAggregatesJobConfig

}