package com.twitter.product\_mixer.component\_library.experiments.metrics

import com.twitter.util.Return

import com.twitter.util.Throw

import com.twitter.util.Try

object MetricDefinition {

val SingleQuote = """""""

val DoubleQuote = """"""""

}

/\*\*

\* Base class for all metric definitions

\*/

sealed trait MetricDefinition {

def toCsvField: Seq[String]

val metricDefinitionType: String

}

/\*\*

\* Pattern Metric Definition

\* @param pattern the regex pattern for this metric

\*/

case class NamedPatternMetricDefinition(

pattern: Seq[String])

extends MetricDefinition {

override def toCsvField: Seq[String] = pattern

override val metricDefinitionType: String = "NAMED\_PATTERN"

}

/\*\*

\* Strainer Metric Definition

\* @param strainerExpression a filter on top of client events

\*/

case class StrainerMetricDefinition(

strainerExpression: String)

extends MetricDefinition {

import MetricDefinition.\_

override def toCsvField: Seq[String] = {

Seq(strainerExpression.replaceAll(SingleQuote, DoubleQuote))

}

override val metricDefinitionType: String = "STRAINER"

}

/\*\*

\* Lambda Metric Definition

\* @param lambdaExpression a scala function mapping client events to a double

\*/

case class LambdaMetricDefinition(

lambdaExpression: String)

extends MetricDefinition {

import MetricDefinition.\_

override def toCsvField: Seq[String] = {

Seq(lambdaExpression.replaceAll(SingleQuote, DoubleQuote))

}

override val metricDefinitionType: String = "LAMBDA"

}

case class BucketRatioMetricDefinition(

numerator: String,

denominator: String)

extends MetricDefinition {

override def toCsvField: Seq[String] = {

Seq(s"(${numerator}) / (${denominator})")

}

override val metricDefinitionType: String = "BUCKET\_RATIO"

}

object Metric {

val bucketRatioPattern = "[(]+(.+)[)]+ / [(]+(.+)[)]+".r

/\*\*

\* Creates a new Metric given a template line.

\* @param line semicolon separated line string

\* ignore line with comment, represented by hashtag at the beginning of the line

\* @throws RuntimeException if the line is invalid

\*/

def fromLine(line: String): Metric = {

val splits = line.split(";")

// at least two parts separated by semicolon (third part is optional)

if (splits.lengthCompare(2) >= 0) {

val metricExpression = splits(0)

val metricName = splits(1)

val metricDefinition = Try(splits(2)) match {

case Return("NAMED\_PATTERN") => NamedPatternMetricDefinition(Seq(metricExpression))

case Return("STRAINER") => StrainerMetricDefinition(metricExpression)

case Return("LAMBDA") => LambdaMetricDefinition(metricExpression)

case Return("BUCKET\_RATIO") =>

metricExpression match {

case bucketRatioPattern(numerator, denominator) =>

BucketRatioMetricDefinition(numerator, denominator)

case \_ =>

throw new RuntimeException(

s"Invalid metric definition for Bucket Ratio. Expected format (numerator)<space>/<space>(denominator) but found $metricExpression")

}

case Return(other) =>

throw new RuntimeException(s"Invalid metric definition in line in template file: $line")

// default to named pattern

case Throw(\_) => NamedPatternMetricDefinition(List(metricExpression))

}

Metric(metricName, metricDefinition)

} else {

throw new RuntimeException(s"Invalid line in template file: $line")

}

}

}

/\*\*

\*

\* @param name globally unique metric name (current DDG limitation)

\* @param definition the metric definition for this metric

\*/

case class Metric(

name: String,

definition: MetricDefinition)