package com.twitter.product\_mixer.core.pipeline.step.quality\_factor

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.product\_mixer.core.model.common.identifier.ComponentIdentifier

import com.twitter.product\_mixer.core.pipeline.PipelineQuery

import com.twitter.product\_mixer.core.pipeline.state.HasQuery

import com.twitter.product\_mixer.core.pipeline.step.Step

import com.twitter.product\_mixer.core.quality\_factor.HasQualityFactorStatus

import com.twitter.product\_mixer.core.quality\_factor.QualityFactorStatus

import com.twitter.product\_mixer.core.service.Executor

import com.twitter.product\_mixer.core.service.ExecutorResult

import com.twitter.stitch.Arrow

import javax.inject.Inject

/\*\*

\* Quality Factor building step that builds up the state snapshot for a map of configs.

\*

\* @param statsReceiver Stats Receiver used to build finagle gauges for QF State

\*

\* @tparam Query Pipeline query model with quality factor status

\* @tparam State The pipeline state domain model.

\*/

case class QualityFactorStep[

Query <: PipelineQuery with HasQualityFactorStatus,

State <: HasQuery[Query, State]] @Inject() (

statsReceiver: StatsReceiver)

extends Step[

State,

QualityFactorStepConfig,

Any,

QualityFactorStepResult

] {

override def isEmpty(config: QualityFactorStepConfig): Boolean =

config.qualityFactorStatus.qualityFactorByPipeline.isEmpty

override def adaptInput(

state: State,

config: QualityFactorStepConfig

): Any = ()

override def arrow(

config: QualityFactorStepConfig,

context: Executor.Context

): Arrow[Any, QualityFactorStepResult] = {

// We use provideGauge so these gauges live forever even without a reference.

val currentValues = config.qualityFactorStatus.qualityFactorByPipeline.map {

case (identifier, qualityFactor) =>

// QF is a relative stat (since the parent pipeline is monitoring a child pipeline)

val scopes = config.pipelineIdentifier.toScopes ++ identifier.toScopes :+ "QualityFactor"

val currentValue = qualityFactor.currentValue.toFloat

statsReceiver.provideGauge(scopes: \_\*) {

currentValue

}

identifier -> currentValue

}

Arrow.value(QualityFactorStepResult(currentValues))

}

override def updateState(

state: State,

executorResult: QualityFactorStepResult,

config: QualityFactorStepConfig

): State = state.updateQuery(

state.query.withQualityFactorStatus(config.qualityFactorStatus).asInstanceOf[Query])

}

case class QualityFactorStepConfig(

pipelineIdentifier: ComponentIdentifier,

qualityFactorStatus: QualityFactorStatus)

case class QualityFactorStepResult(currentValues: Map[ComponentIdentifier, Float])

extends ExecutorResult