package com.twitter.search.ingester.pipeline.twitter;

import java.util.concurrent.TimeUnit;

import javax.naming.NamingException;

import scala.runtime.BoxedUnit;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.base.Preconditions;

import org.apache.commons.pipeline.Pipeline;

import org.apache.commons.pipeline.StageDriver;

import org.apache.thrift.TBase;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.eventbus.client.EventBusSubscriber;

import com.twitter.search.common.decider.SearchDecider;

import com.twitter.search.common.metrics.SearchCounter;

import com.twitter.search.ingester.model.PromiseContainer;

import com.twitter.search.ingester.pipeline.util.PipelineUtil;

import com.twitter.util.Await;

import com.twitter.util.Function;

import com.twitter.util.Future;

import com.twitter.util.Promise;

public abstract class EventBusReaderStage<T extends TBase<?, ?>> extends TwitterBaseStage

<Void, Void> {

private static final Logger LOG = LoggerFactory.getLogger(EventBusReaderStage.class);

private static final int DECIDER\_POLL\_INTERVAL\_IN\_SECS = 5;

private SearchCounter totalEventsCount;

private String environment = null;

private String eventBusReaderEnabledDeciderKey;

private StageDriver stageDriver;

private EventBusSubscriber<T> eventBusSubscriber = null;

// XML configuration options

private String eventBusSubscriberId;

private int maxConcurrentEvents;

private SearchDecider searchDecider;

protected EventBusReaderStage() {

}

@Override

protected void initStats() {

super.initStats();

totalEventsCount = SearchCounter.export(getStageNamePrefix() + "\_total\_events\_count");

}

@Override

protected void doInnerPreprocess() throws NamingException {

searchDecider = new SearchDecider(decider);

if (stageDriver == null) {

stageDriver = ((Pipeline) stageContext).getStageDriver(this);

}

eventBusReaderEnabledDeciderKey = String.format(

getDeciderKeyTemplate(),

earlybirdCluster.getNameForStats(),

environment);

PipelineUtil.feedStartObjectToStage(this);

}

protected abstract PromiseContainer<BoxedUnit, T> eventAndPromiseToContainer(

T incomingEvent,

Promise<BoxedUnit> p);

private Future<BoxedUnit> processEvent(T incomingEvent) {

Promise<BoxedUnit> p = new Promise<>();

PromiseContainer<BoxedUnit, T> promiseContainer = eventAndPromiseToContainer(incomingEvent, p);

totalEventsCount.increment();

emitAndCount(promiseContainer);

return p;

}

private void closeEventBusSubscriber() throws Exception {

if (eventBusSubscriber != null) {

Await.result(eventBusSubscriber.close());

eventBusSubscriber = null;

}

}

protected abstract Class<T> getThriftClass();

protected abstract String getDeciderKeyTemplate();

private void startUpEventBusSubscriber() {

// Start reading from eventbus if it is null

if (eventBusSubscriber == null) {

//noinspection unchecked

eventBusSubscriber = wireModule.createEventBusSubscriber(

Function.func(this::processEvent),

getThriftClass(),

eventBusSubscriberId,

maxConcurrentEvents);

}

Preconditions.checkNotNull(eventBusSubscriber);

}

/\*\*

\* This is only kicked off once with a start object which is ignored. Then we loop

\* checking the decider. If it turns off then we close the eventbus reader,

\* and if it turns on, then we create a new eventbus reader.

\*

\* @param obj ignored

\*/

@Override

public void innerProcess(Object obj) {

boolean interrupted = false;

Preconditions.checkNotNull("The environment is not set.", environment);

int previousEventBusReaderEnabledAvailability = 0;

while (stageDriver.getState() == StageDriver.State.RUNNING) {

int eventBusReaderEnabledAvailability =

searchDecider.getAvailability(eventBusReaderEnabledDeciderKey);

if (previousEventBusReaderEnabledAvailability != eventBusReaderEnabledAvailability) {

LOG.info("EventBusReaderStage availability decider changed from {} to {}.",

previousEventBusReaderEnabledAvailability, eventBusReaderEnabledAvailability);

// If the availability is 0 then disable the reader, otherwise read from EventBus.

if (eventBusReaderEnabledAvailability == 0) {

try {

closeEventBusSubscriber();

} catch (Exception e) {

LOG.warn("Exception while closing eventbus subscriber", e);

}

} else {

startUpEventBusSubscriber();

}

}

previousEventBusReaderEnabledAvailability = eventBusReaderEnabledAvailability;

try {

clock.waitFor(TimeUnit.SECONDS.toMillis(DECIDER\_POLL\_INTERVAL\_IN\_SECS));

} catch (InterruptedException e) {

interrupted = true;

}

}

LOG.info("StageDriver is not RUNNING anymore, closing EventBus subscriber");

try {

closeEventBusSubscriber();

} catch (InterruptedException e) {

interrupted = true;

} catch (Exception e) {

LOG.warn("Exception while closing eventbus subscriber", e);

} finally {

if (interrupted) {

Thread.currentThread().interrupt();

}

}

}

// This is needed to set the value from XML config.

public void setEventBusSubscriberId(String eventBusSubscriberId) {

this.eventBusSubscriberId = eventBusSubscriberId;

LOG.info("EventBusReaderStage with eventBusSubscriberId: {}", eventBusSubscriberId);

}

// This is needed to set the value from XML config.

public void setEnvironment(String environment) {

this.environment = environment;

LOG.info("Ingester is running in {}", environment);

}

// This is needed to set the value from XML config.

public void setMaxConcurrentEvents(int maxConcurrentEvents) {

this.maxConcurrentEvents = maxConcurrentEvents;

}

@VisibleForTesting

public void setStageDriver(StageDriver stageDriver) {

this.stageDriver = stageDriver;

}

}