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

import com.google.common.annotations.VisibleForTesting;

import org.apache.commons.pipeline.StageException;

import org.apache.commons.pipeline.validation.ConsumedTypes;

import org.apache.commons.pipeline.validation.ProducedTypes;

import org.apache.thrift.TDeserializer;

import org.apache.thrift.TException;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.search.common.debug.DebugEventUtil;

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

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

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

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

/\*\*

\* Deserializes {@link KafkaRawRecord} into IngesterTweetEvent and emits those.

\*/

@ConsumedTypes(KafkaRawRecord.class)

@ProducedTypes(IngesterTweetEvent.class)

public class TweetEventDeserializerStage extends TwitterBaseStage

<KafkaRawRecord, IngesterTweetEvent> {

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

// Limit how much the logs get polluted

private static final int MAX\_OOM\_SERIALIZED\_BYTES\_LOGGED = 5000;

private static final char[] HEX\_ARRAY = "0123456789ABCDEF".toCharArray();

private final TDeserializer deserializer = new TDeserializer();

private SearchCounter outOfMemoryErrors;

private SearchCounter outOfMemoryErrors2;

private SearchCounter totalEventsCount;

private SearchCounter validEventsCount;

private SearchCounter deserializationErrorsCount;

@Override

public void initStats() {

super.initStats();

innerSetupStats();

}

@Override

protected void innerSetupStats() {

outOfMemoryErrors = SearchCounter.export(getStageNamePrefix() + "\_out\_of\_memory\_errors");

outOfMemoryErrors2 = SearchCounter.export(getStageNamePrefix() + "\_out\_of\_memory\_errors\_2");

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

validEventsCount = SearchCounter.export(getStageNamePrefix() + "\_valid\_events\_count");

deserializationErrorsCount =

SearchCounter.export(getStageNamePrefix() + "\_deserialization\_errors\_count");

}

@Override

public void innerProcess(Object obj) throws StageException {

if (!(obj instanceof KafkaRawRecord)) {

throw new StageException(this, "Object is not a KafkaRawRecord: " + obj);

}

KafkaRawRecord kafkaRecord = (KafkaRawRecord) obj;

IngesterTweetEvent tweetEvent = tryDeserializeRecord(kafkaRecord);

if (tweetEvent != null) {

emitAndCount(tweetEvent);

}

}

@Override

protected IngesterTweetEvent innerRunStageV2(KafkaRawRecord kafkaRawRecord) {

IngesterTweetEvent ingesterTweetEvent = tryDeserializeRecord(kafkaRawRecord);

if (ingesterTweetEvent == null) {

throw new PipelineStageRuntimeException("failed to deserialize KafkaRawRecord : "

+ kafkaRawRecord);

}

return ingesterTweetEvent;

}

private IngesterTweetEvent tryDeserializeRecord(KafkaRawRecord kafkaRecord) {

try {

totalEventsCount.increment();

IngesterTweetEvent tweetEvent = deserialize(kafkaRecord);

validEventsCount.increment();

return tweetEvent;

} catch (OutOfMemoryError e) {

try {

outOfMemoryErrors.increment();

byte[] bytes = kafkaRecord.getData();

int limit = Math.min(bytes.length, MAX\_OOM\_SERIALIZED\_BYTES\_LOGGED);

StringBuilder sb = new StringBuilder(2 \* limit + 100)

.append("OutOfMemoryError deserializing ").append(bytes.length).append(" bytes: ");

appendBytesAsHex(sb, bytes, MAX\_OOM\_SERIALIZED\_BYTES\_LOGGED);

LOG.error(sb.toString(), e);

} catch (OutOfMemoryError e2) {

outOfMemoryErrors2.increment();

}

}

return null;

}

private IngesterTweetEvent deserialize(KafkaRawRecord kafkaRecord) {

try {

IngesterTweetEvent ingesterTweetEvent = new IngesterTweetEvent();

synchronized (this) {

deserializer.deserialize(ingesterTweetEvent, kafkaRecord.getData());

}

// Record the created\_at time and then we first saw this tweet in the ingester for tracking

// down the ingestion pipeline.

addDebugEventsToIncomingTweet(ingesterTweetEvent, kafkaRecord.getReadAtTimestampMs());

return ingesterTweetEvent;

} catch (TException e) {

LOG.error("Unable to deserialize TweetEventData", e);

deserializationErrorsCount.increment();

}

return null;

}

private void addDebugEventsToIncomingTweet(

IngesterTweetEvent ingesterTweetEvent, long readAtTimestampMs) {

DebugEventUtil.setCreatedAtDebugEvent(

ingesterTweetEvent, ingesterTweetEvent.getFlags().getTimestamp\_ms());

DebugEventUtil.setProcessingStartedAtDebugEvent(ingesterTweetEvent, readAtTimestampMs);

// The TweetEventDeserializerStage takes in a byte[] representation of a tweet, so debug events

// are not automatically appended by TwitterBaseStage. We do that explicitly here.

DebugEventUtil.addDebugEvent(ingesterTweetEvent, getFullStageName(), clock.nowMillis());

}

@VisibleForTesting

static void appendBytesAsHex(StringBuilder sb, byte[] bytes, int maxLength) {

int limit = Math.min(bytes.length, maxLength);

for (int j = 0; j < limit; j++) {

sb.append(HEX\_ARRAY[(bytes[j] >>> 4) & 0x0F]);

sb.append(HEX\_ARRAY[bytes[j] & 0x0F]);

}

}

}