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

import java.util.List;

import java.util.concurrent.CompletableFuture;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.SynchronousQueue;

import java.util.concurrent.ThreadPoolExecutor;

import java.util.concurrent.TimeUnit;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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.twitter.TweetEventDeserializerStage;

import com.twitter.search.ingester.pipeline.twitter.kafka.KafkaConsumerStage;

import com.twitter.search.ingester.pipeline.twitter.kafka.KafkaRawRecordConsumerStage;

import com.twitter.search.ingester.pipeline.util.PipelineV2CreationException;

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

public class RealtimeIngesterPipelineV2 {

private static final Logger LOG = LoggerFactory.getLogger(RealtimeIngesterPipelineV2.class);

private static final String PROD\_ENV = "prod";

private static final String STAGING\_ENV = "staging";

private static final String STAGING1\_ENV = "staging1";

private static final String REALTIME\_CLUSTER = "realtime";

private static final String PROTECTED\_CLUSTER = "protected";

private static final String REALTIME\_CG\_CLUSTER = "realtime\_cg";

private static final String KAFKA\_CLIENT\_ID = "";

private static final String KAFKA\_TOPIC\_NAME = "";

private static final String KAFKA\_CONSUMER\_GROUP\_ID = "";

private static final String KAFKA\_CLUSTER\_PATH = "";

private static final String KAFKA\_DECIDER\_KEY = "ingester\_tweets\_consume\_from\_kafka";

private static final String STATS\_PREFIX = "realtimeingesterpipelinev2";

private SearchCounter kafkaErrorCount = SearchCounter.create(STATS\_PREFIX

+ "\_kafka\_error\_count");

private Boolean running;

private String environment;

private String cluster;

private ExecutorService threadPool;

private KafkaConsumerStage<KafkaRawRecord> kafkaConsumer;

private TweetEventDeserializerStage tweetEventDeserializerStage;

public RealtimeIngesterPipelineV2(String environment, String cluster, int threadsToSpawn) throws

PipelineV2CreationException, PipelineStageException {

if (!environment.equals(PROD\_ENV) && !environment.equals(STAGING\_ENV)

&& !environment.equals(STAGING1\_ENV)) {

throw new PipelineV2CreationException("invalid value for environment");

}

if (!cluster.equals(REALTIME\_CLUSTER)

&& !cluster.equals(PROTECTED\_CLUSTER) && !cluster.equals(REALTIME\_CG\_CLUSTER)) {

throw new PipelineV2CreationException("invalid value for cluster.");

}

int numberOfThreads = Math.max(1, threadsToSpawn);

this.environment = environment;

this.cluster = cluster;

this.threadPool = new ThreadPoolExecutor(numberOfThreads, numberOfThreads, 0L,

TimeUnit.MILLISECONDS, new SynchronousQueue<>(), new ThreadPoolExecutor.CallerRunsPolicy());

initStages();

}

private void initStages() throws PipelineStageException {

kafkaConsumer = new KafkaRawRecordConsumerStage(KAFKA\_CLIENT\_ID, KAFKA\_TOPIC\_NAME,

KAFKA\_CONSUMER\_GROUP\_ID, KAFKA\_CLUSTER\_PATH, KAFKA\_DECIDER\_KEY);

kafkaConsumer.setupStageV2();

tweetEventDeserializerStage = new TweetEventDeserializerStage();

tweetEventDeserializerStage.setupStageV2();

}

/\*\*\*

\* Starts the pipeline by starting the polling from Kafka and passing the events to the first

\* stage of the pipeline.

\*/

public void run() {

running = true;

while (running) {

pollFromKafkaAndSendToPipeline();

}

}

private void pollFromKafkaAndSendToPipeline() {

try {

List<KafkaRawRecord> records = kafkaConsumer.pollFromTopic();

for (KafkaRawRecord record : records) {

processKafkaRecord(record);

}

} catch (PipelineStageException e) {

kafkaErrorCount.increment();

LOG.error("Error polling from Kafka", e);

}

}

private void processKafkaRecord(KafkaRawRecord record) {

CompletableFuture<KafkaRawRecord> stage1 = CompletableFuture.supplyAsync(() -> record,

threadPool);

CompletableFuture<IngesterTweetEvent> stage2 = stage1.thenApplyAsync((KafkaRawRecord r) ->

tweetEventDeserializerStage.runStageV2(r), threadPool);

}

/\*\*\*

\* Stop the pipeline from processing any further events.

\*/

public void shutdown() {

running = false;

kafkaConsumer.cleanupStageV2();

tweetEventDeserializerStage.cleanupStageV2();

}

}