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

import java.time.Duration;

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

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.commons.pipeline.StageException;

import org.apache.kafka.clients.consumer.ConsumerRecords;

import org.apache.kafka.clients.consumer.KafkaConsumer;

import org.apache.kafka.common.TopicPartition;

import org.apache.kafka.common.errors.SaslAuthenticationException;

import org.apache.kafka.common.errors.SerializationException;

import org.apache.kafka.common.serialization.Deserializer;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

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

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

import com.twitter.search.ingester.pipeline.twitter.TwitterBaseStage;

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

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

/\*\*

\* A stage to read Thrift payloads from a Kafka topic.

\*/

public abstract class KafkaConsumerStage<R> extends TwitterBaseStage<Void, R> {

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

private static final String SHUT\_DOWN\_ON\_AUTH\_FAIL = "shut\_down\_on\_authentication\_fail";

private String kafkaClientId;

private String kafkaTopicName;

private String kafkaConsumerGroupId;

private String kafkaClusterPath;

private int maxPollRecords = 1;

private int pollTimeoutMs = 1000;

private boolean partitioned;

private String deciderKey;

private final Deserializer<R> deserializer;

private SearchCounter pollCount;

private SearchCounter deserializationErrorCount;

private SearchRateCounter droppedMessages;

private KafkaConsumer<Long, R> kafkaConsumer;

protected KafkaConsumerStage(String kafkaClientId, String kafkaTopicName,

String kafkaConsumerGroupId, String kafkaClusterPath,

String deciderKey, Deserializer<R> deserializer) {

this.kafkaClientId = kafkaClientId;

this.kafkaTopicName = kafkaTopicName;

this.kafkaConsumerGroupId = kafkaConsumerGroupId;

this.kafkaClusterPath = kafkaClusterPath;

this.deciderKey = deciderKey;

this.deserializer = deserializer;

}

protected KafkaConsumerStage(Deserializer<R> deserializer) {

this.deserializer = deserializer;

}

@Override

protected void initStats() {

super.initStats();

commonInnerSetupStats();

}

private void commonInnerSetupStats() {

pollCount = SearchCounter.export(getStageNamePrefix() + "\_poll\_count");

deserializationErrorCount =

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

droppedMessages =

SearchRateCounter.export(getStageNamePrefix() + "\_dropped\_messages");

}

@Override

protected void innerSetupStats() {

commonInnerSetupStats();

}

@Override

protected void doInnerPreprocess() {

commonInnerSetup();

PipelineUtil.feedStartObjectToStage(this);

}

private void commonInnerSetup() {

Preconditions.checkNotNull(kafkaClientId);

Preconditions.checkNotNull(kafkaClusterPath);

Preconditions.checkNotNull(kafkaTopicName);

kafkaConsumer = wireModule.newKafkaConsumer(

kafkaClusterPath,

deserializer,

kafkaClientId,

kafkaConsumerGroupId,

maxPollRecords);

if (partitioned) {

kafkaConsumer.assign(Collections.singletonList(

new TopicPartition(kafkaTopicName, wireModule.getPartition())));

} else {

kafkaConsumer.subscribe(Collections.singleton(kafkaTopicName));

}

}

@Override

protected void innerSetup() {

commonInnerSetup();

}

@Override

public void innerProcess(Object obj) throws StageException {

StageDriver driver = ((Pipeline) stageContext).getStageDriver(this);

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

pollAndEmit();

}

LOG.info("StageDriver state is no longer RUNNING, closing Kafka consumer.");

closeKafkaConsumer();

}

@VisibleForTesting

void pollAndEmit() throws StageException {

try {

List<R> records = poll();

for (R record : records) {

emitAndCount(record);

}

} catch (PipelineStageException e) {

throw new StageException(this, e);

}

}

/\*\*\*

\* Poll Kafka and get the items from the topic. Record stats.

\* @return

\* @throws PipelineStageException

\*/

public List<R> pollFromTopic() throws PipelineStageException {

long startingTime = startProcessing();

List<R> polledItems = poll();

endProcessing(startingTime);

return polledItems;

}

private List<R> poll() throws PipelineStageException {

List<R> recordsFromKafka = new ArrayList<>();

try {

ConsumerRecords<Long, R> records = kafkaConsumer.poll(Duration.ofMillis(pollTimeoutMs));

pollCount.increment();

records.iterator().forEachRemaining(record -> {

if (deciderKey == null || DeciderUtil.isAvailableForRandomRecipient(decider, deciderKey)) {

recordsFromKafka.add(record.value());

} else {

droppedMessages.increment();

}

});

} catch (SerializationException e) {

deserializationErrorCount.increment();

LOG.error("Failed to deserialize the value.", e);

} catch (SaslAuthenticationException e) {

if (DeciderUtil.isAvailableForRandomRecipient(decider, SHUT\_DOWN\_ON\_AUTH\_FAIL)) {

wireModule.getPipelineExceptionHandler()

.logAndShutdown("Authentication error connecting to Kafka broker: " + e);

} else {

throw new PipelineStageException(this, "Kafka Authentication Error", e);

}

} catch (Exception e) {

throw new PipelineStageException(e);

}

return recordsFromKafka;

}

@VisibleForTesting

void closeKafkaConsumer() {

try {

kafkaConsumer.close();

LOG.info("Kafka kafkaConsumer for {} was closed", getFullStageName());

} catch (Exception e) {

log.error("Failed to close Kafka kafkaConsumer", e);

}

}

@Override

public void release() {

closeKafkaConsumer();

super.release();

}

@Override

public void cleanupStageV2() {

closeKafkaConsumer();

}

@SuppressWarnings("unused") // set from pipeline config

public void setKafkaClientId(String kafkaClientId) {

this.kafkaClientId = kafkaClientId;

}

@SuppressWarnings("unused") // set from pipeline config

public void setKafkaTopicName(String kafkaTopicName) {

this.kafkaTopicName = kafkaTopicName;

}

@SuppressWarnings("unused") // set from pipeline config

public void setKafkaConsumerGroupId(String kafkaConsumerGroupId) {

this.kafkaConsumerGroupId = kafkaConsumerGroupId;

}

@SuppressWarnings("unused") // set from pipeline config

public void setMaxPollRecords(int maxPollRecords) {

this.maxPollRecords = maxPollRecords;

}

@SuppressWarnings("unused") // set from pipeline config

public void setPollTimeoutMs(int pollTimeoutMs) {

this.pollTimeoutMs = pollTimeoutMs;

}

@SuppressWarnings("unused") // set from pipeline config

public void setPartitioned(boolean partitioned) {

this.partitioned = partitioned;

}

@SuppressWarnings("unused") // set from pipeline config

public void setDeciderKey(String deciderKey) {

this.deciderKey = deciderKey;

}

@VisibleForTesting

KafkaConsumer<Long, R> getKafkaConsumer() {

return kafkaConsumer;

}

@SuppressWarnings("unused") // set from pipeline config

public void setKafkaClusterPath(String kafkaClusterPath) {

this.kafkaClusterPath = kafkaClusterPath;

}

}