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

import java.io.File;

import java.net.URL;

import java.util.concurrent.CountDownLatch;

import java.util.concurrent.atomic.AtomicBoolean;

import com.google.common.annotations.VisibleForTesting;

import org.apache.commons.pipeline.Pipeline;

import org.apache.commons.pipeline.PipelineCreationException;

import org.apache.commons.pipeline.StageException;

import org.apache.commons.pipeline.config.DigesterPipelineFactory;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.app.Flag;

import com.twitter.app.Flaggable;

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

import com.twitter.search.ingester.pipeline.wire.ProductionWireModule;

import com.twitter.search.ingester.pipeline.wire.WireModule;

import com.twitter.search.ingester.util.jndi.JndiUtil;

import com.twitter.server.AbstractTwitterServer;

import com.twitter.server.handler.DeciderHandler$;

/\*\* Starts the ingester/indexer pipeline. \*/

public class IngesterPipelineApplication extends AbstractTwitterServer {

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

private static final String VERSION\_2 = "v2";

private final Flag<String> pipelineConfigFile = flag().create(

"config\_file",

"",

"xml file to load pipeline config from. Required.",

Flaggable.ofString());

private final Flag<String> pipelineVersion = flag().create(

"version",

"",

"Specifies if we want to run the acp pipeline or non acp pipeline.",

Flaggable.ofString());

private final Flag<Integer> partitionArg = flag().create(

"shard",

-1,

"The partition this indexer is responsible for.",

Flaggable.ofJavaInteger());

private final Flag<String> deciderOverlay = flag().create(

"decider\_overlay",

"",

"Decider overlay",

Flaggable.ofString());

private final Flag<String> serviceIdentifierFlag = flag().create(

"service\_identifier",

"",

"Service identifier for mutual TLS authentication",

Flaggable.ofString());

private final Flag<String> environment = flag().create(

"environment",

"",

"Specifies the environment the app is running in. Valid values : prod, staging, "

+ "staging1. Required if pipelineVersion == 'v2'",

Flaggable.ofString()

);

private final Flag<String> cluster = flag().create(

"cluster",

"",

"Specifies the cluster the app is running in. Valid values : realtime, protected, "

+ "realtime\_cg, user\_updates. Required if pipelineVersion == 'v2'",

Flaggable.ofString()

);

private final Flag<Float> cores = flag().create(

"cores",

1F,

"Specifies the number of cores this cluster is using. ",

Flaggable.ofJavaFloat()

);

private final CountDownLatch shutdownLatch = new CountDownLatch(1);

public void shutdown() {

shutdownLatch.countDown();

}

private Pipeline pipeline;

private final AtomicBoolean started = new AtomicBoolean(false);

private final AtomicBoolean finished = new AtomicBoolean(false);

/\*\*

\* Boilerplate for the Java-friendly AbstractTwitterServer

\*/

public static class Main {

public static void main(String[] args) {

new IngesterPipelineApplication().main(args);

}

}

/\*\*

\* Code is based on DigesterPipelineFactory.main. We only require reading in one config file.

\*/

@Override

public void main() {

try {

JndiUtil.loadJNDI();

ProductionWireModule wireModule = new ProductionWireModule(

deciderOverlay.get().get(),

partitionArg.getWithDefault().get(),

serviceIdentifierFlag.get());

WireModule.bindWireModule(wireModule);

addAdminRoute(DeciderHandler$.MODULE$.route(

"ingester",

wireModule.getMutableDecisionMaker(),

wireModule.getDecider()));

BuildInfoStats.export();

if (pipelineVersion.get().get().equals(VERSION\_2)) {

runPipelineV2(wireModule);

} else {

runPipelineV1(wireModule);

}

LOG.info("Pipeline terminated. Ingester is DOWN.");

} catch (Exception e) {

LOG.error("Exception in pipeline. Ingester is DOWN.", e);

throw new RuntimeException(e);

}

}

@VisibleForTesting

boolean isFinished() {

return finished.get();

}

@VisibleForTesting

Pipeline createPipeline(URL pipelineConfigFileURL) throws PipelineCreationException {

DigesterPipelineFactory factory = new DigesterPipelineFactory(pipelineConfigFileURL);

LOG.info("Pipeline created from {}, about to begin processing...", pipelineConfigFileURL);

return factory.createPipeline();

}

void runPipelineV1(ProductionWireModule wireModule) throws Exception {

LOG.info("Running Pipeline V1");

final File pipelineFile = new File(pipelineConfigFile.get().get());

URL pipelineConfigFileUrl = pipelineFile.toURI().toURL();

wireModule.setPipelineExceptionHandler(new PipelineExceptionImpl(this));

runPipelineV1(pipelineConfigFileUrl);

shutdownLatch.await();

}

@VisibleForTesting

void runPipelineV1(URL pipelineConfigFileUrl) throws Exception {

pipeline = createPipeline(pipelineConfigFileUrl);

pipeline.start();

started.set(true);

}

void runPipelineV2(ProductionWireModule wireModule) throws Exception {

LOG.info("Running Pipeline V2");

int threadsToSpawn = cores.get().get().intValue() - 1;

RealtimeIngesterPipelineV2 realtimePipeline = new RealtimeIngesterPipelineV2(

environment.get().get(), cluster.get().get(), threadsToSpawn);

wireModule.setPipelineExceptionHandler(new PipelineExceptionImplV2(realtimePipeline));

realtimePipeline.run();

}

@Override

public void onExit() {

try {

LOG.info("Attempting to shutdown gracefully.");

/\*

\* Iterates over each Stage and calls finish(). The Stage is considered finished when

\* its queue is empty. If there is a backup, finish() waits for the queues to empty.

\*/

// We don't call finish() unless the pipeline exists and has started because if any stage

// fails to initialize, no processing is started and not only is calling finish() unnecessary,

// but it will also deadlock any DedicatedThreadStageDriver.

if (pipeline != null && started.get()) {

pipeline.finish();

finished.set(true);

LOG.info("Pipeline exited cleanly.");

} else {

LOG.info("Pipeline not yet started.");

}

} catch (StageException e) {

LOG.error("Unable to shutdown pipeline.", e);

}

}

}