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

import java.util.ArrayList;

import java.util.List;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import javax.annotation.Nullable;

import javax.naming.Context;

import javax.naming.InitialContext;

import javax.naming.NamingException;

import scala.Option;

import scala.collection.JavaConversions$;

import com.google.common.base.Preconditions;

import com.google.common.collect.ImmutableList;

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

import org.apache.kafka.clients.producer.Partitioner;

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

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

import org.apache.thrift.TBase;

import org.apache.thrift.protocol.TBinaryProtocol;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.common.util.Clock;

import com.twitter.common\_internal.text.version.PenguinVersion;

import com.twitter.decider.Decider;

import com.twitter.decider.DeciderFactory;

import com.twitter.decider.DeciderFactory$;

import com.twitter.decider.decisionmaker.DecisionMaker;

import com.twitter.decider.decisionmaker.MutableDecisionMaker;

import com.twitter.eventbus.client.EventBusSubscriber;

import com.twitter.eventbus.client.EventBusSubscriberBuilder;

import com.twitter.finagle.Service;

import com.twitter.finagle.ThriftMux;

import com.twitter.finagle.builder.ClientBuilder;

import com.twitter.finagle.builder.ClientConfig;

import com.twitter.finagle.mtls.authentication.ServiceIdentifier;

import com.twitter.finagle.mtls.client.MtlsThriftMuxClient;

import com.twitter.finagle.mux.transport.OpportunisticTls;

import com.twitter.finagle.service.RetryPolicy;

import com.twitter.finagle.stats.DefaultStatsReceiver;

import com.twitter.finagle.thrift.ClientId;

import com.twitter.finagle.thrift.ThriftClientRequest;

import com.twitter.finatra.kafka.producers.BlockingFinagleKafkaProducer;

import com.twitter.gizmoduck.thriftjava.UserService;

import com.twitter.metastore.client\_v2.MetastoreClient;

import com.twitter.pink\_floyd.thrift.Storer;

import com.twitter.search.common.partitioning.base.PartitionMappingManager;

import com.twitter.search.common.relevance.classifiers.TweetOffensiveEvaluator;

import com.twitter.search.common.schema.earlybird.EarlybirdCluster;

import com.twitter.search.common.util.io.kafka.FinagleKafkaClientUtils;

import com.twitter.search.ingester.pipeline.strato\_fetchers.AudioSpaceCoreFetcher;

import com.twitter.search.ingester.pipeline.strato\_fetchers.AudioSpaceParticipantsFetcher;

import com.twitter.search.ingester.pipeline.strato\_fetchers.NamedEntityFetcher;

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

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

import com.twitter.storage.client.manhattan.kv.JavaManhattanKVEndpoint;

import com.twitter.storage.client.manhattan.kv.ManhattanKVClient;

import com.twitter.storage.client.manhattan.kv.ManhattanKVClientMtlsParams;

import com.twitter.storage.client.manhattan.kv.ManhattanKVEndpointBuilder;

import com.twitter.strato.client.Client;

import com.twitter.strato.client.Strato;

import com.twitter.tweetypie.thriftjava.TweetService;

import com.twitter.util.Duration;

import com.twitter.util.Function;

import com.twitter.util.Future;

/\*\*

\* The injection module that provides all production bindings.

\*/

public class ProductionWireModule extends WireModule {

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

private static final String DECIDER\_BASE = "config/ingester-indexer-decider.yml";

private static final String GEOCODE\_APP\_ID = "search\_ingester\_readonly";

private static final String CLUSTER\_DEST\_NAME = "";

private static final String JNDI\_GIZMODUCK\_DEST = JNDI\_PIPELINE\_ROOT + "gizmoduckDest";

private static final String PENGUIN\_VERSIONS\_JNDI\_NAME = JNDI\_PIPELINE\_ROOT + "penguinVersions";

private static final String SEGMENT\_BUFFER\_SIZE\_JNDI\_NAME =

JNDI\_PIPELINE\_ROOT + "segmentBufferSize";

private static final String SEGMENT\_SEAL\_DELAY\_TIME\_MS\_JNDI\_NAME =

JNDI\_PIPELINE\_ROOT + "segmentSealDelayTimeMs";

private static final String JNDI\_DL\_URI = JNDI\_PIPELINE\_ROOT + "distributedlog/dlUri";

private static final String JNDI\_DL\_CONFIG\_FILE =

JNDI\_PIPELINE\_ROOT + "distributedlog/configFile";

private static final String CLUSTER\_JNDI\_NAME = JNDI\_PIPELINE\_ROOT + "cluster";

private static final String TIME\_SLICE\_MANAGER\_ROOT\_PATH = "";

private static final String MAX\_TIMESLICES\_JNDI\_NAME =

TIME\_SLICE\_MANAGER\_ROOT\_PATH + "hashPartition/maxTimeSlices";

private static final String MAX\_SEGMENT\_SIZE\_JNDI\_NAME =

TIME\_SLICE\_MANAGER\_ROOT\_PATH + "hashPartition/maxSegmentSize";

private static final String NUM\_PARTITIONS\_JNDI\_NAME =

TIME\_SLICE\_MANAGER\_ROOT\_PATH + "hashPartition/numPartitions";

private static final String PINK\_CLIENT\_ID = "search\_ingester";

private final Decider decider;

private final MutableDecisionMaker mutableDecisionMaker;

private final int partition;

private PipelineExceptionHandler pipelineExceptionHandler;

private final StratoMetaStoreWireModule stratoMetaStoreWireModule;

private final Client stratoClient;

private ServiceIdentifier serviceIdentifier = ServiceIdentifier.empty();

private List<PenguinVersion> penguinVersions;

public ProductionWireModule(String deciderOverlay, int partition, Option<String>

serviceIdentifierFlag) {

mutableDecisionMaker = new MutableDecisionMaker();

decider = DeciderFactory.get()

.withBaseConfig(DECIDER\_BASE)

.withOverlayConfig(deciderOverlay)

.withRefreshBase(false)

.withDecisionMakers(

ImmutableList.<DecisionMaker>builder()

.add(mutableDecisionMaker)

.addAll(JavaConversions$.MODULE$.asJavaCollection(

DeciderFactory$.MODULE$.DefaultDecisionMakers()))

.build())

.apply();

this.partition = partition;

this.stratoMetaStoreWireModule = new StratoMetaStoreWireModule(this);

if (serviceIdentifierFlag.isDefined()) {

this.serviceIdentifier =

ServiceIdentifier.flagOfServiceIdentifier().parse(serviceIdentifierFlag.get());

}

this.stratoClient = Strato.client()

.withMutualTls(serviceIdentifier)

.withRequestTimeout(Duration.fromMilliseconds(500))

.build();

}

public ProductionWireModule(String deciderOverlay,

int partition,

PipelineExceptionHandler pipelineExceptionHandler,

Option<String> serviceIdentifierFlag) {

this(deciderOverlay, partition, serviceIdentifierFlag);

this.pipelineExceptionHandler = pipelineExceptionHandler;

}

public void setPipelineExceptionHandler(PipelineExceptionHandler pipelineExceptionHandler) {

this.pipelineExceptionHandler = pipelineExceptionHandler;

}

@Override

public ServiceIdentifier getServiceIdentifier() {

return serviceIdentifier;

}

@Override

public PartitionMappingManager getPartitionMappingManager() {

return PartitionMappingManager.getInstance();

}

@Override

public JavaManhattanKVEndpoint getJavaManhattanKVEndpoint() {

Preconditions.checkNotNull(serviceIdentifier,

"Can't create Manhattan client with S2S authentication because Service Identifier is null");

LOG.info(String.format("Service identifier for Manhattan client: %s",

ServiceIdentifier.asString(serviceIdentifier)));

ManhattanKVClientMtlsParams mtlsParams = ManhattanKVClientMtlsParams.apply(serviceIdentifier,

ManhattanKVClientMtlsParams.apply$default$2(),

OpportunisticTls.Required()

);

return ManhattanKVEndpointBuilder

.apply(ManhattanKVClient.apply(GEOCODE\_APP\_ID, CLUSTER\_DEST\_NAME, mtlsParams))

.buildJava();

}

@Override

public Decider getDecider() {

return decider;

}

// Since MutableDecisionMaker is needed only for production TwitterServer, this method is defined

// only in ProductionWireModule.

public MutableDecisionMaker getMutableDecisionMaker() {

return mutableDecisionMaker;

}

@Override

public int getPartition() {

return partition;

}

@Override

public PipelineExceptionHandler getPipelineExceptionHandler() {

return pipelineExceptionHandler;

}

@Override

public Storer.ServiceIface getStorer(Duration requestTimeout, int retries) {

TBinaryProtocol.Factory factory = new TBinaryProtocol.Factory();

MtlsThriftMuxClient mtlsThriftMuxClient = new MtlsThriftMuxClient(

ThriftMux.client().withClientId(new ClientId(PINK\_CLIENT\_ID)));

ThriftMux.Client tmuxClient = mtlsThriftMuxClient

.withMutualTls(serviceIdentifier)

.withOpportunisticTls(OpportunisticTls.Required());

ClientBuilder<

ThriftClientRequest,

byte[],

ClientConfig.Yes,

ClientConfig.Yes,

ClientConfig.Yes> builder = ClientBuilder.get()

.dest("")

.requestTimeout(requestTimeout)

.retries(retries)

.timeout(requestTimeout.mul(retries))

.stack(tmuxClient)

.name("pinkclient")

.reportTo(DefaultStatsReceiver.get());

return new Storer.ServiceToClient(ClientBuilder.safeBuild(builder), factory);

}

@Override

public MetastoreClient getMetastoreClient() throws NamingException {

return stratoMetaStoreWireModule.getMetastoreClient(this.serviceIdentifier);

}

@Override

public ExecutorService getThreadPool(int numThreads) {

return Executors.newFixedThreadPool(numThreads);

}

@Override

public TweetService.ServiceToClient getTweetyPieClient(String tweetypieClientId)

throws NamingException {

return TweetyPieWireModule.getTweetyPieClient(tweetypieClientId, serviceIdentifier);

}

@Override

public UserService.ServiceToClient getGizmoduckClient(String clientId)

throws NamingException {

Context context = new InitialContext();

String dest = (String) context.lookup(JNDI\_GIZMODUCK\_DEST);

MtlsThriftMuxClient mtlsThriftMuxClient = new MtlsThriftMuxClient(

ThriftMux.client().withClientId(new ClientId(clientId)));

Service<ThriftClientRequest, byte[]> clientBuilder =

ClientBuilder.safeBuild(

ClientBuilder

.get()

.requestTimeout(Duration.fromMilliseconds(800))

.retryPolicy(RetryPolicy.tries(3))

.name("search\_ingester\_gizmoduck\_client")

.reportTo(DefaultStatsReceiver.get())

.daemon(true)

.dest(dest)

.stack(mtlsThriftMuxClient.withMutualTls(serviceIdentifier)

.withOpportunisticTls(OpportunisticTls.Required())));

return new UserService.ServiceToClient(clientBuilder, new TBinaryProtocol.Factory());

}

@Override

public <T extends TBase<?, ?>> EventBusSubscriber<T> createEventBusSubscriber(

Function<T, Future<?>> process,

Class<T> thriftStructClass,

String eventBusSubscriberId,

int maxConcurrentEvents) {

Preconditions.checkNotNull(serviceIdentifier,

"Can't create EventBusSubscriber with S2S auth because Service Identifier is null");

LOG.info(String.format("Service identifier for EventBusSubscriber Manhattan client: %s",

ServiceIdentifier.asString(serviceIdentifier)));

// We set the processTimeoutMs parameter here to be Duration.Top because we do not want to read

// more events from EventBus if we are experiencing back pressure and cannot write them to the

// downstream queue.

return EventBusSubscriberBuilder.apply()

.subscriberId(eventBusSubscriberId)

.skipToLatest(false)

.fromAllZones(true)

.statsReceiver(DefaultStatsReceiver.get().scope("eventbus"))

.thriftStruct(thriftStructClass)

.serviceIdentifier(serviceIdentifier)

.maxConcurrentEvents(maxConcurrentEvents)

.processTimeout(Duration.Top())

.build(process);

}

@Override

public Clock getClock() {

return Clock.SYSTEM\_CLOCK;

}

@Override

public TweetOffensiveEvaluator getTweetOffensiveEvaluator() {

return new TweetOffensiveEvaluator();

}

@Override

public EarlybirdCluster getEarlybirdCluster() throws NamingException {

Context jndiContext = new InitialContext();

String clusterName = (String) jndiContext.lookup(CLUSTER\_JNDI\_NAME);

return EarlybirdCluster.valueOf(clusterName.toUpperCase());

}

@Override

public List<PenguinVersion> getPenguinVersions() throws NamingException {

Context context = new InitialContext();

String penguinVersionsStr = (String) context.lookup(PENGUIN\_VERSIONS\_JNDI\_NAME);

penguinVersions = new ArrayList<>();

for (String penguinVersion : penguinVersionsStr.split(",")) {

PenguinVersion pv = PenguinVersion.versionFromByteValue(Byte.parseByte(penguinVersion));

if (PenguinVersionsUtil.isPenguinVersionAvailable(pv, decider)) {

penguinVersions.add(pv);

}

}

Preconditions.checkArgument(penguinVersions.size() > 0,

"At least one penguin version must be specified.");

return penguinVersions;

}

// We update penguin versions via deciders in order to disable one in case of an emergency.

@Override

public List<PenguinVersion> getCurrentlyEnabledPenguinVersions() {

return PenguinVersionsUtil.filterPenguinVersionsWithDeciders(penguinVersions, decider);

}

@Override

public NamedEntityFetcher getNamedEntityFetcher() {

return new NamedEntityFetcher(stratoClient);

}

@Override

public AudioSpaceParticipantsFetcher getAudioSpaceParticipantsFetcher() {

return new AudioSpaceParticipantsFetcher(stratoClient);

}

@Override

public AudioSpaceCoreFetcher getAudioSpaceCoreFetcher() {

return new AudioSpaceCoreFetcher(stratoClient);

}

@Override

public <T> KafkaConsumer<Long, T> newKafkaConsumer(

String kafkaClusterPath, Deserializer<T> deserializer, String clientId, String groupId,

int maxPollRecords) {

return FinagleKafkaClientUtils.newKafkaConsumer(

kafkaClusterPath, deserializer, clientId, groupId, maxPollRecords);

}

@Override

public <T> BlockingFinagleKafkaProducer<Long, T> newFinagleKafkaProducer(

String kafkaClusterPath, Serializer<T> serializer, String clientId,

@Nullable Class<? extends Partitioner> partitionerClass) {

return FinagleKafkaClientUtils.newFinagleKafkaProducer(

kafkaClusterPath, true, serializer, clientId, partitionerClass);

}

}