package com.twitter.search.earlybird.factory;

import java.io.IOException;

import java.lang.management.ManagementFactory;

import java.util.Optional;

import java.util.concurrent.ScheduledThreadPoolExecutor;

import java.util.concurrent.TimeUnit;

import com.google.common.base.Preconditions;

import com.google.common.collect.Lists;

import com.sun.management.OperatingSystemMXBean;

import org.apache.directory.api.util.Strings;

import org.apache.hadoop.fs.FileSystem;

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

import org.apache.kafka.common.TopicPartition;

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.finagle.stats.MetricsStatsReceiver;

import com.twitter.finagle.stats.StatsReceiver;

import com.twitter.search.common.aurora.AuroraSchedulerClient;

import com.twitter.search.common.concurrent.ScheduledExecutorServiceFactory;

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

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

import com.twitter.search.common.file.FileUtils;

import com.twitter.search.common.indexing.thriftjava.ThriftVersionedEvents;

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

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

import com.twitter.search.common.partitioning.zookeeper.SearchZkClient;

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

import com.twitter.search.common.schema.earlybird.EarlybirdFieldConstants.EarlybirdFieldConstant;

import com.twitter.search.common.search.termination.QueryTimeoutFactory;

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

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

import com.twitter.search.common.util.ml.tensorflow\_engine.TensorflowModelsManager;

import com.twitter.search.common.util.zktrylock.ZooKeeperTryLockFactory;

import com.twitter.search.common.util.zookeeper.ZooKeeperProxy;

import com.twitter.search.earlybird.EarlybirdCPUQualityFactor;

import com.twitter.search.earlybird.EarlybirdDarkProxy;

import com.twitter.search.earlybird.EarlybirdFinagleServerManager;

import com.twitter.search.earlybird.EarlybirdFuturePoolManager;

import com.twitter.search.earlybird.EarlybirdIndexConfig;

import com.twitter.search.earlybird.EarlybirdProductionFinagleServerManager;

import com.twitter.search.earlybird.EarlybirdServerSetManager;

import com.twitter.search.earlybird.EarlybirdWarmUpManager;

import com.twitter.search.earlybird.QualityFactor;

import com.twitter.search.earlybird.ServerSetMember;

import com.twitter.search.earlybird.UpdateableEarlybirdStateManager;

import com.twitter.search.earlybird.archive.ArchiveEarlybirdIndexConfig;

import com.twitter.search.earlybird.archive.ArchiveSearchPartitionManager;

import com.twitter.search.earlybird.common.CaughtUpMonitor;

import com.twitter.search.earlybird.common.config.EarlybirdConfig;

import com.twitter.search.earlybird.common.config.EarlybirdProperty;

import com.twitter.search.earlybird.common.userupdates.UserScrubGeoMap;

import com.twitter.search.earlybird.common.userupdates.UserUpdatesChecker;

import com.twitter.search.earlybird.common.userupdates.UserTable;

import com.twitter.search.earlybird.exception.MissingKafkaTopicException;

import com.twitter.search.earlybird.exception.CriticalExceptionHandler;

import com.twitter.search.earlybird.index.EarlybirdSegmentFactory;

import com.twitter.search.earlybird.ml.ScoringModelsManager;

import com.twitter.search.earlybird.partition.AudioSpaceEventsStreamIndexer;

import com.twitter.search.earlybird.partition.AudioSpaceTable;

import com.twitter.search.earlybird.partition.DynamicPartitionConfig;

import com.twitter.search.earlybird.partition.EarlybirdIndexFlusher;

import com.twitter.search.earlybird.partition.EarlybirdIndexLoader;

import com.twitter.search.earlybird.partition.EarlybirdKafkaConsumer;

import com.twitter.search.earlybird.partition.EarlybirdStartup;

import com.twitter.search.earlybird.partition.OptimizationAndFlushingCoordinationLock;

import com.twitter.search.earlybird.partition.TimeLimitedHadoopExistsCall;

import com.twitter.search.earlybird.partition.UserScrubGeoEventStreamIndexer;

import com.twitter.search.earlybird.partition.freshstartup.FreshStartupHandler;

import com.twitter.search.earlybird.partition.HdfsUtil;

import com.twitter.search.earlybird.partition.KafkaStartup;

import com.twitter.search.earlybird.partition.MultiSegmentTermDictionaryManager;

import com.twitter.search.earlybird.partition.PartitionManager;

import com.twitter.search.earlybird.partition.PartitionManagerStartup;

import com.twitter.search.earlybird.partition.PartitionWriter;

import com.twitter.search.earlybird.partition.SearchIndexingMetricSet;

import com.twitter.search.earlybird.partition.SegmentManager;

import com.twitter.search.earlybird.partition.SegmentSyncConfig;

import com.twitter.search.earlybird.partition.StartupUserEventIndexer;

import com.twitter.search.earlybird.partition.TweetCreateHandler;

import com.twitter.search.earlybird.partition.TweetUpdateHandler;

import com.twitter.search.earlybird.partition.UserUpdatesStreamIndexer;

import com.twitter.search.earlybird.querycache.QueryCacheConfig;

import com.twitter.search.earlybird.querycache.QueryCacheManager;

import com.twitter.search.earlybird.stats.EarlybirdSearcherStats;

import com.twitter.search.earlybird.util.CoordinatedEarlybirdAction;

import com.twitter.search.earlybird.util.EarlybirdDecider;

import com.twitter.search.earlybird.util.TermCountMonitor;

import com.twitter.search.earlybird.util.TweetCountMonitor;

import com.twitter.ubs.thriftjava.AudioSpaceBaseEvent;

/\*\*

\* Production module that provides Earlybird components.

\*/

public class EarlybirdWireModule {

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

private static final int MAX\_POLL\_RECORDS = 1000;

/\*\*

\* How many threads we will use for building up the query cache during startup.

\* The number of threads will be set to 1 after this earlybird is current.

\*/

private static final int QUERY\_CACHE\_NUM\_WORKER\_THREADS\_AT\_STARTUP =

EarlybirdConfig.getInt("query\_cache\_updater\_startup\_threads", 1);

/\*\*

\* Scheduled executor service factory can be re-used in production.

\* All the managers can share the same executor service factory.

\*/

private final ScheduledExecutorServiceFactory sharedExecutorServiceFactory =

new ScheduledExecutorServiceFactory();

private final SearchStatsReceiver sharedSearchStatsReceiver = new SearchStatsReceiverImpl();

private final StatsReceiver sharedFinagleStatsReceiver = new MetricsStatsReceiver();

private final SearchIndexingMetricSet searchIndexingMetricSet =

new SearchIndexingMetricSet(sharedSearchStatsReceiver);

private final EarlybirdSearcherStats tweetsSearcherStats =

new EarlybirdSearcherStats(sharedSearchStatsReceiver);

private final CaughtUpMonitor indexCaughtUpMonitor = new CaughtUpMonitor("dl\_index");

public CaughtUpMonitor provideIndexCaughtUpMonitor() {

return indexCaughtUpMonitor;

}

private final CaughtUpMonitor kafkaIndexCaughtUpMonitor = new CaughtUpMonitor("kafka\_index");

public CaughtUpMonitor provideKafkaIndexCaughtUpMonitor() {

return kafkaIndexCaughtUpMonitor;

}

private final OptimizationAndFlushingCoordinationLock optimizationAndFlushingCoordinationLock =

new OptimizationAndFlushingCoordinationLock();

public OptimizationAndFlushingCoordinationLock provideOptimizationAndFlushingCoordinationLock() {

return optimizationAndFlushingCoordinationLock;

}

public QueryTimeoutFactory provideQueryTimeoutFactory() {

return new QueryTimeoutFactory();

}

public static class ZooKeeperClients {

public ZooKeeperProxy discoveryClient;

public ZooKeeperProxy stateClient;

public ZooKeeperClients() {

this(

SearchZkClient.getServiceDiscoveryZooKeeperClient(),

SearchZkClient.getSZooKeeperClient());

}

public ZooKeeperClients(ZooKeeperProxy discoveryClient, ZooKeeperProxy stateClient) {

this.discoveryClient = discoveryClient;

this.stateClient = stateClient;

}

}

/\*\*

\* Provides the earlybird decider.

\*/

public Decider provideDecider() {

return EarlybirdDecider.initialize();

}

/\*\*

\* Provides the set of ZooKeeper clients to be used by earlybird.

\*/

public ZooKeeperClients provideZooKeeperClients() {

return new ZooKeeperClients();

}

/\*\*

\* Provides the query cache config.

\*/

public QueryCacheConfig provideQueryCacheConfig(SearchStatsReceiver searchStatsReceiver) {

return new QueryCacheConfig(searchStatsReceiver);

}

/\*\*

\* Provides the earlybird index config.

\*/

public EarlybirdIndexConfig provideEarlybirdIndexConfig(

Decider decider, SearchIndexingMetricSet indexingMetricSet,

CriticalExceptionHandler criticalExceptionHandler) {

return EarlybirdIndexConfigUtil.createEarlybirdIndexConfig(decider, indexingMetricSet,

criticalExceptionHandler);

}

public DynamicPartitionConfig provideDynamicPartitionConfig() {

return new DynamicPartitionConfig(PartitionConfigUtil.initPartitionConfig());

}

/\*\*

\* Provides the segment manager to be used by this earlybird.

\*/

public SegmentManager provideSegmentManager(

DynamicPartitionConfig dynamicPartitionConfig,

EarlybirdIndexConfig earlybirdIndexConfig,

SearchIndexingMetricSet partitionIndexingMetricSet,

EarlybirdSearcherStats searcherStats,

SearchStatsReceiver earlybirdServerStats,

UserUpdatesChecker userUpdatesChecker,

SegmentSyncConfig segmentSyncConfig,

UserTable userTable,

UserScrubGeoMap userScrubGeoMap,

Clock clock,

CriticalExceptionHandler criticalExceptionHandler) {

return new SegmentManager(

dynamicPartitionConfig,

earlybirdIndexConfig,

partitionIndexingMetricSet,

searcherStats,

earlybirdServerStats,

userUpdatesChecker,

segmentSyncConfig,

userTable,

userScrubGeoMap,

clock,

EarlybirdConfig.getMaxSegmentSize(),

criticalExceptionHandler,

provideKafkaIndexCaughtUpMonitor());

}

public QueryCacheManager provideQueryCacheManager(

QueryCacheConfig config,

EarlybirdIndexConfig indexConfig,

int maxEnabledSegments,

UserTable userTable,

UserScrubGeoMap userScrubGeoMap,

ScheduledExecutorServiceFactory queryCacheUpdaterScheduledExecutorFactory,

SearchStatsReceiver searchStatsReceiver,

EarlybirdSearcherStats searcherStats,

Decider decider,

CriticalExceptionHandler criticalExceptionHandler,

Clock clock) {

return new QueryCacheManager(config, indexConfig, maxEnabledSegments, userTable,

userScrubGeoMap, queryCacheUpdaterScheduledExecutorFactory, searchStatsReceiver,

searcherStats, decider, criticalExceptionHandler, clock);

}

public TermCountMonitor provideTermCountMonitor(

SegmentManager segmentManager, ScheduledExecutorServiceFactory executorServiceFactory,

SearchStatsReceiver searchStatsReceiver,

CriticalExceptionHandler criticalExceptionHandler) {

return new TermCountMonitor(segmentManager, executorServiceFactory, 500, TimeUnit.MILLISECONDS,

searchStatsReceiver, criticalExceptionHandler);

}

public TweetCountMonitor provideTweetCountMonitor(

SegmentManager segmentManager,

ScheduledExecutorServiceFactory executorServiceFactory,

SearchStatsReceiver searchStatsReceiver,

CriticalExceptionHandler criticalExceptionHandler) {

return new TweetCountMonitor(segmentManager, executorServiceFactory, 500,

TimeUnit.MILLISECONDS, searchStatsReceiver, criticalExceptionHandler);

}

/\*\*

\* Returns a manager that keeps track of earlybird's global state while it runs.

\*/

public UpdateableEarlybirdStateManager provideUpdateableEarlybirdStateManager(

EarlybirdIndexConfig earlybirdIndexConfig,

DynamicPartitionConfig dynamicPartitionConfig,

ZooKeeperProxy zooKeeperClient,

AuroraSchedulerClient schedulerClient,

ScheduledExecutorServiceFactory executorServiceFactory,

ScoringModelsManager scoringModelsManager,

TensorflowModelsManager tensorflowModelsManager,

SearchStatsReceiver searchStatsReceiver,

SearchDecider searchDecider,

CriticalExceptionHandler criticalExceptionHandler) {

Clock clock = provideClockForStateManager();

return new UpdateableEarlybirdStateManager(

earlybirdIndexConfig, dynamicPartitionConfig, zooKeeperClient, schedulerClient,

executorServiceFactory, scoringModelsManager, tensorflowModelsManager, searchStatsReceiver,

searchDecider, criticalExceptionHandler,

clock);

}

public Clock provideClockForStateManager() {

return this.provideClock();

}

public ScheduledExecutorServiceFactory providePartitionManagerExecutorFactory() {

return sharedExecutorServiceFactory;

}

public ScheduledExecutorServiceFactory provideStateUpdateManagerExecutorFactory() {

return sharedExecutorServiceFactory;

}

public ScheduledExecutorServiceFactory provideTermCountMonitorScheduledExecutorFactory() {

return sharedExecutorServiceFactory;

}

public ScheduledExecutorServiceFactory provideTweetCountMonitorScheduledExecutorFactory() {

return sharedExecutorServiceFactory;

}

/\*\*

\* Provides the ScheduledExecutorServiceFactory that will be used to schedule all query cache

\* update tasks.

\*/

public ScheduledExecutorServiceFactory provideQueryCacheUpdateTaskScheduledExecutorFactory() {

return new ScheduledExecutorServiceFactory() {

@Override

public QueryCacheUpdaterScheduledExecutorService<ScheduledThreadPoolExecutor> build(

String threadNameFormat, boolean isDaemon) {

ScheduledThreadPoolExecutor threadpoolExecutor =

new ScheduledThreadPoolExecutor(QUERY\_CACHE\_NUM\_WORKER\_THREADS\_AT\_STARTUP,

buildThreadFactory(threadNameFormat, isDaemon));

threadpoolExecutor.setMaximumPoolSize(QUERY\_CACHE\_NUM\_WORKER\_THREADS\_AT\_STARTUP);

threadpoolExecutor.setCorePoolSize(QUERY\_CACHE\_NUM\_WORKER\_THREADS\_AT\_STARTUP);

threadpoolExecutor.setExecuteExistingDelayedTasksAfterShutdownPolicy(false);

threadpoolExecutor.setContinueExistingPeriodicTasksAfterShutdownPolicy(false);

threadpoolExecutor.setRemoveOnCancelPolicy(true);

LOG.info("Starting query cache executor with {} thread.",

QUERY\_CACHE\_NUM\_WORKER\_THREADS\_AT\_STARTUP);

return new QueryCacheUpdaterScheduledExecutorService<ScheduledThreadPoolExecutor>(

threadpoolExecutor) {

@Override public void setWorkerPoolSizeAfterStartup() {

delegate.setCorePoolSize(1);

delegate.setMaximumPoolSize(1);

LOG.info("Reset query cache executor to be single threaded.");

}

};

}

};

}

public ScheduledExecutorServiceFactory provideSimpleUserUpdateIndexerScheduledExecutorFactory() {

return sharedExecutorServiceFactory;

}

/\*\*

\* Returns the manager that manages the pool of searcher threads.

\*/

public EarlybirdFuturePoolManager provideFuturePoolManager() {

return new EarlybirdFuturePoolManager("SearcherWorker");

}

/\*\*

\* Returns the manager that manages all earlybird finagle servers (warm up and production).

\*/

public EarlybirdFinagleServerManager provideFinagleServerManager(

CriticalExceptionHandler criticalExceptionHandler) {

return new EarlybirdProductionFinagleServerManager(criticalExceptionHandler);

}

/\*\*

\* Creates the production serverset manager.

\*/

public EarlybirdServerSetManager provideServerSetManager(

ZooKeeperProxy discoveryClient,

DynamicPartitionConfig dynamicPartitionConfig,

SearchStatsReceiver searchStatsReceiver,

int port,

String serverSetNamePrefix) {

return new EarlybirdServerSetManager(

searchStatsReceiver,

discoveryClient,

dynamicPartitionConfig.getCurrentPartitionConfig(),

port,

serverSetNamePrefix);

}

/\*\*

\* Creates the warm up serverset manager.

\*/

public EarlybirdWarmUpManager provideWarmUpManager(

ZooKeeperProxy discoveryClient,

DynamicPartitionConfig dynamicPartitionConfig,

SearchStatsReceiver searchStatsReceiver,

Decider decider,

Clock clock,

int port,

String serverSetNamePrefix) {

return new EarlybirdWarmUpManager(

new EarlybirdServerSetManager(

searchStatsReceiver,

discoveryClient,

dynamicPartitionConfig.getCurrentPartitionConfig(),

port,

serverSetNamePrefix),

dynamicPartitionConfig.getCurrentPartitionConfig(),

searchIndexingMetricSet,

decider,

clock);

}

/\*\*

\* Returns a dark proxy that knows how to send dark traffic to the warm up earlybird serverset.

\*/

public EarlybirdDarkProxy provideEarlybirdDarkProxy(

SearchDecider searchDecider,

StatsReceiver finagleStatsReceiver,

EarlybirdServerSetManager earlybirdServerSetManager,

EarlybirdWarmUpManager earlybirdWarmUpManager,

String clusterName) {

return new EarlybirdDarkProxy(searchDecider,

finagleStatsReceiver.scope("dark\_proxy"),

earlybirdServerSetManager,

earlybirdWarmUpManager,

clusterName);

}

/\*\*

\* Returns the manager for all (non-Tensorflow) scoring models.

\*/

public ScoringModelsManager provideScoringModelsManager(

SearchStatsReceiver serverStats,

EarlybirdIndexConfig earlybirdIndexConfig) {

boolean modelsEnabled = EarlybirdConfig.getBool("scoring\_models\_enabled", false);

if (!modelsEnabled) {

LOG.info("Scoring Models - Disabled in the config. Not loading any models.");

serverStats.getCounter("scoring\_models\_disabled\_in\_config").increment();

return ScoringModelsManager.NO\_OP\_MANAGER;

}

String hdfsNameNode = EarlybirdConfig.getString("scoring\_models\_namenode");

String hdfsModelsPath = EarlybirdConfig.getString("scoring\_models\_basedir");

try {

return ScoringModelsManager.create(

serverStats, hdfsNameNode, hdfsModelsPath, earlybirdIndexConfig.getSchema());

} catch (IOException e) {

LOG.error("Scoring Models - Error creating ScoringModelsManager", e);

serverStats.getCounter("scoring\_models\_initialization\_errors").increment();

return ScoringModelsManager.NO\_OP\_MANAGER;

}

}

/\*\*

\* Provides the manager for all Tensorflow models.

\*/

public TensorflowModelsManager provideTensorflowModelsManager(

SearchStatsReceiver serverStats,

String statsPrefix,

Decider decider,

EarlybirdIndexConfig earlybirdIndexConfig) {

boolean modelsEnabled = EarlybirdProperty.TF\_MODELS\_ENABLED.get(false);

if (!modelsEnabled) {

LOG.info("Tensorflow Models - Disabled in the config. Not loading any models.");

serverStats.getCounter("tf\_models\_disabled\_in\_config").increment();

return TensorflowModelsManager.createNoOp(statsPrefix);

}

String modelsConfigPath =

Preconditions.checkNotNull(EarlybirdProperty.TF\_MODELS\_CONFIG\_PATH.get());

int intraOpThreads = Preconditions.checkNotNull(EarlybirdProperty.TF\_INTRA\_OP\_THREADS.get(0));

int interOpThreads = Preconditions.checkNotNull(EarlybirdProperty.TF\_INTER\_OP\_THREADS.get(0));

TensorflowModelsManager.initTensorflowThreadPools(intraOpThreads, interOpThreads);

return TensorflowModelsManager.createUsingConfigFile(

FileUtils.getFileHandle(modelsConfigPath),

true,

statsPrefix,

() -> DeciderUtil.isAvailableForRandomRecipient(

decider, "enable\_tf\_serve\_models"),

() -> decider.isAvailable("enable\_tf\_load\_models"),

earlybirdIndexConfig.getSchema());

}

public SearchStatsReceiver provideEarlybirdServerStatsReceiver() {

return sharedSearchStatsReceiver;

}

public StatsReceiver provideFinagleStatsReceiver() {

return sharedFinagleStatsReceiver;

}

public SearchIndexingMetricSet provideSearchIndexingMetricSet() {

return searchIndexingMetricSet;

}

public EarlybirdSearcherStats provideTweetsSearcherStats() {

return tweetsSearcherStats;

}

/\*\*

\* Provides the clock to be used by this earlybird.

\*/

public Clock provideClock() {

return Clock.SYSTEM\_CLOCK;

}

/\*\*

\* Provides the config for the multi-segment term dictionary manager.

\*/

public MultiSegmentTermDictionaryManager.Config provideMultiSegmentTermDictionaryManagerConfig() {

return new MultiSegmentTermDictionaryManager.Config(

Lists.newArrayList(

EarlybirdFieldConstant.FROM\_USER\_ID\_FIELD.getFieldName()));

}

/\*\*

\* Provides the manager for the term dictionary that spans all segments.

\*/

public MultiSegmentTermDictionaryManager provideMultiSegmentTermDictionaryManager(

MultiSegmentTermDictionaryManager.Config termDictionaryConfig,

SegmentManager segmentManager,

SearchStatsReceiver statsReceiver,

Decider decider,

EarlybirdCluster earlybirdCluster) {

return new MultiSegmentTermDictionaryManager(

termDictionaryConfig, segmentManager, statsReceiver, decider, earlybirdCluster);

}

/\*\*

\* Returns the partition manager to be used by the archive earlybirds.

\*/

public PartitionManager provideFullArchivePartitionManager(

ZooKeeperTryLockFactory zooKeeperTryLockFactory,

QueryCacheManager queryCacheManager,

SegmentManager segmentManager,

DynamicPartitionConfig dynamicPartitionConfig,

UserUpdatesStreamIndexer userUpdatesStreamIndexer,

UserScrubGeoEventStreamIndexer userScrubGeoEventStreamIndexer,

SearchStatsReceiver searchStatsReceiver,

ArchiveEarlybirdIndexConfig earlybirdIndexConfig,

ServerSetMember serverSetMember,

ScheduledExecutorServiceFactory executorServiceFactory,

ScheduledExecutorServiceFactory userUpdateIndexerExecutorFactory,

SearchIndexingMetricSet earlybirdSearchIndexingMetricSet,

Clock clock,

SegmentSyncConfig segmentSyncConfig,

CriticalExceptionHandler criticalExceptionHandler) throws IOException {

return new ArchiveSearchPartitionManager(

zooKeeperTryLockFactory,

queryCacheManager,

segmentManager,

dynamicPartitionConfig,

userUpdatesStreamIndexer,

userScrubGeoEventStreamIndexer,

searchStatsReceiver,

earlybirdIndexConfig,

serverSetMember,

executorServiceFactory,

userUpdateIndexerExecutorFactory,

earlybirdSearchIndexingMetricSet,

segmentSyncConfig,

clock,

criticalExceptionHandler);

}

/\*\*

\* Provides the SegmentSyncConfig instance to be used by earlybird.

\*/

public SegmentSyncConfig provideSegmentSyncConfig(EarlybirdCluster cluster) {

String scrubGen = null;

if (cluster == EarlybirdCluster.FULL\_ARCHIVE) {

scrubGen = EarlybirdProperty.EARLYBIRD\_SCRUB\_GEN.get();

LOG.info("The scrubGen provided from Aurora is: {}", scrubGen);

Preconditions.checkState(Strings.isNotEmpty(scrubGen));

}

return new SegmentSyncConfig(Optional.ofNullable(scrubGen));

}

protected void storeEarlybirdStartupProducts(

TweetCreateHandler tweetCreateHandler,

PartitionWriter partitionWriter,

EarlybirdIndexFlusher earlybirdIndexFlusher

) {

// TestWireModule wants to store these for further use.

}

/\*\*

\* What directory are we going to load segments from on startup.

\*

\* When you're running loadtests or stagingN instances and they don't have a recent index

\* flushed, it can take hours to generate a new index with a fresh startup. This slows

\* down development. If the read\_index\_from\_prod\_location flag is set to true, we will read

\* the index from the location where prod instances are flushing their index to.

\* Unset it if you want to generate your own index.

\*

\* @return a string with the directory.

\*/

public String getIndexLoadingDirectory() {

boolean readIndexFromProdLocation = EarlybirdProperty.READ\_INDEX\_FROM\_PROD\_LOCATION.get(false);

String environment = EarlybirdProperty.ENV.get("no\_env\_specified"); // default value for tests.

String readIndexDir = EarlybirdProperty.HDFS\_INDEX\_SYNC\_DIR.get();

if (readIndexFromProdLocation) {

LOG.info("Will attempt to read index from prod locations");

LOG.info("Index directory provided: {}", readIndexDir);

// Replacing the path is a bit hacky, but it works ok.

readIndexDir = readIndexDir.replace("/" + environment + "/", "/prod/");

LOG.info("Will instead use index directory: {}", readIndexDir);

}

return readIndexDir;

}

/\*\*

\* Indexer for audio space events.

\*/

public AudioSpaceEventsStreamIndexer provideAudioSpaceEventsStreamIndexer(

AudioSpaceTable audioSpaceTable,

Clock clock) {

try {

return new AudioSpaceEventsStreamIndexer(

FinagleKafkaClientUtils.newKafkaConsumerForAssigning(

"",

new ThriftDeserializer<>(AudioSpaceBaseEvent.class),

"",

20

), audioSpaceTable, clock);

} catch (MissingKafkaTopicException ex) {

LOG.error("Missing kafka stream", ex);

return null;

}

}

/\*\*

\* Returns a class to start the Earlybird. See {@link EarlybirdStartup}.

\*/

public EarlybirdStartup provideEarlybirdStartup(

PartitionManager partitionManager,

UserUpdatesStreamIndexer userUpdatesStreamIndexer,

UserScrubGeoEventStreamIndexer userScrubGeoEventStreamIndexer,

AudioSpaceEventsStreamIndexer audioSpaceEventsStreamIndexer,

DynamicPartitionConfig dynamicPartitionConfig,

CriticalExceptionHandler criticalExceptionHandler,

SegmentManager segmentManager,

MultiSegmentTermDictionaryManager multiSegmentTermDictionaryManager,

QueryCacheManager queryCacheManager,

ZooKeeperTryLockFactory zooKeeperTryLockFactory,

ServerSetMember serverSetMember,

Clock clock,

SegmentSyncConfig segmentSyncConfig,

EarlybirdSegmentFactory earlybirdSegmentFactory,

EarlybirdCluster cluster,

SearchDecider decider) throws IOException {

if (cluster == EarlybirdCluster.FULL\_ARCHIVE) {

return new PartitionManagerStartup(clock, partitionManager);

}

// Check that the earlybird name is what we're expecting so we can build the kafka topics.

String earlybirdName = EarlybirdProperty.EARLYBIRD\_NAME.get();

Preconditions.checkArgument("earlybird-realtime".equals(earlybirdName)

|| "earlybird-protected".equals(earlybirdName)

|| "earlybird-realtime-exp0".equals(earlybirdName)

|| "earlybird-realtime\_cg".equals(earlybirdName));

StartupUserEventIndexer startupUserEventIndexer = new StartupUserEventIndexer(

provideSearchIndexingMetricSet(),

userUpdatesStreamIndexer,

userScrubGeoEventStreamIndexer,

segmentManager,

clock);

// Coordinate leaving the serverset to flush segments to HDFS.

CoordinatedEarlybirdAction actionCoordinator = new CoordinatedEarlybirdAction(

zooKeeperTryLockFactory,

"segment\_flusher",

dynamicPartitionConfig,

serverSetMember,

criticalExceptionHandler,

segmentSyncConfig);

actionCoordinator.setShouldSynchronize(true);

FileSystem hdfsFileSystem = HdfsUtil.getHdfsFileSystem();

EarlybirdIndexFlusher earlybirdIndexFlusher = new EarlybirdIndexFlusher(

actionCoordinator,

hdfsFileSystem,

EarlybirdProperty.HDFS\_INDEX\_SYNC\_DIR.get(),

segmentManager,

dynamicPartitionConfig.getCurrentPartitionConfig(),

clock,

new TimeLimitedHadoopExistsCall(hdfsFileSystem),

provideOptimizationAndFlushingCoordinationLock());

String baseTopicName = "search\_ingester\_%s\_events\_%s\_%s";

String earlybirdType;

if ("earlybird-protected".equals(earlybirdName)) {

earlybirdType = "protected";

} else if ("earlybird-realtime\_cg".equals(earlybirdName)) {

earlybirdType = "realtime\_cg";

} else {

earlybirdType = "realtime";

}

String tweetTopicName = String.format(

baseTopicName,

"indexing",

earlybirdType,

EarlybirdProperty.KAFKA\_ENV.get());

String updateTopicName = String.format(

baseTopicName,

"update",

earlybirdType,

EarlybirdProperty.KAFKA\_ENV.get());

LOG.info("Tweet topic: {}", tweetTopicName);

LOG.info("Update topic: {}", updateTopicName);

TopicPartition tweetTopic = new TopicPartition(

tweetTopicName,

dynamicPartitionConfig.getCurrentPartitionConfig().getIndexingHashPartitionID());

TopicPartition updateTopic = new TopicPartition(

updateTopicName,

dynamicPartitionConfig.getCurrentPartitionConfig().getIndexingHashPartitionID());

EarlybirdKafkaConsumersFactory earlybirdKafkaConsumersFactory =

provideEarlybirdKafkaConsumersFactory();

FreshStartupHandler freshStartupHandler = new FreshStartupHandler(

clock,

earlybirdKafkaConsumersFactory,

tweetTopic,

updateTopic,

segmentManager,

EarlybirdConfig.getMaxSegmentSize(),

EarlybirdConfig.getLateTweetBuffer(),

criticalExceptionHandler

);

TweetUpdateHandler updateHandler = new TweetUpdateHandler(segmentManager);

CoordinatedEarlybirdAction postOptimizationRebuilds = new CoordinatedEarlybirdAction(

zooKeeperTryLockFactory,

"post\_optimization\_rebuilds",

dynamicPartitionConfig,

serverSetMember,

criticalExceptionHandler,

segmentSyncConfig

);

postOptimizationRebuilds.setShouldSynchronize(true);

CoordinatedEarlybirdAction gcAction = new CoordinatedEarlybirdAction(

zooKeeperTryLockFactory,

"gc\_before\_optimization",

dynamicPartitionConfig,

serverSetMember,

criticalExceptionHandler,

segmentSyncConfig

);

gcAction.setShouldSynchronize(true);

TweetCreateHandler createHandler = new TweetCreateHandler(

segmentManager,

provideSearchIndexingMetricSet(),

criticalExceptionHandler,

multiSegmentTermDictionaryManager,

queryCacheManager,

postOptimizationRebuilds,

gcAction,

EarlybirdConfig.getLateTweetBuffer(),

EarlybirdConfig.getMaxSegmentSize(),

provideKafkaIndexCaughtUpMonitor(),

provideOptimizationAndFlushingCoordinationLock());

PartitionWriter partitionWriter = new PartitionWriter(

createHandler,

updateHandler,

criticalExceptionHandler,

PenguinVersion.versionFromByteValue(EarlybirdConfig.getPenguinVersionByte()),

clock);

KafkaConsumer<Long, ThriftVersionedEvents> rawKafkaConsumer =

earlybirdKafkaConsumersFactory.createKafkaConsumer(

"earlybird\_tweet\_kafka\_consumer");

EarlybirdKafkaConsumer earlybirdKafkaConsumer = provideKafkaConsumer(

criticalExceptionHandler,

rawKafkaConsumer,

tweetTopic,

updateTopic,

partitionWriter,

earlybirdIndexFlusher);

EarlybirdIndexLoader earlybirdIndexLoader = new EarlybirdIndexLoader(

hdfsFileSystem,

getIndexLoadingDirectory(), // See SEARCH-32839

EarlybirdProperty.ENV.get("default\_env\_value"),

dynamicPartitionConfig.getCurrentPartitionConfig(),

earlybirdSegmentFactory,

segmentSyncConfig,

clock);

this.storeEarlybirdStartupProducts(

createHandler,

partitionWriter,

earlybirdIndexFlusher

);

return new KafkaStartup(

segmentManager,

earlybirdKafkaConsumer,

startupUserEventIndexer,

userUpdatesStreamIndexer,

userScrubGeoEventStreamIndexer,

audioSpaceEventsStreamIndexer,

queryCacheManager,

earlybirdIndexLoader,

freshStartupHandler,

provideSearchIndexingMetricSet(),

multiSegmentTermDictionaryManager,

criticalExceptionHandler,

decider

);

}

public QualityFactor provideQualityFactor(

Decider decider,

SearchStatsReceiver searchStatsReceiver

) {

return new EarlybirdCPUQualityFactor(decider,

ManagementFactory.getPlatformMXBean(OperatingSystemMXBean.class),

searchStatsReceiver);

}

/\*\*

\* Returns a new UserUpdatesKafkaConsumer to read user updates.

\*/

public UserUpdatesStreamIndexer provideUserUpdatesKafkaConsumer(

SegmentManager segmentManager) {

try {

return new UserUpdatesStreamIndexer(

UserUpdatesStreamIndexer.provideKafkaConsumer(),

EarlybirdProperty.USER\_UPDATES\_KAFKA\_TOPIC.get(),

provideSearchIndexingMetricSet(),

segmentManager);

} catch (MissingKafkaTopicException ex) {

// Yes, it will crash the server. We've never seen this topic missing, but

// we've seen some others, so we had to build this functionality in the

// constructor. If one day this one goes missing, we'll have to figure out

// how to handle it. For now, we crash.

throw new RuntimeException(ex);

}

}

/\*\*

\* Returns a new UserScrubGeosKafkaConsumer to read geo scrubbing events.

\*/

public UserScrubGeoEventStreamIndexer provideUserScrubGeoEventKafkaConsumer(

SegmentManager segmentManager) {

try {

return new UserScrubGeoEventStreamIndexer(

UserScrubGeoEventStreamIndexer.provideKafkaConsumer(),

EarlybirdProperty.USER\_SCRUB\_GEO\_KAFKA\_TOPIC.get(),

provideSearchIndexingMetricSet(),

segmentManager);

} catch (MissingKafkaTopicException ex) {

/\*\*

\* See {@link #provideUserUpdatesKafkaConsumer}

\*/

throw new RuntimeException(ex);

}

}

/\*\*

\* Returns a new ProductionEarlybirdKafkaConsumer to read ThriftVersionedEvents.

\*/

public EarlybirdKafkaConsumersFactory provideEarlybirdKafkaConsumersFactory() {

return new ProductionEarlybirdKafkaConsumersFactory(

EarlybirdProperty.KAFKA\_PATH.get(),

MAX\_POLL\_RECORDS

);

}

/\*\*

\* Returns a class to read Tweets in the Earlybird. See {@link EarlybirdKafkaConsumer}.

\*/

public EarlybirdKafkaConsumer provideKafkaConsumer(

CriticalExceptionHandler criticalExceptionHandler,

KafkaConsumer<Long, ThriftVersionedEvents> rawKafkaConsumer,

TopicPartition tweetTopic,

TopicPartition updateTopic,

PartitionWriter partitionWriter,

EarlybirdIndexFlusher earlybirdIndexFlusher

) {

return new EarlybirdKafkaConsumer(

rawKafkaConsumer,

provideSearchIndexingMetricSet(),

criticalExceptionHandler,

partitionWriter,

tweetTopic,

updateTopic,

earlybirdIndexFlusher,

provideKafkaIndexCaughtUpMonitor());

}

}