package com.twitter.search.earlybird.config;

import java.util.Date;

import java.util.Map;

import java.util.Set;

import javax.annotation.Nullable;

import com.google.common.base.Preconditions;

import com.twitter.common.util.Clock;

import com.twitter.search.common.config.Config;

import com.twitter.search.common.config.ConfigFile;

import com.twitter.search.common.config.ConfigurationException;

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

import com.twitter.search.common.util.date.DateUtil;

/\*\*

\* This class provides APIs to access the tier configurations for a cluster.

\* Each tier has tier name, number of partitions, tier start time and end time.

\*/

public final class TierConfig {

private static final org.slf4j.Logger LOG = org.slf4j.LoggerFactory.getLogger(TierConfig.class);

private static final String DEFAULT\_CONFIG\_DIR = "common/config";

public static final String DEFAULT\_TIER\_FILE = "earlybird-tiers.yml";

public static final Date DEFAULT\_TIER\_START\_DATE = DateUtil.toDate(2006, 3, 21);

// It's convenient for DEFAULT\_TIER\_END\_DATE to be before ~2100, because then the output of

// FieldTermCounter.getHourValue(DEFAULT\_TIER\_END\_END\_DATE) can still fit into an integer.

public static final Date DEFAULT\_TIER\_END\_DATE = DateUtil.toDate(2099, 1, 1);

public static final String DEFAULT\_TIER\_NAME = "all";

public static final boolean DEFAULT\_ENABLED = true;

public static final TierInfo.RequestReadType DEFAULT\_READ\_TYPE = TierInfo.RequestReadType.LIGHT;

private static ConfigFile tierConfigFile = null;

private static ConfigSource tierConfigSource = null;

public enum ConfigSource {

LOCAL,

ZOOKEEPER

}

private TierConfig() { }

private static synchronized void init() {

if (tierConfigFile == null) {

tierConfigFile = new ConfigFile(DEFAULT\_CONFIG\_DIR, DEFAULT\_TIER\_FILE);

tierConfigSource = ConfigSource.LOCAL;

SearchLongGauge.export("tier\_config\_source\_" + tierConfigSource.name()).set(1);

LOG.info("Tier config file " + DEFAULT\_TIER\_FILE + " is successfully loaded from bundle.");

}

}

public static ConfigFile getConfigFile() {

init();

return tierConfigFile;

}

public static String getConfigFileName() {

return getConfigFile().getConfigFileName();

}

/\*\*

\* Return all the tier names specified in the config file.

\*/

public static Set<String> getTierNames() {

return Config.getConfig().getMapCopy(getConfigFileName()).keySet();

}

/\*\*

\* Sets the value of the given tier config property to the given value.

\*/

public static void setForTests(String property, Object value) {

Config.getConfig().setForTests(DEFAULT\_TIER\_FILE, property, value);

}

/\*\*

\* Returns the config info for the specified tier.

\*/

public static TierInfo getTierInfo(String tierName) {

return getTierInfo(tierName, null /\* use current environment \*/);

}

/\*\*

\* Returns the config info for the specified tier and environment.

\*/

public static TierInfo getTierInfo(String tierName, @Nullable String environment) {

String tierConfigFileType = getConfigFileName();

Map<String, Object> tierInfo;

try {

tierInfo = (Map<String, Object>) Config.getConfig()

.getFromEnvironment(environment, tierConfigFileType, tierName);

} catch (ConfigurationException e) {

throw new RuntimeException(e);

}

if (tierInfo == null) {

LOG.error("Cannot find tier config for "

+ tierName + "in config file: " + tierConfigFileType);

throw new RuntimeException("Configuration error: " + tierConfigFileType);

}

Long partitions = (Long) tierInfo.get("number\_of\_partitions");

if (partitions == null) {

LOG.error("No number of partition is specified for tier "

+ tierName + " in tier config file " + tierConfigFileType);

throw new RuntimeException("Configuration error: " + tierConfigFileType);

}

Long numTimeslices = (Long) tierInfo.get("serving\_timeslices");

if (numTimeslices == null) {

LOG.info("No max timeslices is specified for tier "

+ tierName + " in tier config file " + tierConfigFileType

+ ", not setting a cap on number of serving timeslices");

// NOTE: we use max int32 here because it will ultimately be cast to an int, but the config

// map expects Longs for all integral types. Using Long.MAX\_VALUE leads to max serving

// timeslices being set to -1 when it is truncated to an int.

numTimeslices = (long) Integer.MAX\_VALUE;

}

Date tierStartDate = (Date) tierInfo.get("data\_range\_start\_date\_inclusive");

if (tierStartDate == null) {

tierStartDate = DEFAULT\_TIER\_START\_DATE;

}

Date tierEndDate = (Date) tierInfo.get("data\_range\_end\_date\_exclusive");

if (tierEndDate == null) {

tierEndDate = DEFAULT\_TIER\_END\_DATE;

}

Boolean tierEnabled = (Boolean) tierInfo.get("tier\_enabled");

if (tierEnabled == null) {

tierEnabled = DEFAULT\_ENABLED;

}

TierInfo.RequestReadType readType =

getRequestReadType((String) tierInfo.get("tier\_read\_type"), DEFAULT\_READ\_TYPE);

TierInfo.RequestReadType readTypeOverride =

getRequestReadType((String) tierInfo.get("tier\_read\_type\_override"), readType);

return new TierInfo(

tierName,

tierStartDate,

tierEndDate,

partitions.intValue(),

numTimeslices.intValue(),

tierEnabled,

(String) tierInfo.get("serving\_range\_since\_id\_exclusive"),

(String) tierInfo.get("serving\_range\_max\_id\_inclusive"),

(Date) tierInfo.get("serving\_range\_start\_date\_inclusive\_override"),

(Date) tierInfo.get("serving\_range\_end\_date\_exclusive\_override"),

readType,

readTypeOverride,

Clock.SYSTEM\_CLOCK);

}

public static synchronized void clear() {

tierConfigFile = null;

tierConfigSource = null;

}

protected static synchronized ConfigSource getTierConfigSource() {

return tierConfigSource;

}

private static TierInfo.RequestReadType getRequestReadType(

String readTypeEnumName, TierInfo.RequestReadType defaultReadType) {

TierInfo.RequestReadType readType = defaultReadType;

if (readTypeEnumName != null) {

readType = TierInfo.RequestReadType.valueOf(readTypeEnumName.trim().toUpperCase());

Preconditions.checkState(readType != null);

}

return readType;

}

}