package com.twitter.search.feature\_update\_service.whitelist;

import java.io.InputStream;

import java.util.Set;

import java.util.concurrent.Executors;

import java.util.concurrent.ScheduledExecutorService;

import java.util.concurrent.atomic.AtomicReference;

import com.google.common.collect.ImmutableSet;

import com.google.common.util.concurrent.ThreadFactoryBuilder;

import org.yaml.snakeyaml.Yaml;

import com.twitter.common.util.Clock;

import com.twitter.finagle.thrift.ClientId;

import com.twitter.search.common.util.io.periodic.PeriodicFileLoader;

/\*\*

\* ClientIdWhitelist extends PeriodicFileLoader to load client whitelist

\* from configbus and checks to see if current clientId is allowed

\*/

public class ClientIdWhitelist extends PeriodicFileLoader {

private final AtomicReference<ImmutableSet<ClientId>> clientIdSet = new AtomicReference<>();

public ClientIdWhitelist(String clientIdWhitelistPath, ScheduledExecutorService executorService,

Clock clock) {

super("ClientIdWhitelist", clientIdWhitelistPath, executorService, clock);

}

/\*\*

\* Creates the object that manages loads from the clientIdWhitelistpath in config.

\* It periodically reloads the client whitelist file using the given executor service.

\*/

public static ClientIdWhitelist initWhitelist(

String clientIdWhitelistPath, ScheduledExecutorService executorService,

Clock clock) throws Exception {

ClientIdWhitelist clientIdWhitelist = new ClientIdWhitelist(

clientIdWhitelistPath, executorService, clock);

clientIdWhitelist.init();

return clientIdWhitelist;

}

/\*\*

\* Creates clock and executor service needed to create a periodic file loading object

\* then returns object that accpets file.

\* @param clientWhitelistPath

\* @return ClientIdWhitelist

\* @throws Exception

\*/

public static ClientIdWhitelist initWhitelist(String clientWhitelistPath) throws Exception {

Clock clock = Clock.SYSTEM\_CLOCK;

ScheduledExecutorService executorService = Executors.newSingleThreadScheduledExecutor(

new ThreadFactoryBuilder()

.setNameFormat("client-whitelist-reloader")

.setDaemon(true)

.build());

return initWhitelist(clientWhitelistPath, executorService, clock);

}

@Override

protected void accept(InputStream fileStream) {

ImmutableSet.Builder<ClientId> clientIdBuilder = new ImmutableSet.Builder<>();

Yaml yaml = new Yaml();

Set<String> set = yaml.loadAs(fileStream, Set.class);

for (String id : set) {

clientIdBuilder.add(ClientId.apply(id));

}

clientIdSet.set(clientIdBuilder.build());

}

// checks to see if clientId is in set of whitelisted clients

public boolean isClientAllowed(ClientId clientId) {

return clientIdSet.get().contains(clientId);

}

}