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

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Collection;

import java.util.List;

import java.util.concurrent.TimeUnit;

import com.google.common.base.Preconditions;

import com.google.inject.Module;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.app.Flag;

import com.twitter.app.Flaggable;

import com.twitter.finagle.Filter;

import com.twitter.finagle.Service;

import com.twitter.finagle.ThriftMux;

import com.twitter.finatra.annotations.DarkTrafficFilterType;

import com.twitter.finatra.decider.modules.DeciderModule$;

import com.twitter.finatra.mtls.thriftmux.modules.MtlsThriftWebFormsModule;

import com.twitter.finatra.mtls.thriftmux.AbstractMtlsThriftServer;

import com.twitter.finatra.thrift.filters.AccessLoggingFilter;

import com.twitter.finatra.thrift.filters.LoggingMDCFilter;

import com.twitter.finatra.thrift.filters.StatsFilter;

import com.twitter.finatra.thrift.filters.ThriftMDCFilter;

import com.twitter.finatra.thrift.filters.TraceIdMDCFilter;

import com.twitter.finatra.thrift.routing.JavaThriftRouter;

import com.twitter.inject.thrift.modules.ThriftClientIdModule$;

import com.twitter.search.common.constants.SearchThriftWebFormsAccess;

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

import com.twitter.search.common.util.PlatformStatsExporter;

import com.twitter.search.feature\_update\_service.filters.ClientIdWhitelistFilter;

import com.twitter.search.feature\_update\_service.modules.ClientIdWhitelistModule;

import com.twitter.search.feature\_update\_service.modules.EarlybirdUtilModule;

import com.twitter.search.feature\_update\_service.modules.FeatureUpdateServiceDiffyModule;

import com.twitter.search.feature\_update\_service.modules.FinagleKafkaProducerModule;

import com.twitter.search.feature\_update\_service.modules.FuturePoolModule;

import com.twitter.search.feature\_update\_service.modules.TweetypieModule;

import com.twitter.search.feature\_update\_service.thriftjava.FeatureUpdateService;

import com.twitter.thriftwebforms.MethodOptionsAccessConfig;

import com.twitter.util.ExecutorServiceFuturePool;

public class FeatureUpdateServiceThriftServer extends AbstractMtlsThriftServer {

private static final Logger LOG =

LoggerFactory.getLogger(FeatureUpdateServiceThriftServer.class);

// Ideally we would not have to access the "environment" flag here and we could instead pass

// a flag to the ThriftWebFormsModule that would either enable or disable thrift web forms.

// However, it is not simple to create our own TwitterModule that both extends the

// ThriftWebFormsModule and consumes an injected flag.

private Flag<String> envFlag = flag().create("environment",

"",

"Environment for service (prod, staging, staging1, devel)",

Flaggable.ofString());

FeatureUpdateServiceThriftServer(String[] args) {

BuildInfoStats.export();

PlatformStatsExporter.exportPlatformStats();

flag().parseArgs(args, true);

}

@Override

@SuppressWarnings("unchecked")

public Collection<Module> javaModules() {

List<Module> modules = new ArrayList<>();

modules.addAll(Arrays.asList(

ThriftClientIdModule$.MODULE$,

DeciderModule$.MODULE$,

new ClientIdWhitelistModule(),

new FinagleKafkaProducerModule(),

new EarlybirdUtilModule(),

new FuturePoolModule(),

new FeatureUpdateServiceDiffyModule(),

new TweetypieModule()));

// Only add the Thrift Web Forms module for non-prod services because we should

// not allow write access to production data through Thrift Web Forms.

String environment = envFlag.apply();

if ("prod".equals(environment)) {

LOG.info("Not including Thrift Web Forms because the environment is prod");

} else {

LOG.info("Including Thrift Web Forms because the environment is " + environment);

modules.add(

MtlsThriftWebFormsModule.create(

this,

FeatureUpdateService.ServiceIface.class,

MethodOptionsAccessConfig.byLdapGroup(SearchThriftWebFormsAccess.WRITE\_LDAP\_GROUP)

)

);

}

return modules;

}

@Override

public void configureThrift(JavaThriftRouter router) {

router

// Initialize Mapped Diagnostic Context (MDC) for logging

// (see https://logback.qos.ch/manual/mdc.html)

.filter(LoggingMDCFilter.class)

// Inject trace ID in MDC for logging

.filter(TraceIdMDCFilter.class)

// Inject request method and client ID in MDC for logging

.filter(ThriftMDCFilter.class)

// Log client access

.filter(AccessLoggingFilter.class)

// Export basic service stats

.filter(StatsFilter.class)

.filter(ClientIdWhitelistFilter.class)

.add(FeatureUpdateController.class);

}

@Override

public Service<byte[], byte[]> configureService(Service<byte[], byte[]> service) {

// Add the DarkTrafficFilter in "front" of the service being served.

return injector()

.instance(Filter.TypeAgnostic.class, DarkTrafficFilterType.class)

.andThen(service);

}

@Override

public ThriftMux.Server configureThriftServer(ThriftMux.Server server) {

// This cast looks redundant, but it is required for pants to compile this file.

return (ThriftMux.Server) server.withResponseClassifier(new FeatureUpdateResponseClassifier());

}

@Override

public void postWarmup() {

super.postWarmup();

ExecutorServiceFuturePool futurePool = injector().instance(ExecutorServiceFuturePool.class);

Preconditions.checkNotNull(futurePool);

onExit(() -> {

try {

futurePool.executor().shutdownNow();

futurePool.executor().awaitTermination(10L, TimeUnit.SECONDS);

} catch (InterruptedException e) {

LOG.error("Interrupted while awaiting future pool termination", e);

}

return null;

});

}

}