package com.twitter.search.earlybird;

import java.net.InetSocketAddress;

import java.util.concurrent.atomic.AtomicReference;

import org.apache.thrift.protocol.TCompactProtocol;

import org.apache.thrift.protocol.TProtocolFactory;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.finagle.ListeningServer;

import com.twitter.finagle.Service;

import com.twitter.finagle.SslException;

import com.twitter.finagle.ThriftMux;

import com.twitter.finagle.mtls.server.MtlsThriftMuxServer;

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

import com.twitter.finagle.stats.MetricsStatsReceiver;

import com.twitter.finagle.thrift.ThriftClientRequest;

import com.twitter.finagle.util.ExitGuard;

import com.twitter.finagle.zipkin.thrift.ZipkinTracer;

import com.twitter.search.common.dark.DarkProxy;

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

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

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

import com.twitter.search.earlybird.thrift.EarlybirdService;

import com.twitter.server.filter.AdmissionControl;

import com.twitter.server.filter.cpuAdmissionControl;

import com.twitter.util.Await;

import com.twitter.util.Duration;

import com.twitter.util.TimeoutException;

public class EarlybirdProductionFinagleServerManager implements EarlybirdFinagleServerManager {

private static final Logger LOG =

LoggerFactory.getLogger(EarlybirdProductionFinagleServerManager.class);

private final AtomicReference<ListeningServer> warmUpFinagleServer = new AtomicReference<>();

private final AtomicReference<ListeningServer> productionFinagleServer = new AtomicReference<>();

private final EarlybirdFinagleServerMonitor unhandledExceptionMonitor;

public EarlybirdProductionFinagleServerManager(

CriticalExceptionHandler criticalExceptionHandler) {

this.unhandledExceptionMonitor =

new EarlybirdFinagleServerMonitor(criticalExceptionHandler);

}

@Override

public boolean isWarmUpServerRunning() {

return warmUpFinagleServer.get() != null;

}

@Override

public void startWarmUpFinagleServer(EarlybirdService.ServiceIface serviceIface,

String serviceName,

int port) {

TProtocolFactory protocolFactory = new TCompactProtocol.Factory();

startFinagleServer(warmUpFinagleServer, "warmup",

new EarlybirdService.Service(serviceIface, protocolFactory),

protocolFactory, serviceName, port);

}

@Override

public void stopWarmUpFinagleServer(Duration serverCloseWaitTime) throws InterruptedException {

stopFinagleServer(warmUpFinagleServer, serverCloseWaitTime, "Warm up");

}

@Override

public boolean isProductionServerRunning() {

return productionFinagleServer.get() != null;

}

@Override

public void startProductionFinagleServer(DarkProxy<ThriftClientRequest, byte[]> darkProxy,

EarlybirdService.ServiceIface serviceIface,

String serviceName,

int port) {

TProtocolFactory protocolFactory = new TCompactProtocol.Factory();

startFinagleServer(productionFinagleServer, "production",

darkProxy.toFilter().andThen(new EarlybirdService.Service(serviceIface, protocolFactory)),

protocolFactory, serviceName, port);

}

@Override

public void stopProductionFinagleServer(Duration serverCloseWaitTime)

throws InterruptedException {

stopFinagleServer(productionFinagleServer, serverCloseWaitTime, "Production");

}

private void startFinagleServer(AtomicReference target, String serverDescription,

Service<byte[], byte[]> service, TProtocolFactory protocolFactory, String serviceName,

int port) {

target.set(getServer(service, serviceName, port, protocolFactory));

LOG.info("Started EarlybirdServer " + serverDescription + " finagle server on port " + port);

}

private ListeningServer getServer(

Service<byte[], byte[]> service, String serviceName, int port,

TProtocolFactory protocolFactory) {

MetricsStatsReceiver statsReceiver = new MetricsStatsReceiver();

ThriftMux.Server server = new MtlsThriftMuxServer(ThriftMux.server())

.withMutualTls(EarlybirdProperty.getServiceIdentifier())

.withServiceClass(EarlybirdService.class)

.withOpportunisticTls(OpportunisticTls.Required())

.withLabel(serviceName)

.withStatsReceiver(statsReceiver)

.withTracer(ZipkinTracer.mk(statsReceiver))

.withMonitor(unhandledExceptionMonitor)

.withProtocolFactory(protocolFactory);

if (cpuAdmissionControl.isDefined()) {

LOG.info("cpuAdmissionControl flag is set, replacing AuroraThrottlingAdmissionFilter"

+ " with LinuxCpuAdmissionFilter");

server = server

.configured(AdmissionControl.auroraThrottling().off().mk())

.configured(AdmissionControl.linuxCpu().useGlobalFlag().mk());

}

return server.serve(new InetSocketAddress(port), service);

}

private void stopFinagleServer(AtomicReference<ListeningServer> finagleServer,

Duration serverCloseWaitTime,

String serverDescription) throws InterruptedException {

try {

LOG.info("Waiting for " + serverDescription + " finagle server to close. "

+ "Current time is " + System.currentTimeMillis());

Await.result(finagleServer.get().close(), serverCloseWaitTime);

LOG.info("Stopped " + serverDescription + " finagle server. Current time is "

+ System.currentTimeMillis());

finagleServer.set(null);

} catch (TimeoutException e) {

LOG.warn(serverDescription + " finagle server did not shutdown cleanly.", e);

} catch (SslException e) {

// Closing the Thrift port seems to throw an SSLException (SSLEngine closed already).

// See SEARCH-29449. Log the exception and reset finagleServer, so that future calls to

// startProductionFinagleServer() succeed.

LOG.warn("Got a SSLException while trying to close the Thrift port.", e);

finagleServer.set(null);

} catch (InterruptedException e) {

// If we catch an InterruptedException here, it means that we're probably shutting down.

// We should propagate this exception, and rely on EarlybirdServer.stopThriftService()

// to do the right thing.

throw e;

} catch (Exception e) {

LOG.error(e.getMessage(), e);

} finally {

// If the finagle server does not close cleanly, this line prints details about

// the ExitGuards.

LOG.info(serverDescription + " server ExitGuard explanation: " + ExitGuard.explainGuards());

}

}

}