package com.twitter.search.earlybird\_root;

import javax.inject.Named;

import javax.inject.Singleton;

import com.google.inject.Key;

import com.google.inject.Provides;

import com.google.inject.util.Providers;

import com.twitter.app.Flag;

import com.twitter.app.Flaggable;

import com.twitter.common.util.Clock;

import com.twitter.common\_internal.text.version.PenguinVersionConfig;

import com.twitter.finagle.Name;

import com.twitter.finagle.Service;

import com.twitter.finagle.stats.StatsReceiver;

import com.twitter.inject.TwitterModule;

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

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

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

import com.twitter.search.common.root.LoggingSupport;

import com.twitter.search.common.root.RemoteClientBuilder;

import com.twitter.search.common.root.SearchRootWarmup;

import com.twitter.search.common.root.ValidationBehavior;

import com.twitter.search.common.root.WarmupConfig;

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

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

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

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

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

import com.twitter.search.earlybird\_root.common.EarlybirdRequestContext;

import com.twitter.search.earlybird\_root.common.InjectionNames;

import com.twitter.search.earlybird\_root.filters.EarlybirdClusterAvailableFilter;

import com.twitter.search.earlybird\_root.filters.MarkTweetSourceFilter;

import com.twitter.search.earlybird\_root.filters.RequestContextToEarlybirdRequestFilter;

import com.twitter.search.earlybird\_root.filters.RequestTypeCountFilter;

import com.twitter.search.earlybird\_root.filters.ServiceExceptionHandlingFilter;

import com.twitter.search.earlybird\_root.filters.ServiceResponseValidationFilter;

import com.twitter.search.earlybird\_root.filters.UnsetSuperRootFieldsFilter;

import com.twitter.util.Future;

public class SuperRootAppModule extends TwitterModule {

private final Flag<String> rootRealtimeFlag = createFlag(

"root-realtime",

"",

"Override the path to root-realtime",

Flaggable.ofString());

private final Flag<String> rootProtectedFlag = createFlag(

"root-protected",

"",

"Override the path to root-protected",

Flaggable.ofString());

private final Flag<String> rootArchiveFullFlag = createFlag(

"root-archive-full",

"",

"Override the path to root-archive-full",

Flaggable.ofString());

private final Flag<String> penguinVersionsFlag = createMandatoryFlag(

"penguin\_versions",

"Penguin versions to be tokenized",

"",

Flaggable.ofString());

@Override

public void configure() {

// SuperRoot uses all clusters, not just one. We bind EarlybirdCluster to null to indicate that

// there is not one specific cluster to use.

bind(Key.get(EarlybirdCluster.class)).toProvider(Providers.<EarlybirdCluster>of(null));

bind(EarlybirdService.ServiceIface.class).to(SuperRootService.class);

}

@Provides

SearchRootWarmup<EarlybirdService.ServiceIface, ?, ?> providesSearchRootWarmup(

Clock clock,

WarmupConfig config) {

return new EarlybirdWarmup(clock, config);

}

@Provides

@Singleton

@Named(InjectionNames.REALTIME)

private EarlybirdService.ServiceIface providesRealtimeIface(

RemoteClientBuilder<EarlybirdService.ServiceIface> builder,

ResolverProxy proxy) throws Exception {

Name name = proxy.resolve(rootRealtimeFlag.apply());

return builder.createRemoteClient(name, "realtime", "realtime\_");

}

@Provides

@Singleton

@Named(InjectionNames.REALTIME)

private Service<EarlybirdRequestContext, EarlybirdResponse> providesRealtimeService(

@Named(InjectionNames.REALTIME)

EarlybirdService.ServiceIface realtimeServiceIface,

RequestContextToEarlybirdRequestFilter requestContextToEarlybirdRequestFilter,

StatsReceiver statsReceiver,

SearchDecider decider) {

return buildClientService(

realtimeServiceIface,

new EarlybirdClusterAvailableFilter(decider, EarlybirdCluster.REALTIME),

new MarkTweetSourceFilter(ThriftTweetSource.REALTIME\_CLUSTER),

new ServiceExceptionHandlingFilter(EarlybirdCluster.REALTIME),

new ServiceResponseValidationFilter(EarlybirdCluster.REALTIME),

new RequestTypeCountFilter(EarlybirdCluster.REALTIME.getNameForStats()),

requestContextToEarlybirdRequestFilter,

new UnsetSuperRootFieldsFilter(),

new ClientLatencyFilter(EarlybirdCluster.REALTIME.getNameForStats()));

}

@Provides

@Singleton

@Named(InjectionNames.FULL\_ARCHIVE)

private EarlybirdService.ServiceIface providesFullArchiveIface(

RemoteClientBuilder<EarlybirdService.ServiceIface> builder,

ResolverProxy proxy) throws Exception {

Name name = proxy.resolve(rootArchiveFullFlag.apply());

return builder.createRemoteClient(name, "fullarchive", "full\_archive\_");

}

@Provides

@Singleton

@Named(InjectionNames.FULL\_ARCHIVE)

private Service<EarlybirdRequestContext, EarlybirdResponse> providesFullArchiveService(

@Named(InjectionNames.FULL\_ARCHIVE)

EarlybirdService.ServiceIface fullArchiveServiceIface,

RequestContextToEarlybirdRequestFilter requestContextToEarlybirdRequestFilter,

StatsReceiver statsReceiver,

SearchDecider decider) {

return buildClientService(

fullArchiveServiceIface,

new EarlybirdClusterAvailableFilter(decider, EarlybirdCluster.FULL\_ARCHIVE),

new MarkTweetSourceFilter(ThriftTweetSource.FULL\_ARCHIVE\_CLUSTER),

new ServiceExceptionHandlingFilter(EarlybirdCluster.FULL\_ARCHIVE),

new ServiceResponseValidationFilter(EarlybirdCluster.FULL\_ARCHIVE),

new RequestTypeCountFilter(EarlybirdCluster.FULL\_ARCHIVE.getNameForStats()),

requestContextToEarlybirdRequestFilter,

// Disable unset followedUserIds for archive since archive earlybirds rely on this field

// to rewrite query to include protected Tweets

new UnsetSuperRootFieldsFilter(false),

new ClientLatencyFilter(EarlybirdCluster.FULL\_ARCHIVE.getNameForStats()));

}

@Provides

@Singleton

@Named(InjectionNames.PROTECTED)

private EarlybirdService.ServiceIface providesProtectedIface(

RemoteClientBuilder<EarlybirdService.ServiceIface> builder,

ResolverProxy proxy) throws Exception {

Name name = proxy.resolve(rootProtectedFlag.apply());

return builder.createRemoteClient(name, "protected", "protected\_");

}

@Provides

@Singleton

@Named(InjectionNames.PROTECTED)

private Service<EarlybirdRequestContext, EarlybirdResponse> providesProtectedService(

@Named(InjectionNames.PROTECTED)

EarlybirdService.ServiceIface protectedServiceIface,

RequestContextToEarlybirdRequestFilter requestContextToEarlybirdRequestFilter,

StatsReceiver statsReceiver,

SearchDecider decider) {

return buildClientService(

protectedServiceIface,

new EarlybirdClusterAvailableFilter(decider, EarlybirdCluster.PROTECTED),

new MarkTweetSourceFilter(ThriftTweetSource.REALTIME\_PROTECTED\_CLUSTER),

new ServiceExceptionHandlingFilter(EarlybirdCluster.PROTECTED),

new ServiceResponseValidationFilter(EarlybirdCluster.PROTECTED),

new RequestTypeCountFilter(EarlybirdCluster.PROTECTED.getNameForStats()),

requestContextToEarlybirdRequestFilter,

new UnsetSuperRootFieldsFilter(),

new ClientLatencyFilter(EarlybirdCluster.PROTECTED.getNameForStats()));

}

/\*\*

\* Builds a Finagle Service based on a EarlybirdService.ServiceIface.

\*/

private Service<EarlybirdRequestContext, EarlybirdResponse> buildClientService(

final EarlybirdService.ServiceIface serviceIface,

EarlybirdClusterAvailableFilter earlybirdClusterAvailableFilter,

MarkTweetSourceFilter markTweetSourceFilter,

ServiceExceptionHandlingFilter serviceExceptionHandlingFilter,

ServiceResponseValidationFilter serviceResponseValidationFilter,

RequestTypeCountFilter requestTypeCountFilter,

RequestContextToEarlybirdRequestFilter requestContextToEarlybirdRequestFilter,

UnsetSuperRootFieldsFilter unsetSuperRootFieldsFilter,

ClientLatencyFilter latencyFilter) {

Service<EarlybirdRequest, EarlybirdResponse> service =

new Service<EarlybirdRequest, EarlybirdResponse>() {

@Override

public Future<EarlybirdResponse> apply(EarlybirdRequest requestContext) {

return serviceIface.search(requestContext);

}

};

// We should apply ServiceResponseValidationFilter first, to validate the response.

// Then, if the response is valid, we should tag all results with the appropriate tweet source.

// ServiceExceptionHandlingFilter should come last, to catch all possible exceptions (that were

// thrown by the service, or by ServiceResponseValidationFilter and MarkTweetSourceFilter).

//

// But before we do all of this, we should apply the EarlybirdClusterAvailableFilter to see if

// we even need to send the request to this cluster.

return earlybirdClusterAvailableFilter

.andThen(serviceExceptionHandlingFilter)

.andThen(markTweetSourceFilter)

.andThen(serviceResponseValidationFilter)

.andThen(requestTypeCountFilter)

.andThen(requestContextToEarlybirdRequestFilter)

.andThen(latencyFilter)

.andThen(unsetSuperRootFieldsFilter)

.andThen(service);

}

@Provides

public LoggingSupport<EarlybirdRequest, EarlybirdResponse> provideLoggingSupport(

SearchDecider decider) {

return new EarlybirdServiceLoggingSupport(decider);

}

@Provides

public ValidationBehavior<EarlybirdRequest, EarlybirdResponse> provideValidationBehavior() {

return new EarlybirdServiceValidationBehavior();

}

/\*\*

\* Provides the penguin versions that we should use to retokenize the query if requested.

\*/

@Provides

@Singleton

public PenguinVersionConfig providePenguinVersions() {

return SearchPenguinVersionsConfig.deserialize(penguinVersionsFlag.apply());

}

}