Making Elasticsearch flexible

Alexander Reelsen
@spinscale
alexander.reelsen@elasticsearch.com

Why extending at all?

- Keeping the core small
- Tailored functionality
- Corner cases
- Prevent custom builds

Plugin basics

Concepts

- Everything is wired via google guice
- Plugins allow you to wire in your own implementations
- Structure: Zip files in a special format

```
bin/plugin -install de.spinscale/elasticsearch-plugin-suggest/0.90.1-0.7
```

- Loading via es-plugins.properties

```
plugin=org.elasticsearch.plugin.suggest.SuggestPlugin
```

Creating a plugin

```
public class SuggestPlugin extends AbstractPlugin {

    public String name() {
        return "suggest";
    }

    public String description() {
        return "Suggest Plugin";
    }

    public void onModule(RestModule restModule) {
        restModule.addRestAction(RestSuggestAction.class);
    }

}
```

Plugin extension points

```
Collection<Class<? extends Module>> modules() {}

Collection<Class<? extends LifecycleComponent>> services() {}

Collection<Class<? extends Module>> indexModules() {}

Collection<Class<? extends CloseableIndexComponent>>
indexServices() {}

Collection<Class<? extends Module>> shardModules() {}

Collection<Class<? extends CloseableIndexComponent>>
shardServices() {}
```

Adding modules

```
public class SuggestPlugin extends AbstractPlugin {

  public Collection<Class<? extends Module>> shardModules() {
    Collection<Class<? extends Module>> modules =
                                          Lists.newArrayList();

    modules.add(ShardSuggestModule.class);
    return modules;
  }

}
```

Loading via guice

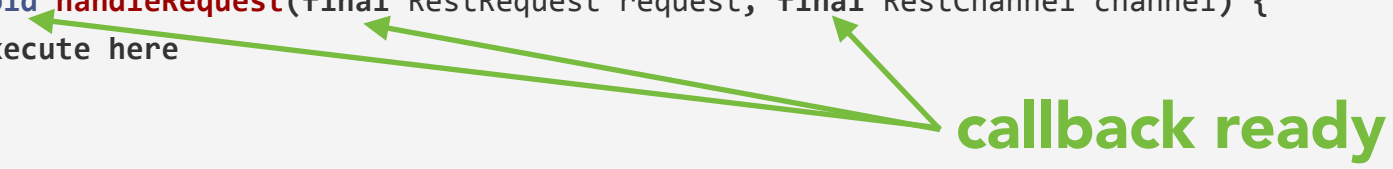
```
public class ShardSuggestModule extends AbstractModule {

    @Override
    protected void configure() {
        bind(ShardSuggestService.class).asEagerSingleton();
    }

}
```

Services, ShardServices

```
public class SuggestService extends AbstractLifecycleComponent<SuggestService> {

    private volatile Thread suggestUpdaterThread;

    @Inject
    public SuggestService(Settings settings, TransportSuggestRefreshAction
suggestRefreshAction, ClusterService clusterService, IndicesService indicesService) {
        super(settings);
    }

    protected void doStart() throws ElasticSearchException {
            suggestUpdaterThread = EsExecutors.daemonThreadFactory(settings,
"suggest_updater").newThread(new SuggestUpdaterThread());
            suggestUpdaterThread.start();
    }

    protected void doClose() throws ElasticSearchException {
        // stop thread
    }
}
```

HTTP REST endpoints

```
public class RestSuggestAction extends BaseRestHandler {

    @Inject
    public RestSuggestAction(Settings settings, Client client, RestController controller) {
        super(settings, client);
        controller.registerHandler(GET, "/{index}/__suggest", this);
        controller.registerHandler(GET, "/{index}/{type}/__suggest", this);
        controller.registerHandler(POST, "/{index}/__suggest", this);
        controller.registerHandler(POST, "/{index}/{type}/__suggest", this);
    }

    @Override
    public void handleRequest(final RestRequest request, final RestChannel channel) {
        // execute here
    }

}
```



HTTP REST endpoints

```
client.execute(SuggestAction.INSTANCE, suggestRequest, new ActionListener<SuggestResponse>()
{
  public void onResponse(SuggestResponse response) {
    try {
      XContentBuilder builder = RestXContentBuilder.restContentBuilder(request);
      builder.startObject();
      buildBroadcastShardsHeader(builder, response);
      builder.field("suggestions", response.suggestions());

      builder.endObject();
      channel.sendResponse(new XContentRestResponse(request, OK, builder));
    } catch (Exception e) {
      onFailure(e);
    }
  }

  public void onFailure(Throwable e) {
    handleException(channel, request, e);
  }
});
```

Adding own transport actions

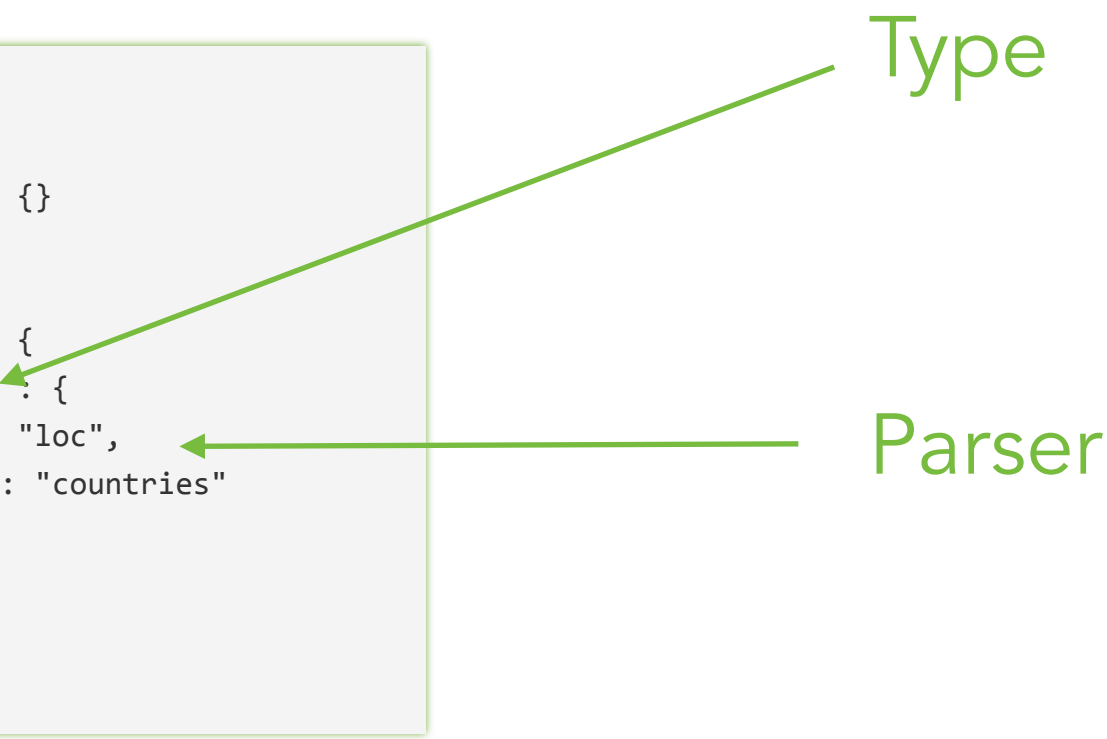
```
public class SuggestModule extends AbstractModule {

  protected void configure() {
    bind(TransportSuggestAction.class).asEagerSingleton();

    MapBinder<GenericAction, TransportAction> transportActionsBinder =
            MapBinder.newMapBinder(binder(), GenericAction.class,
TransportAction.class);
    transportActionsBinder.addBinding(SuggestAction.INSTANCE).to(TransportSugg
estAction.class).asEagerSingleton();


    MapBinder<String, GenericAction> actionsBinder =
MapBinder.newMapBinder(binder(), String.class, GenericAction.class);
    actionsBinder.addBinding(SuggestAction.NAME).toInstance(SuggestAction.INST
ANCE);
  }
}
```

Facets

Facets

- Extract data from query

```
{
  "query" : {
    "match_all" : {}
  },
  "facets" : {
    "countries" : {
      "georegion" : {
        "field" : "loc",
        "region" : "countries"
      }
    }
  }
}
```



FacetProcessor

```
public class GeoRegionFacetProcessor extends AbstractComponent implements FacetProcessor {

    public FacetCollector parse(String facetName, XContentParser parser,
                                SearchContext context) throws IOException {
        // some code before

        while ((token = parser.nextToken()) != XContentParser.Token.END_OBJECT) {
            if (token == XContentParser.Token.FIELD_NAME) {
                currentFieldName = parser.currentName();
            } else if (token.isValue()) {
                if ("field".equals(currentFieldName)) {
                    field = parser.text();
                } else if ("region".equals(currentFieldName)) {
                    region = parser.text();
                }
            }
        }

        // ...
        return new GeoRegionFacetCollector(facetName, field, region, context);
    }
}
```

FacetProcessor

```
public class GeoRegionFacetProcessor extends AbstractComponent implements FacetProcessor {

  public Facet reduce(String facetName, List<Facet> facets) {
    GeoRegionFacet geoRegionFacet = (GeoRegionFacet) facets.get(0);
    for (int i = 1 ; i < facets.size() ; i++) {
      Facet facet = facets.get(i);
      if (facet instanceof GeoRegionFacet) {
        GeoRegionFacet regionFacet = (GeoRegionFacet) facet;
        for (Map.Entry<String, AtomicLong> entry : regionFacet.counts().entrySet()) {
          if (geoRegionFacet.counts().containsKey(entry.getKey())) {
            geoRegionFacet.counts().get(entry.getKey()).addAndGet(entry.getValue().longValue
());
          } else {
            geoRegionFacet.counts().put(entry.getKey(), entry.getValue());
          }
        }
      }
    }
    return geoRegionFacet;
  }
}
```

FacetCollector

```
public class GeoRegionFacetCollector extends AbstractFacetCollector implements
FieldData.StringValueInDocProc {

  public void onValue(int docId, String value) {
    double[] location = GeoHashUtils.decode(value);
    Point point = ShapeBuilder.newPoint(location[0], location[1]);
    for (Map.Entry<String, Shape> entry : GeoRegionService.shapes.get(region).entrySet()) {
      if (entry.getValue().relate(point) == SpatialRelation.CONTAINS) {
        addOrIncrement(entry.getKey());
        return;
      }
    }

    addOrIncrement("_unknown");
  }

  public void onMissing(int docId) {
    addOrIncrement("_missing");
  }

}
```

Facet registration

```
public class GeoRegionFacetPlugin extends AbstractPlugin {

    // plugin name, description, setup here
    ...

    public void onModule(FacetModule facetModule) {
        facetModule.addFacetProcessor(GeoRegionFacetProcessor
.class);
    }

}
```

Mapping

Mapper

```
public class MapperAttachmentsPlugin extends AbstractPlugin {

    public String name() {
        return "mapper-attachments";
    }

    public String description() {
        return "Adds the attachment type allowing to parse difference
attachment formats";
    }

    public Collection<Class<? extends Module>> indexModules() {
        Collection<Class<? extends Module>> modules = newArrayList();
        modules.add(AttachmentsIndexModule.class);
        return modules;
    }
}
```

Mapper

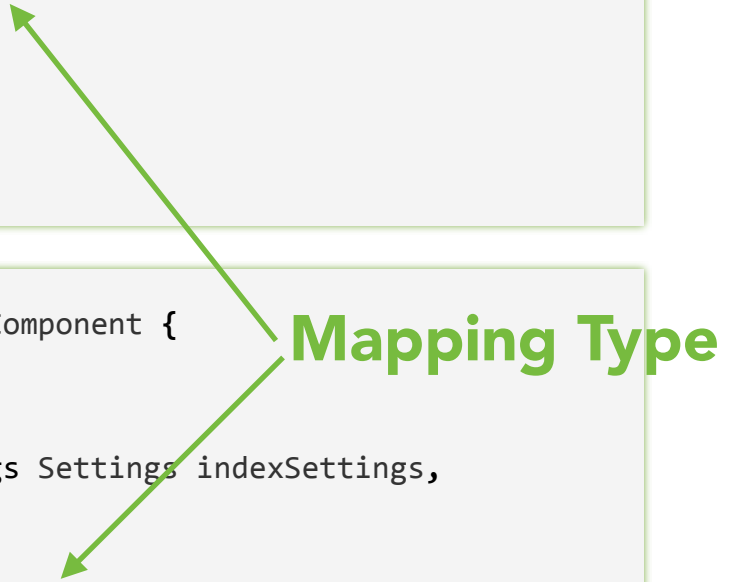
```
public class AttachmentsIndexModule extends AbstractModule {

    @Override
    protected void configure() {
        bind(RegisterAttachmentType.class).asEagerSingleton();
    }
}
```

```
public class RegisterAttachmentType extends AbstractIndexComponent {

  @Inject
  public RegisterAttachmentType(Index index, @IndexSettings Settings indexSettings,
MapperService mapperService) {
    super(index, indexSettings);
    mapperService.documentMapperParser().putTypeParser("attachment",
                                    new AttachmentMapper.TypeParser());
    }
}
```

Mapper

```
{
    "person" : {
        "properties" : {
            "my_attachment" : { "type" : "attachment" }
        }
    }
}
```

```
public class RegisterAttachmentType extends AbstractIndexComponent {

  @Inject
  public RegisterAttachmentType(Index index, @IndexSettings Settings indexSettings,
MapperService mapperService) {
    super(index, indexSettings);
    mapperService.documentMapperParser().putTypeParser("attachment",
                                    new AttachmentMapper.TypeParser());
    }
}
```



Mapper

```
public class AttachmentMapper implements Mapper {

  public String name() { return "attachment"; }

  public void parse(ParseContext context) throws IOException {}

  public void merge(Mapper mergeWith, MergeContext mergeContext) throws
MergeMappingException {}

  public XContentBuilder toXContent(XContentBuilder builder, Params params) throws
IOException {}
}
```

```
public static class TypeParser implements Mapper.TypeParser {
  public Mapper.Builder parse(String name, Map<String, Object> node, ParserContext
parserContext) throws MapperParsingException {
  }
}
```

Mapper

```
public static class TypeParser implements Mapper.TypeParser {
    public Mapper.Builder parse(String name, Map<String, Object> node,
                                ParserContext parserContext) throws MapperParsingException {
  }
}
```

PUT Mapping calls parse()

```
curl -X PUT http://localhost:9200/foo/bar/_mapping -d '{
  "person" : {
    "properties" : {
      "foo" : {
        "type" : "attachment"
      }
    }
  }
}'
```

Mapper

```
public class AttachmentMapper implements Mapper {

  public String name() { return "attachment"; }

  public void parse(ParseContext context) throws IOException {}

  public void merge(Mapper mergeWith, MergeContext mergeContext) throws
MergeMappingException {}

  public XContentBuilder toXContent(XContentBuilder builder, Params params) throws
IOException {}
}
```

GET Mapping calls toXContent()

```
curl -X GET http://localhost:9200/foo/bar/_mapping
```

Mapper

```
public class AttachmentMapper implements Mapper {

  public String name() { return "attachment"; }

  public void parse(ParseContext context) throws IOException {}

  public void merge(Mapper mergeWith, MergeContext mergeContext) throws
MergeMappingException {}

  public XContentBuilder toXContent(XContentBuilder builder, Params params) throws
IOException {}
}
```

Mapping Update calls `merge()`

```
curl -X PUT http://localhost:9200/foo/bar/_mapping -d '
{ "person" : { "properties" : { "foo" : { "type" : "attachment" } } } }'
```

Mapper

```
public class AttachmentMapper implements Mapper {

  public String name() { return "attachment"; }

  public void parse(ParseContext context) throws IOException {}

  public void merge(Mapper mergeWith, MergeContext mergeContext) throws
MergeMappingException {}

  public XContentBuilder toXContent(XContentBuilder builder, Params params) throws
IOException {}
}
```

Index operation calls parse()

```
curl -X PUT http://localhost:9200/foo/bar/1 -d '
{ "foo" : "BASE_64_ENCODED" }'
```

Highlighting

Highlighters

```
public class MyHighlightPlugin extends AbstractPlugin {

  // plugin name, description, setup here

  public void onModule(HighlightModule highlightModule) {
    highlightModule.registerHighlighter(MyHighlighter.class);
  }

}
```

Highlighters

```
{
    "query" : {...},
    "highlight" : {
        "type" : "my-highlighter",
        "fields" : { "content" : {} }
    }
}
```

Type

```
public class MyHighlighter implements Highlighter {

    @Override
    public String[] names() {
        return new String[] { "my", "my-highlighter" };
    }

    public HighlightField highlight(HighlighterContext highlighterContext) {
    }
}
```

Suggest

Suggester

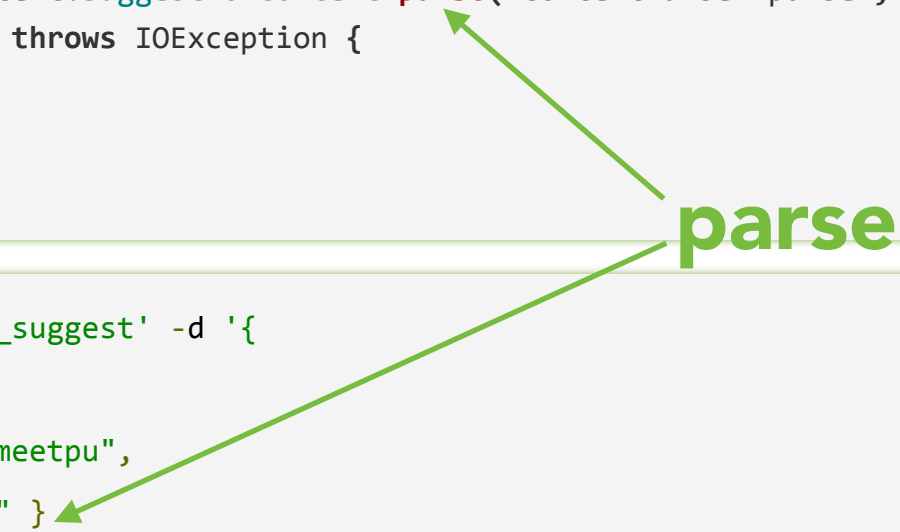
```
public class MySuggestPlugin extends AbstractPlugin {

  // plugin name, description, setup here

  public void onModule(SuggestModule suggestModule) {
    suggestModule.registerSuggester(MySuggester.class);
  }

}
```

Suggester

```
public final class MySuggester implements Suggester<MySuggestionContext> {

    @Override
    public String[] names() {
        return new String[] { "my", "my-suggester" };
    }

    @Override
    public SuggestContextParser getContextParser() {
        return new MySuggestParser(this);
    }

    public TermSuggestion execute(String name, MySuggestionContext suggestion, IndexReader
indexReader, CharsRef spare) throws IOException {
       // impl here
    }
}
```

Suggester

```
public final class MySuggestParser implements SuggestContextParser {

  private MySuggester suggester;

  public MySuggestParser(MySuggester suggester) {
      this.suggester = suggester;
  }

  public SuggestionSearchContext.SuggestionContext parse(XContentParser parser,
MapperService mapperService) throws IOException {
    // impl here
  }
}
```




```
curl -XPOST 'localhost:9200/_suggest' -d '{
  "my-suggestion" : {
    "text" : "the amsterdma meetpu",
    "my" : { "field" : "body" }
  }
}'
```

Rivers

Rivers

```
public class JsonRiverPlugin extends AbstractPlugin {

    public String name() {
        return "json";
    }

    public String description() {
        return "River Streaming JSON Plugin";
    }

    public void onModule(RiversModule module) {
        module.registerRiver("json", JsonRiverModule.class);
    }
}
```

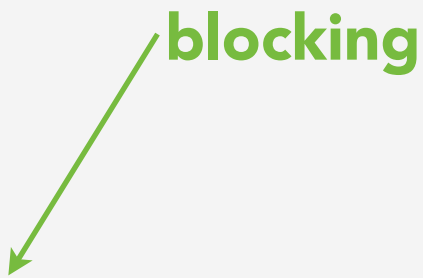
```
public class JsonRiverModule extends AbstractModule {
    protected void configure() {
        bind(River.class).to(JsonRiver.class).asEagerSingleton();
    }
}
```

Rivers

```
public class JsonRiver extends AbstractRiverComponent implements River {

  private volatile Thread slurperThread;
  private volatile Thread indexerThread;
  private final TransferQueue<RiverProduct> stream = new LinkedTransferQueue<RiverProduct>();

  @Override
  public void start() {
    // start both threads
    // hand over transferqueue
  }

  @Override
  public void close() {
  }
}
```

Rivers

```
private class Indexer implements Runnable {
    private BulkRequestBuilder bulk;

    public void run() {
        while (!closed) {
            try {
                RiverProduct product = stream.take();
                bulk = client.prepareBulk();
                do {
                    addProductToBulkRequest(product);
                } while ((product = stream.poll(250, TimeUnit.MILLISECONDS)) != null &&
deletedDocuments + insertedDocuments < RIVER_MAX_BULK_SIZE);
            } catch (InterruptedException e) {
                continue;
            } finally {
                bulk.execute().actionGet();
            }
        }
    }
}
```



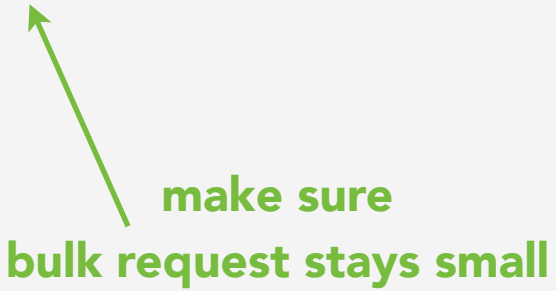
Rivers

```
private class Indexer implements Runnable {
    private BulkRequestBuilder bulk;

    public void run() {
        while (!closed) {
            try {
                RiverProduct product = stream.take();
                bulk = client.prepareBulk();
                do {
                    addProductToBulkRequest(product);
                } while ((product = stream.poll(250, TimeUnit.MILLISECONDS)) != null &&
deletedDocuments + insertedDocuments < RIVER_MAX_BULK_SIZE);
            } catch (InterruptedException e) {
                continue;
            } finally {
                bulk.execute().actionGet();
            }
        }
    }
}
```

wait for new data in stream

Rivers

```
private class Indexer implements Runnable {
    private BulkRequestBuilder bulk;

    public void run() {
        while (!closed) {
            try {
                RiverProduct product = stream.take();
                bulk = client.prepareBulk();
                do {
                    addProductToBulkRequest(product);
                } while ((product = stream.poll(250, TimeUnit.MILLISECONDS)) != null &&
deletedDocuments + insertedDocuments < RIVER_MAX_BULK_SIZE);
            } catch (InterruptedException e) {
                continue;
            } finally {
                bulk.execute().actionGet();
            }
        }
    }
}
```



Rivers

```
private class Indexer implements Runnable {
    private BulkRequestBuilder bulk;

    public void run() {
        while (!closed) {
            try {
                RiverProduct product = stream.take();
                bulk = client.prepareBulk();
                do {
                    addProductToBulkRequest(product);
                } while ((product = stream.poll(250, TimeUnit.MILLISECONDS)) != null &&
deletedDocuments + insertedDocuments < RIVER_MAX_BULK_SIZE);
            } catch (InterruptedException e) {
                continue;
            } finally {
                bulk.execute().actionGet();
            }
        }
    }
}
```

Scripting

Scripting

```
public class JavaScriptPlugin extends AbstractPlugin {

    @Override
    public String name() {
        return "lang-javascript";
    }

    @Override
    public String description() {
        return "JavaScript plugin allowing to add javascript scripting support";
    }

    public void onModule(ScriptModule module) {
        module.addScriptEngine(JavaScriptScriptEngineService.class);
    }
}
```

Scripting

```
public class JavaScriptScriptEngineService extends AbstractComponent implements
ScriptEngineService {

    public void close() {}

    public String[] types() { return new String[]{"js", "javascript"}; }

    public String[] extensions() { return new String[]{"js"}; }

    public Object compile(String script) {}

    public ExecutableScript executable(Object compiledScript, Map<String, Object> vars) {}

    public SearchScript search(Object compiledScript, SearchLookup lookup, @Nullable
Map<String, Object> vars) {}

    public Object execute(Object compiledScript, Map<String, Object> vars) {}

    public Object unwrap(Object value) {}
}
```

Tokenizer, Filter & Analyzer

Analyzer

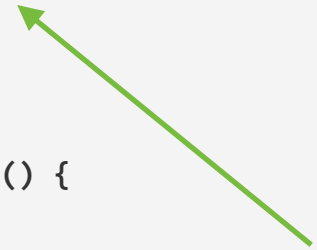
```
public class PolishIndicesAnalysis extends AbstractComponent {

  @Inject
  public PolishIndicesAnalysis(Settings settings, IndicesAnalysisService
indicesAnalysisService) {
    super(settings);
    indicesAnalysisService.analyzerProviderFactories().put("default",
       new PreBuiltAnalyzerProviderFactory("default", AnalyzerScope.INDICES,
         new PolishAnalyzer(Lucene.ANALYZER_VERSION)
       ));
  }

}
```

Analyzer

```
public class PolishAnalysisBinderProcessor extends AnalysisModule.AnalysisBinderProcessor {

    @Override
    public void processAnalyzers(AnalyzersBindings analyzersBindings) {
        analyzersBindings.processAnalyzer("polish", PolishAnalyzerProvider.class);
    }
}
```

Analyzer

```
public class PolishAnalyzerProvider extends AbstractIndexAnalyzerProvider<PolishAnalyzer> {

    private final PolishAnalyzer analyzer;

    @Inject
    public PolishAnalyzerProvider(Index index, @IndexSettings Settings indexSettings,
Environment env, @Assisted String name, @Assisted Settings settings) {
        super(index, indexSettings, name, settings);
        analyzer = new PolishAnalyzer(version, PolishAnalyzer.getDefaultStopSet());
    }

    @Override
    public PolishAnalyzer get() {
        return this.analyzer;
    }
}
```



Lucene Analyzer

Filter

```
public class EnglishMorphologyTokenFilterFactory extends AbstractTokenFilterFactory {
    private final LuceneMorphology luceneMorph;

    @Inject
    public EnglishMorphologyTokenFilterFactory(Index index,
            @IndexSettings Settings indexSettings, String name, Settings settings) {
        super(index, indexSettings, name, settings);
        try {
            luceneMorph = new EnglishLuceneMorphology();
        } catch (IOException ex) {
            throw new ElasticSearchIllegalArgumentException("Unable to load English
morphology analyzer", ex);
        }
    }

    public TokenStream create(TokenStream tokenStream) {
        return new MorphologyFilter(tokenStream, luceneMorph);
    }
}
```

Listeners

Index lifecycle listeners

```
public void shardRoutingChanged(IndexShard indexShard, @Nullable ShardRouting oldRouting,
ShardRouting newRouting) {

public void beforeIndexCreated(Index index) {}

public void afterIndexCreated(IndexService indexService) {}

public void beforeIndexShardCreated(ShardId shardId) {}

public void afterIndexShardCreated(IndexShard indexShard) {}

public void afterIndexShardStarted(IndexShard indexShard) {}

public void beforeIndexClosed(IndexService indexService) {}

public void afterIndexClosed(Index index) {}

public void beforeIndexShardClosed(ShardId shardId, @Nullable IndexShard indexShard)

public void afterIndexShardClosed(ShardId shardId) {}
```

Tons of listeners

- Remember: elasticsearch is async by nature

- `ClusterStateListener`
- `FieldMapperListener`
- `IndexingOperationListener`
- `LocalNodeMasterListener`
- `RiverClusterStateListener`

Similarity

Discovery

Index store

Transport

Snippets

Execute on master only

```
public class SuggestUpdaterThread implements Runnable {
    @Override
    public void run() {
        while (!closed) {
            DiscoveryNode node = clusterService.localNode();
            boolean isClusterStarted = clusterService.lifecycleState().
                                       equals(Lifecycle.State.STARTED);

            if (isClusterStarted && node != null && node.isMasterNode()) {
                suggestRefreshAction.execute(new SuggestRefreshRequest()).actionGet();
            }

            try {
                Thread.sleep(suggestRefreshInterval.millis());
            } catch (InterruptedException e1) {
                continue;
            }
        }
    }
}
```

Thanks for listening!

We're hiring
http://elasticsearch.com/about/jobs/

elasticsearch.