KBaseSearchEngine Documentation

Release 0.01

Gavin Price, Arfath Pasha

May 07, 2018

CONTENTS:

CHAPTER

ONE

INTRODUCTION

This project consists of,

1. an ETL application that extracts information from Workspace objects, transforms and loads a subset of this information into an ElasticSearch Index.
2. a search API that makes queries into the ElasticSearch index.

CHAPTER

TWO

ABOUT THIS RELEASE

CHAPTER

THREE

KNOWN ISSUES

CHAPTER FOUR

SYSTEM REQUIREMENTS

1. KBase Workspace (See section on Deploying the Workspace Service locally)
2. ElasticSearch v5.5.2. Note: the current implementation is not compatible with ElasticSearch v6+.
3. Kibana v5.5.2

CHAPTER

FIVE

CONFIGURATION DETAILS

When mapping storage object type versions to search object type versions, the storage object types must be integers. This means that versions that use the format major#.minor# (as in the case of KBase Workspace types) cannot be mapped. But since Search only cares about backwards incompatible changes it therefore expects only the major# in this case.

search_tools.cfg is for CLI operations (see search_tools.sh), deploy.cfg is for the service. deploy.cfg is the standard name we use for all services. Both cfgs need to be populated with the same key-values where there are duplicates. Note that the two cfgs are for separate applications - deployment and CLI. If they were merged, then there would be config items in the merged set that the service doesn't care about and the same goes for search_tools. In addtions, deploy.cfg contains templated variables that kb-sdk fills in and these templated variables won't get filled in in search_tools.cfg.

CHAPTER

SIX

INSTALLATION INSTRUCTIONS

6.1 Local Deployment

Follow these instructions for a local deployment once the *System Requirements* have been satisfied. These instructions are known to work on Ubuntu 16.04 LTS. The rest of this playbook assumes that you have all dependency binaries in your system environment path variable. At a high level, the steps are -

```
1. Start ElasticSearch
2. Start Kibana
3. Configure Workspace listeners to write events to Search mongo db
4. Restart Workspace service
5. Create a Workspace data type
6. Configure KBaseSearchEngine
7. Start worker
8. Start coordinator
9. Verify ElasticSearch index
```

1. Open a new terminal and start ElasticSearch.

Note: Elastic Search can only be started up by a non-root user

```
$ elasticsearch
```

2. Open a new terminal and start Kibana. Then open http://localhost:5601 in a browser tab to view the initial state of ElasticSearch.

```
$ kibana
```

3. Configure the Workspace listeners to write events to the Search mongodb.

```
$ gedit [PATH_TO_YOUR_WORKSPACE_DIR]/deploy.cfg
```

Add the following lines under the listener configuration section -

```
listeners = Search
listener-Search-class = us.kbase.workspace.modules.SearchPrototypeEventHandlerFactory
listener-Search-config-mongohost = localhost
listener-Search-config-mongodatabase = Search_test
listener-Search-config-mongouser = ""
listener-Search-config-mongopwd = ""
```

4. Restart the Workspace Service. (See section on Deploying the Workspace Service locally)

5. Open a new terminal and save the following document as Empty.spec. Then load into ipython, register the spec and save an object of this type to the Workspace. Saving a new object will cause the Workspace listener to write a new event to the mongo instance. Note that the ws.administer() command below requires administration privileges on the workspace.

```
module Empty {

    /* @optional foo */
    typedef structure {
        int foo;
    } AType;
};
```

```
$ ipython

In [1]: spec = open("[PATH_TO_SPEC]/Empty.spec").read()
In [2]: ws.request_module_ownership('Empty')
In [3]: ws.administer({'command': 'listModRequests'})
Out[4]:
[{u'moduleName': u'Empty', ...}]
In [5]: ws.administer({'command': 'approveModRequest', 'module': 'Empty'})
In [6]: ws.register_typespec({'spec': spec, 'new_types': ['AType'], 'dryrun': 0})
Out[7]: {u'Empty.Atype-0.1': ....}
In [8]: ws.release_module('Empty')
Out[9]: [u'Empty.AType-1.0']
In [10]: ws.save_objects({'id': 1, 'objects': [{'type': 'Empty.AType', 'data': {'bar
↪': 'baz'}, 'name': 'myobj'}]})
Out[11]:
[[1,
u'myobj',
...
]]
```

Create a new terminal and start mongo to check to make sure the event has been written. Note that the status is UNPROC (unprocessed event).

```
$ mongo
> show dbs
Search_test
admin
local
workspace
ws_types
> use Search_test
switched to db Search_test
> db.getCollectionNames()
["searchEvents"]
> db.searchEvents.findOne()
{
      "_id": ...,
      "strcde": "WS",
      "accgrp": 1,
      ...
      "status": "UNPROC"
}
```

6. Create a new terminal and edit search_tools.cfg, create a test data type and build the executable script.

```
$ cd [PATH_TO_YOUR_KBaseSearchEngine_DIR]
$ git checkout master
$ git pull
$ cp search_tools.cfg.example search_tools.cfg
$ gedit search_tools.cfg
```

Make the following edits. Note: the user for the token used below must have workspace admin privileges.

```
search-mongo-host=localhost
search-mongo-db=Search_test
elastic-host=localhost
elastic-port=9200
scratch=[PATH_TO_DIR_WHERE_TEMP_FILES_CAN_BE_STORED_BY_APP]
workspace-url=http://localhost:7058
auth-service-url=https://ci.kbase.us/services/auth/api/legacy/KBase/Sessions/Login
indexer-token=[YOUR_CI_TOKEN]
types-dir=[PATH_TO_YOUR_KBaseSearchEngine_DIR]/KBaseSearchEngine/test_types
type-mappings-dir=[PATH_TO_YOUR_KBaseSearchEngine_DIR]/KBaseSearchEngine/test_type_
↪mappings
workspace-mongo-host=fake
workspace-mongo-db=fake
```

```
$ mkdir test_types
$ cd test_types
$ gedit Empty.json
```

```
{
    "global-object-type": "EmptyAType2",
    "ui-type-name": "A Type",
    "storage-type": "WS",
    "storage-object-type": "Empty.AType",
    "indexing-rules": [
        {
            "path": "whee",
            "keyword-type": "string"
        },
        {
            "path": "whee2",
            "keyword-type": "string"
        }
    ]
}
```

```
$ cd ..
$ mkdir test_type_mappings
$ make build-executable-script JARS_DIR=[ABSOLUTE_PATH_TO_KBASE_JARS_DIR] KB_
↪RUNTIME=[PATH_TO_YOUR_ANT_INSTALL_DIR (example /usr/share)]
```

8. Start a worker

```
$ bin/search_tools.sh -c search_tools.cfg -k myworker
Press return to shut down process
```

9. Start the coordinator. Note that the event is processed and data has been indexed.

```
$ bin/search_tools.sh -c search_tools.cfg -s
Press return to shut down process
```

```
Moved event xxx NEW_VERSION WS:1/1/1 from UNPROC to READY
Event xxx NEW_VERSION WS:1/1/1 completed processing with state INDX on myworker
```

10. Open Kibana in browser with url localhost:/5601/app/kibana#/dev_tools/console?_g=()

On Kibana console, make the following query

```
GET _search
{
 "query": {
    "match_all": {}
 }
}

GET _cat/indices

GET kbase.1.emptytype2/data/_search
```

The results for the query should appear on the right panel.

6.2 Production Deployment

CHAPTER

SEVEN

API

CHAPTER

EIGHT

FUNCTIONAL REQUIREMENTS

CHAPTER

NINE

NONFUNCTIONAL REQUIREMENTS

CHAPTER

TEN

ARCHITECTURE - HIGH LEVEL DESIGN

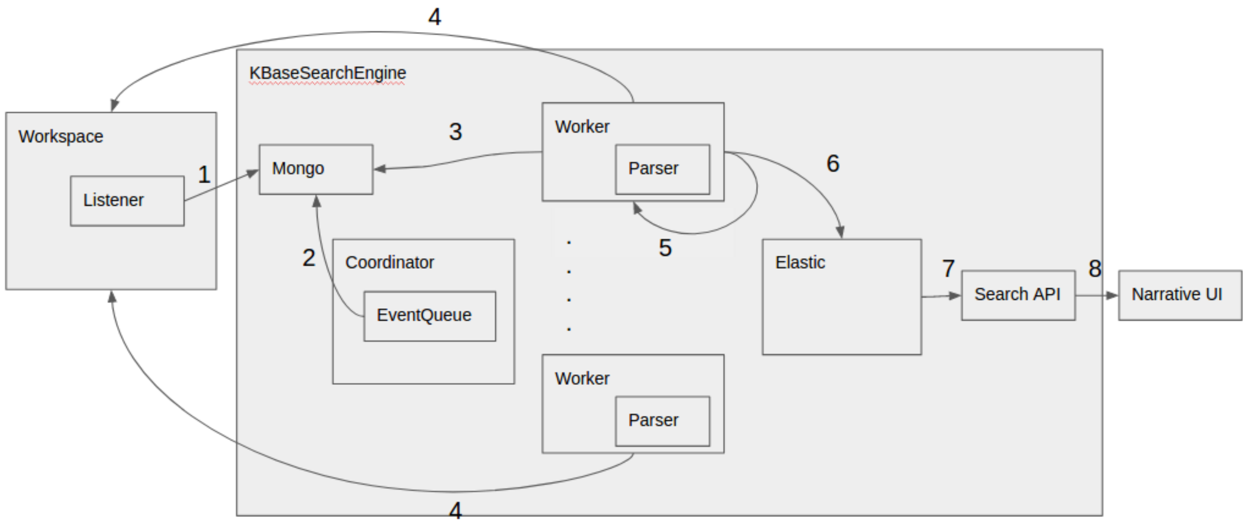

Fig. 10.1: KBaseSearchEngine component diagram.

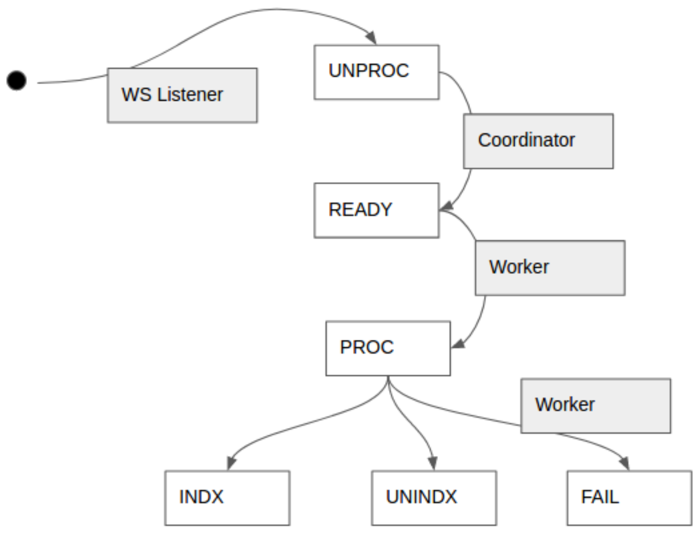

Fig. 10.2: Event state transition diagram.

10.1 Data Flow

1. The workspace pushes workspace level, object level and version level events into the KBaseSearchEngine Mongodb instance. The initial state of the events is UNPROC (or unprocessed).
2. The EventQueue periodically fetches events from the database and sets those that can be processed into a READY state. The EventQueue is a three-level blocking queue that blocks events that may cause an out-of-order update on the index. For example, an object level event like "rename" must block another object level event like "delete". i.e. these two events cannot be executed in parallel by the workers. Also, importantly, the queue prevents simultaneous updates on the same document in ElasticSearch, which can cause update conflicts.
3. The workers pull events that are READY for processing, set their state to PROCESSING in the database instance and begin processing the event.
4. When necessary (like for new version events for example) the workers make requests to the workspace to get object data for processing.
5. If the processing of an event fails due to network connectivity or other such recoverable issues, the event is reprocessed using a Retrier. The Retrier retries an event a finite number of times before setting the event processing state as FAILED in the mongo instance. A log is written out when processing of an event fails.
6. Once the event has been processed successfully, the corresponding object is (re-)indexed into the ElasticSearch index and the event state in the mongo instance is set to INDX (or indexed).

7&8. Queries from the narrative UI are serviced by the search API which in turn makes queries to the ElasticSearch index.

CHAPTER

ELEVEN

ARCHITECTURE - LOW LEVEL DESIGN

CHAPTER TWELVE

FORMAL SPECS

12.1 Type Mapping

### Type transformation specifications

Example type transformation *json* files are in *resources/types*.

TODO documentation of the transformation spec.

### Type mapping specifications

Type mappings are optional *yaml* files that specify how to map data source types to search types. If provided for a source type, they override the mapping provided in the type transformation file(s) (in the *source-type* and *source-object-type* fields). In particular, the mapping files are aware of the source type version while the transformation files are not.

Mapping files also allow using the same set of transformation files for multiple environments where the source types may not have equivalent identifiers by providing environment-specific mapping files.

There is an example mapping file in *resources/typemappings* that explains the structure and how the mappings work.

CHAPTER

THIRTEEN

PERFORMANCE EVALUATION

CHAPTER FOURTEEN

OPERATIONS

[TODO: A future PR will allow for search type versions to be alpha-numeric so we can have index names like genome_1_r1]

[TODO: A future PR will implement a CLI for trickle updates from a user specified time]

[TODO: A future PR will include strict dynamic mapping as a safety measure]

[TODO: A future PR will include html documentation]

14.1 Reindexing Naming Conventions

For each type of index, “genome” for example, we may have multiple indices in ElasticSearch with the naming convention “kbase.genome_1”, “kbase.genome_2” etc. where the namespace “kbase” comes from the “elastic-namespace” key-value pair in search_tools.cfg and each numeric suffix comes from the search type version (See resources/typemappings/GenomeAndAssembly.yaml.example).

In order to make it easy for the client to search across all indices of a certain type, aliases are applied with the following naming convention.

```
alias -> [indices]
genome -> [genome_1, genome_2 ... genome_n]
```

When reindexing is necessary, the index is reindexed to a new index with a new search type version. The first reindexing operation performed on genome_1 will result in a new index genome_1_r1 that replaces genome_1 in the genome alias. n reindexing operations performed on genome_1 will result in a new index genome_1_rn that replaces index genome_1_rn-1 in the genome alias.

14.2 Reindexing Process

For the sake of simplicity and for the reasons described below. a single process has been defined for all reindexing cases -

1. change field value
2. add field type
3. change field type
4. remove field type

The process involves reindexing an existing index into a new index for all of these cases. i.e. in-place reindexing is discouraged because the system may not have a recent snapshot/backup or any snapshot for recovery purposes should anything go wrong with the reindexing process.

In addition, given that there can be as many as a thousand indices for KBase data, maintenance can become a challenge if the process is not simple. Some level of simplicity has been achieved here by defiining a single process that covers all the reindexing cases. If necessary, the process may be further simplified through some level of automation as it matures over time.

Note: The commands below can be copy-pasted into Kibana and executed against the index. The corresponding curl commands can be obtained from Kibana by clicking on the little wrench icon that appears next to the pasted command.

Note: If any of the steps below fail, don't proceed until the issue is resolved by referring to the ElasticSearch documentation.

1. Note current time
2. Stop any workers that are performing trickle updates on the index that needs to be reindexed.
3. Refresh the index that needs reindexing to make sure it has been brought to a consistent state.

```
POST /kbase.genome_1/_refresh
```

4. Get a checksum for the index and record it in a separate file for later verification.

```
GET /kbase.genome_1/_stats/docs,store
```

5a. If mapping needs to be changed (for cases b, c, d above), get current index mapping.

```
GET kbase.genome_1/_mapping
```

5b. Copy-paste the mapping from the current index into the body section of the PUT command below and make the necessary field change (preferably one change per complete reindexing operation).

It is a good practice to make the mapping strict ("dynamic": "strict"). Strict mappings prevent the mapping from being modified dynamically during ingest time.

Update the settings section below the mapping. The number of shards and replicas must be decided on based on your capacity planning rules. It is costly to change the number of shards, so choose wisely! Make sure not to exceed 600 shards for any node in the system. Increase number of replicas to improve availability.

```
PUT kbase.genome_1_r1
{
  "mappings": {
    "data": {
      "dynamic": "strict",
      "_parent": {
        "type": "access"
      },
      "_routing": {
        "required": true
      },
      "properties": {
        "accgrp": {
          "type": "integer"
        },
        . . .
      }
    }
  },
  "settings": {
    "index": {
      "number_of_shards": "5",
      "number_of_replicas": "1"
    }
  }
}
```

5c. If the mapping does not require any change but the document's meta data needs to be changed, use the Painless script to modify metadata. Setting version_type to external will cause Elasticsearch to preserve the version from the source, create any documents that are missing, and update any documents that have an older version in the destination index than they do in the source index.

```
POST _reindex
{
  "source": {
    "index": "kbase.genome_1"
  },
  "dest": {
    "index": "kbase.genome_1_r1",
    "version_type": "external"
  },
  "script": {
    "lang": "painless",
    "inline": "if (ctx._source.foo == 'bar') {ctx._version++; ctx._source.remove('foo
↪')}"
  }
}
```

6. Now, reindex the entire data from current index to new index. Alternately, use a query to reindex only a subset of the current index.

```
POST _reindex
{
  "source": {
    "index": "kbase.genome_1"
  },
  "dest": {
    "index": "kbase.genome_1_r1"
  }
}

    OR

POST _reindex
{
  "source": {
    "index": "kbase.genome_1",
    "query": {
      ...
    }
  },
  "dest": {
    "index": "kbase.genome_1_r1"
  }
}
```

7. Run a checksum on the new index to make sure the numbers line up with the numbers of the current index.

```
GET /kbase.genome_1_r1/_stats/docs,store
```

8. Run a query to specifically check the change that was applied.

```
 GET kbase.genome_1_r1/_search

OR

 GET kbase.genome_1_r1/_search
 {
  "query": {
    "match": {
      "FIELD": "VALUE"
    }
  }
 }

OR

 https://www.elastic.co/guide/en/elasticsearch/reference/5.5/search-request-body.html
```

9. If the new index looks good, update index alias and delete current index.

Note: If you want the current index to linger for a day or two to serve a rollback option, reindex the current index into another new index called kbase.genome_1_backup and then delete the current index. This is one of two ways of renaming an index in ElasticSearch. The other way is to use the snapshot API.

```
POST _aliases
{
 "actions": [
 {
   "add": {
     "index": "kbase.genome_1_r1",
     "alias": "kbase.genome"
     }
   },
   {
     "remove": {
     "index": "kbase.genome_1",
     "alias": "kbase.genome"
   }
 }
 ]
}

DELETE kbase.genome_1
```

10. List all available indexes for the genome alias and all available genome indexes to ensure consistency across the alias map. Verify that all genome indexes that are present (except for backups) are referenced by the alias. Also verify that the alias does not contain an index reference for which no index exists.

```
GET /_cat/aliases/kbase.genome

GET /_cat/indices/kbase.genome_*
```

11. If the change involved in the reindexing operation also requires a corresponding search type spec change (located

in resources/types/genome.yml for example), then this change must be applied.

12. Change mapping version from "1" to "_r1" in the resources/types/genome.yml search type spec and add a comment (for future reference) that describes the change that took place in the r1 reindexing operation.
13. Restart trickle updates from the current time noted in step 1.

CHAPTER

FIFTEEN

TODOS (IDEAS FOR IMPROVEMENT)

1. It seems like it will be better to completely remove “full-text” as a key in the type specs and instead use “keyword-type: text”. It simplifies the spec. If “keyword-type: text” is currently substitutable for “full-text: true” then I think we should make this change and remove the “full-text” as a valid key. It seems like the option “full-text: false” and “keyword-type: string” (for structured content) is supported in ESv5 with just “keyword-type: keyword”.
2. Make all type specs YAML and skip any json files when loading type specs.
3. Must write a formal spec for type spec

CHAPTER

SIXTEEN

RELEASE NOTES

CHAPTER SEVENTEEN

INDICES AND TABLES

- genindex
- modindex
- search