KBaseSearchEngine Documentation

Release 0.01

Gavin Price, Arfath Pasha

Dec 07, 2017

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
3. Kibana v5.5.2

CHAPTER

FIVE

CONFIGURATION DETAILS

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.

```
$ kibana
```

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

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

Make the following changes -

```
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 and save the object to the Workspace. Saving a new object will cause the Workspace listener to write a new event to the mongo instance.

```
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.register_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 develop
$ git pull
$ cp search_tools.cfg.example search_tools.cfg
$ gedit search_tools.cfg
```

Make the following edits.

```
search-mongo-host=localhost
search-mongo-db=Search_test
elastic-host=localhost
elastic-port=9200
scratch=/home/apasha/dev/kbase/KBaseSearchEngine/temp_scratch
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 Emtpy.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 completed processing with state INDX on worked␣
↪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

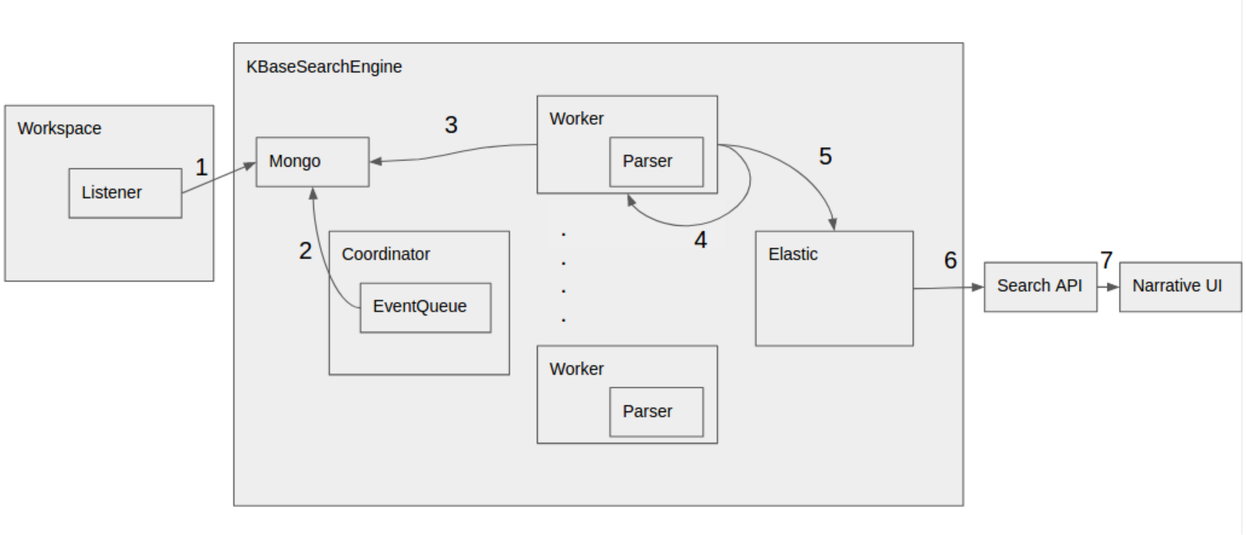

Fig. 10.1: KBaseSearchEngine component diagram.

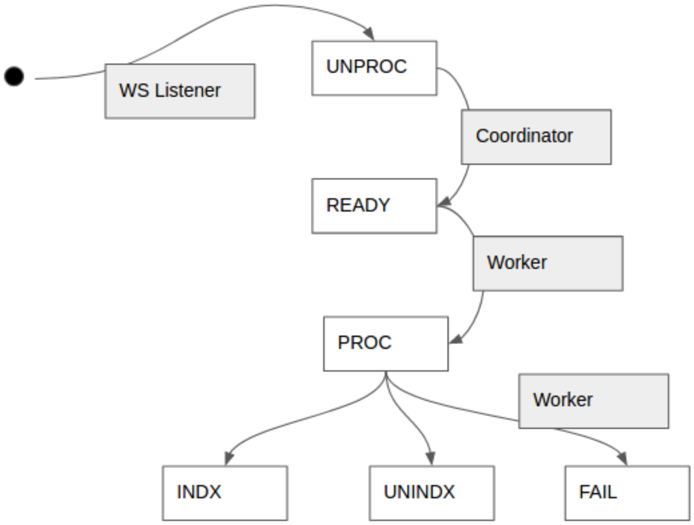

Fig. 10.2: Event state transition diagram.

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.
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. 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.
5. 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).

6&7. 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 names 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

TODOS

CHAPTER

FIFTEEN

RELEASE NOTES

CHAPTER

SIXTEEN

INDICES AND TABLES

- genindex
- modindex
- search