Cockatrice Documentation

Release 0.7.1

Minoru Osuka

Feb 25, 2019

Contents

Cockatrice is the open source search and indexing server written in Python that provides scalable indexing, search and advanced analysis/tokenization capabilities.

CHAPTER 1

Features

Cockatrice provides Indexing and search functionality implemented by Whoosh via the RESTful API based on Flask and it could bring up the cluster with Raft Consensus Algorithm by PySyncObj.

- Easy deployment
- Full-text search and indexing
- Per field similarity (scoring/ranking model) definition.
- Bringing up cluster
- Index replication
- Create indices snapshot
- Recover from indices snapshot
- Synchronize indices from leader node
- An easy-to-use RESTful API

CHAPTER 2

Source Codes

https://github.com/mosuka/cockatrice

CHAPTER 3

Requirements

Python 3.x interpreter

CHAPTER 4

Contents

4.1 Getting Started

Cockatrice makes it easy for programmers to develop search applications with advanced features. This section introduces you to the basic features to help you get up and running quickly.

4.1.1 Installing Cockatrice

Installation of Cockatrice on Unix-compatible or Windows servers generally requires Python interpreter and pip command.

Since Cockatrice is registered in PyPi, you can install it only by executing the following command.

```
$ pip install cockatrice
```

4.1.2 Starting Cockatrice

Cockatrice includes a command line interface tool called `cockatrice`. This tool allows you to start Cockatrice in your system.

You can easily start Cockatrice indexer like the following command:

```
$ cockatrice start indexer
```

The above command starts Cockatrice in the default state. `cockatrice` has many startup flags, so please refer to the help for details.

You can display the help by specifying the following:

```
$ cockatrice start indexer --help
```

When Cockatrice indexer started, following URL available:

```
$ curl -s -X GET http://localhost:8080/
```

You can see the result in plain text format. The result of the above command is:

```
cockatrice <VERSION> is running.
```

4.1.3 Create an index

You can not index documents yet just by starting Cockatrice. You need to create an index with a schema that tells how to index the documents. Creating an index needs to put the schema in the request. The following command creates an index named `myindex`:

```
$ curl -s -X PUT -H 'Content-type: application/yaml' --data-binary @./example/schema.
↪yaml http://localhost:8080/indices/myindex
```

The result of the above command can be seen in the JSON format as follows:

```
{
  "time": 0.30895185470581055,
  "status": {
    "code": 202,
    "phrase": "Accepted",
    "description": "Request accepted, processing continues off-line"
  }
}
```

4.1.4 Get an index

Information on the created index can be retrieve. The following command retrieves information on the index named `myindex`:

```
$ curl -s -X GET http://localhost:8080/indices/myindex
```

The result of the above command can be seen in the JSON format as follows:

```
{
  "index": {
    "name": "myindex",
    "doc_count": 0,
    "doc_count_all": 0,
    "last_modified": 1545792828.5970383,
    "latest_generation": 0,
    "version": -111,
    "storage": {
      "folder": "/tmp/cockatrice/index",
      "supports_mmap": true,
      "readonly": false,
      "files": [
        "_myindex_0.toc"
      ]
    }
  },
  "time": 0.0013620853424072266,
```

(continues on next page)

(continued from previous page)

```
  "status": {
    "code": 200,
    "phrase": "OK",
    "description": "Request fulfilled, document follows"
  }
}
```

4.1.5 Delete an index

Index that are no longer needed can be deleted. The following command deletes the index named `myindex`:

```
$ curl -s -X DELETE http://localhost:8080/indices/myindex
```

The result of the above command can be seen in the JSON format as follows:

```
{
  "time": 0.0001461505889892578,
  "status": {
    "code": 202,
    "phrase": "Accepted",
    "description": "Request accepted, processing continues off-line"
  }
}
```

4.1.6 Put a document

Indexing a document needs to put a document in the request that contains fields and its values. The following command indexes the document that id is `1` to the index named `myindex`:

```
$ curl -s -X PUT -H "Content-Type:application/json" http://localhost:8080/indices/
↪myindex/documents/1 --data-binary @./example/doc1.json
```

The result of the above command can be seen in the JSON format as follows:

```
{
  "time": 0.0008089542388916016,
  "status": {
    "code": 202,
    "phrase": "Accepted",
    "description": "Request accepted, processing continues off-line"
  }
}
```

4.1.7 Get a document

Information on the indexed document can be retrieve. The following command retrieves information on the document that id is `1` in the index named `myindex`:

```
$ curl -s -X GET http://localhost:8080/indices/myindex/documents/1
```

The result of the above command can be seen in the JSON format as follows:

```
{
  "fields": {
    "contributor": "43.225.167.166",
    "id": "1",
    "text": "A search engine is an information retrieval system designed to help find␣
↪information stored on a computer system. The search results are usually presented␣
↪in a list and are commonly called hits. Search engines help to minimize the time␣
↪required to find information and the amount of information which must be consulted,␣
↪akin to other techniques for managing information overload.\nThe most public,␣
↪visible form of a search engine is a Web search engine which searches for␣
↪information on the World Wide Web.",
    "timestamp": "20180704054100",
    "title": "Search engine (computing)"
  },
  "time": 0.014967918395996094,
  "status": {
    "code": 200,
    "phrase": "OK",
    "description": "Request fulfilled, document follows"
  }
}
```


4.1.8 Delete a document

Document that are no longer needed can be deleted. The following command deletes the document that id is `1` in the index named `myindex`:

```
$ curl -s -X DELETE http://localhost:8080/indices/myindex/documents/1
```

The result of the above command can be seen in the JSON format as follows:


```
{
  "time": 0.00019788742065429688,
  "status": {
    "code": 202,
    "phrase": "Accepted",
    "description": "Request accepted, processing continues off-line"
  }
}
```


4.1.9 Index documents in bulk

Include multiple documents in the request, you can index documents at once. The following command puts the documents in bulk into the index called `myindex`.

```
$ curl -s -X PUT -H "Content-Type:application/json" http://localhost:8080/indices/
↪myindex/documents --data-binary @./example/bulk_put.json
```

The result of the above command can be seen in the JSON format as follows:


```
{
  "time": 0.05237007141113281,
  "status": {
    "code": 202,
```


(continues on next page)

(continued from previous page)

```
    "phrase": "Accepted",
    "description": "Request accepted, processing continues off-line"
  }
}
```

4.1.10 Delete documents in bulk

Include multiple document IDs in the request, you can delete documents at once. The following command deletes the documents in bulk from an index named `myindex`.

```
$ curl -s -X DELETE -H "Content-Type:application/json" http://localhost:8080/indices/
↪myindex/documents --data-binary @./example/bulk_delete.json
```

The result of the above command can be seen in the JSON format as follows:

```
{
  "status": {
    "code": 202,
    "description": "Request accepted, processing continues off-line",
    "phrase": "Accepted"
  },
  "time": 0.0012569427490234375
}
```

4.1.11 Searching documents

You can specify the search parameters to search the index under various conditions. The following command searches documents containing the keyword `search` from an index named `myindex`.

```
$ curl -s -X GET http://localhost:8080/indices/myindex/search?query=search
```

The result of the above command can be seen in the JSON format as follows:

```
{
  "results": {
    "is_last_page": true,
    "page_count": 1,
    "page_len": 5,
    "page_num": 1,
    "total": 5,
    "hits": [
      {
        "doc": {
          "fields": {
            "contributor": "KolbertBot",
            "id": "3",
            "text": "Enterprise search is the practice of making content from␣
↪multiple enterprise-type sources, such as databases and intranets, searchable to a␣
↪defined audience.\n\"Enterprise search\" is used to describe the software of search␣
↪information within an enterprise (though the search function and its results may␣
↪still be public). Enterprise search can be contrasted with web search, which␣
↪applies search technology to documents on the open web, and desktop search, which␣
↪applies search technology to the content on a single computer.\nEnterprise search␣
↪systems index data and documents from a variety of sources such as: file systems,␣
↪intranets, document management systems, e-mail, and databases. Many enterprise␣
↪search systems integrate structured and unstructured data in their collections.[3]␣
↪Enterprise search systems also use access controls to enforce a security policy on␣
↪their users.\nEnterprise search can be seen as a type of vertical search of an␣
↪enterprise.",
```

(continues on next page)

(continued from previous page)


```
            "timestamp": "20180129125400",
            "title": "Enterprise search"
          }
        },
        "score": 1.8455226333928205,
        "rank": 0,
        "pos": 0
      },
      {
        "doc": {
          "fields": {
            "contributor": "Nurg",
            "id": "5",
            "text": "Federated search is an information retrieval technology that␣
↪allows the simultaneous search of multiple searchable resources. A user makes a␣
↪single query request which is distributed to the search engines, databases or other␣
↪query engines participating in the federation. The federated search then aggregates␣
↪the results that are received from the search engines for presentation to the user.␣
↪Federated search can be used to integrate disparate information resources within a␣
↪single large organization (\"enterprise\") or for the entire web. Federated search,␣
↪unlike distributed search, requires centralized coordination of the searchable␣
↪resources. This involves both coordination of the queries transmitted to the␣
↪individual search engines and fusion of the search results returned by each of them.
↪",
            "timestamp": "20180716000600",
            "title": "Federated search"
          }
        },
        "score": 1.8252014574100586,
        "rank": 1,
        "pos": 1
      },
      {
        "doc": {
          "fields": {
            "contributor": "Aistoff",
            "id": "2",
            "text": "A web search engine is a software system that is designed to␣
↪search for information on the World Wide Web. The search results are generally␣
↪presented in a line of results often referred to as search engine results pages␣
↪(SERPs). The information may be a mix of web pages, images, and other types of␣
↪files. Some search engines also mine data available in databases or open␣
↪directories. Unlike web directories, which are maintained only by human editors,␣
↪search engines also maintain real-time information by running an algorithm on a web␣
↪crawler. Internet content that is not capable of being searched by a web search␣
↪engine is generally described as the deep web.",
            "timestamp": "20181005132100",
            "title": "Web search engine"
          }
        },
        "score": 1.7381779253336536,
        "rank": 2,
        "pos": 2
      },
      {
        "doc": {
          "fields": {
```


(continues on next page)

(continued from previous page)


```
            "contributor": "43.225.167.166",
            "id": "1",
            "text": "A search engine is an information retrieval system designed to␣
↪help find information stored on a computer system. The search results are usually␣
↪presented in a list and are commonly called hits. Search engines help to minimize␣
↪the time required to find information and the amount of information which must be␣
↪consulted, akin to other techniques for managing information overload.\nThe most␣
↪public, visible form of a search engine is a Web search engine which searches for␣
↪information on the World Wide Web.",
            "timestamp": "20180704054100",
            "title": "Search engine (computing)"
          }
        },
        "score": 1.7118135656658342,
        "rank": 3,
        "pos": 3
      },
      {
        "doc": {
          "fields": {
            "contributor": "Citation bot",
            "id": "4",
            "text": "A distributed search engine is a search engine where there is no␣
↪central server. Unlike traditional centralized search engines, work such as␣
↪crawling, data mining, indexing, and query processing is distributed among several␣
↪peers in a decentralized manner where there is no single point of control.",
            "timestamp": "20180930171400",
            "title": "Distributed search engine"
          }
        },
        "score": 1.635459291513833,
        "rank": 4,
        "pos": 4
      }
    ]
  },
  "time": 0.015053987503051758,
  "status": {
    "code": 200,
    "phrase": "OK",
    "description": "Request fulfilled, document follows"
  }
}
```


4.2 Schema

Cockatrice fully supports the field types, analyzers, tokenizers and filters provided by Whoosh.

This section discusses how Cockatrice organizes its data into documents and fields, as well as how to work with a schema in Cockatrice.

4.2.1 Schema Design

Cockatrice defines the schema in YAML or JSON format.

The following items are defined in configuration:

- schema
- default_search_field
- field_types
- analyzers
- tokenizers
- filters

4.2.2 Schema

The schema is the place where you tell Cockatrice how it should build indexes from input documents.

```
schema:
  <FIELD_NAME>:
    field_type: <FIELD_TYPE>
    args:
      <ARG_NAME>: <ARG_VALUE>
      ...
```

```
{
  "schema": {
    <FIELD_NAME>: {
      "field_type": <FIELD_TYPE>,
      "args": {
        <ARG_NAME>: <ARG_VALUE>,
        ...
      }
    }
  }
}
```

- <FIELD_NAME>: The field name in the document.
- <FIELD_TYPE>: The field type used in this field.
- <ARG_NAME>: The argument name to use constructing the field.
- <ARG_VALUE>: The argument value to use constructing the field.

For example, id field used as a unique key is defined as following:

```
schema:
  id:
    field_type: id
    args:
      unique: true
      stored: true
```

```
{
  "schema": {
    "id": {
      "field_type": "id",
      "args": {
```

(continues on next page)

(continued from previous page)

```
        "unique": true,
        "stored": true
      }
    }
  }
}
```

4.2.3 Default Search Field

The query parser uses this as the field for any terms without an explicit field.

```
default_search_field: <FIELD_NAME>
```

```
{
  "default_search_field": <FIELD_NAME>
}
```

- `<FIELD_NAME>`: Uses this as the field name for any terms without an explicit field name.

For example, uses `text` field as default search field as following:

```
default_search_field: text
```

```
{
  "default_search_field": "text"
}
```

4.2.4 Field Types

The field type defines how Cockatrice should interpret data in a field and how the field can be queried. There are many field types included with Whoosh by default, and they can also be defined directly in YAML or JSON.

```
field_types:
  <FIELD_TYPE>:
    class: <FIELD_TYPE_CLASS>
    args:
      <ARG_NAME>: <ARG_VALUE>
      ...
```

```
{
  "field_types": {
    <FIELD_TYPE>: {
      "class": <FIELD_TYPE_CLASS>,
      "args": {
        <ARG_NAME>: <ARG_VALUE>,
        ...
      }
    }
  }
}
```

- `<FIELD_TYPE>`: The field type name.

- <FIELD_TYPE_CLASS>: The field type class.
- <ARG_NAME>: The argument name to use constructing the field type.
- <ARG_VALUE>: The argument value to use constructing the field type.

For example, defines `text` field type as following:

```
field_types:
  text:
    class: whoosh.fields.TEXT
    args:
      analyzer:
      phrase: true
      chars: false
      stored: false
      field_boost: 1.0
      multitoken_query: default
      spelling: false
      sortable: false
      lang: null
      vector: null
      spelling_prefix: spell_
```

```
{
  "field_types": {
    "text": {
      "class": "whoosh.fields.TEXT",
      "args": {
        "analyzer": null,
        "phrase": true,
        "chars": false,
        "stored": false,
        "field_boost": 1.0,
        "multitoken_query": "default",
        "spelling": false,
        "sortable": false,
        "lang": null,
        "vector": null,
        "spelling_prefix": "spell_"
      }
    }
  }
}
```

4.2.5 Analyzers

An analyzer examines the text of fields and generates a token stream. The simplest way to configure an analyzer is with a single `class` element whose class attribute is a fully qualified Python class name.

Even the most complex analysis requirements can usually be decomposed into a series of discrete, relatively simple processing steps. Cockatrice comes with a large selection of tokenizers and filters. Setting up an analyzer chain is very straightforward; you specify a `tokenizer` and `filters` to use, in the order you want them to run.

```
analyzers:
  <ANALYZER_NAME>:
    class: <ANALYZER_CLASS>
```

(continues on next page)

(continued from previous page)

```
    args:
      <ARG_NAME>: <ARG_VALUE>
      ...
  <ANALYZER_NAME>:
    tokenizer: <TOKENIZER_NAME>
    filters:
      - <FILTER_NAME>
      ...
```

```
{
  "analyzers": {
    <ANALYZER_NAME>: {
      "class": <ANALYZER_CLASS>,
      "args": {
        <ARG_NAME>: <ARG_VALUE>,
        ...
      }
    },
    <ANALYZER_NAME>: {
      "tokenizer": <TOKENIZER_NAME>,
      "filters": [
        <FILTER_NAME>,
        ...
      ]
    }
  }
}
```

- `<ANALYZER_NAME>`: The analyzer name.
- `<ANALYZER_CLASS>`: The analyzer class.
- `<ARG_NAME>`: The argument name to use constructing the analyzer.
- `<ARG_VALUE>`: The argument value to use constructing the analyzer.
- `<TOKENIZER_NAME>`: The tokenizer name to use in the analyzer chain.
- `<FILTER_NAME>`: The filter name to use in the analyzer chain.

For example, defines analyzers using `class`, `tokenizer` and `filters` as follows:

```
analyzers:
  simple:
    class: whoosh.analysis.SimpleAnalyzer
    args:
      expression: "\\w+(\\.?\\w+)*"
      gaps: false
  ngramword:
    tokenizer: regex
    filters:
      - lowercase
      - ngram
```

```
{
  "analyzers": {
    "simple": {
      "class": "whoosh.analysis.SimpleAnalyzer",
```

(continues on next page)

(continued from previous page)

```
      "args": {
        "expression": "\\w+(\\.?\\w+)*",
        "gaps": false
      }
    },
    "ngramword": {
      "tokenizer": "regex",
      "filters": [
        "lowercase",
        "ngram"
      ]
    }
  }
}
```

4.2.6 Tokenizers

The job of a tokenizer is to break up a stream of text into tokens, where each token is (usually) a sub-sequence of the characters in the text.

```
tokenizers:
  <TOKENIZER_NAME>:
    class: <TOKENIZER_CLASS>
    args:
      <ARG_NAME>: <ARG_VALUE>
      ...
```

```
{
  "tokenizers": {
    <TOKENIZER_NAME>: {
      "class": <TOKENIZER_CLASS>,
      "args": {
        <ARG_NAME>: ARG_VALUE>,
        ...
      }
    }
  }
}
```

- <TOKENIZER_NAME>: The tokenizer name.
- <TOKENIZER_CLASS>: The tokenizer class.
- <ARG_NAME>: The argument name to use constructing the tokenizer.
- <ARG_VALUE>: The argument value to use constructing the tokenizer.

For example, defines tokenizer as follows:

```
tokenizers:
  ngram:
    class: whoosh.analysis.NgramTokenizer
    args:
      minsize: 2
      maxsize: null
```

```
{
  "tokenizers": {
    "ngram": {
      "class": "whoosh.analysis.NgramTokenizer",
      "args": {
        "minsize": 2,
        "maxsize": null
      }
    }
  }
}
```

4.2.7 Filters

The job of a filter is usually easier than that of a tokenizer since in most cases a filter looks at each token in the stream sequentially and decides whether to pass it along, replace it or discard it.

```
filters:
  <FILTER_NAME>:
    class: <FILTER_CLASS>
    args:
      <ARG_NAME>: <ARG_VALUE>
      ...
```

```
{
  "filters": {
    <FILTER_NAME>: {
      "class": <FILTER_CLASS>,
      "args": {
        <ARG_NAME>: <ARG_VALUE>,
        ...
      }
    }
  }
}
```

- <FILTER_NAME>: The filter name.
- <FILTER_CLASS>: The filter class.
- <ARG_NAME>: The argument name to use constructing the filter.
- <ARG_VALUE>: The argument value to use constructing the filter.

For example, defines filter as follows:

```
filters:
  stem:
    class: whoosh.analysis.StemFilter
    args:
      lang: en
      ignore: null
      cachesize: 50000
```

```
{
  "filters": {
```

(continues on next page)

(continued from previous page)

```
    "stem": {
      "class": "whoosh.analysis.StemFilter",
      "args": {
        "lang": "en",
        "ignore": null,
        "cachesize": 50000
      }
    }
  }
}
```

4.2.8 Example

Refer to the example for how to define schema.

- YAML example: https://github.com/mosuka/cockatrice/blob/master/example/schema.yaml
- JSON example: https://github.com/mosuka/cockatrice/blob/master/example/schema.json

4.2.9 More information

See documents for more information.

- https://whoosh.readthedocs.io/en/latest/schema.html
- https://whoosh.readthedocs.io/en/latest/api/fields.html
- https://whoosh.readthedocs.io/en/latest/api/analysis.html

4.3 Scoring

Cockatrice fully supports the weighting module (scoring/ranking model) provided by Whoosh.

This section discusses how to work with a weighting in Cockatrice.

4.3.1 Weighting Design

Cockatrice defines the weighting in YAML or JSON format.

The following items are defined in configuration:

- weighting

4.3.2 Weighting

The weighting is the place where you tell Cockatrice how it should weighting documents in search from input queries.

```
weighting:
  default:
    class: <WEIGHTING_MODEL_CLASS>
    args:
      <ARG_NAME>: <ARG_VALUE>
```

(continues on next page)

(continued from previous page)

```
      ...
  <FIELD_NAME>:
    class: <WEIGHTING_MODEL_CLASS>
    args:
      <ARG_NAME>: <ARG_VALUE>
      ...
```

```
{
  "weighting": {
    "default": {
      "class": <WEIGHTING_MODEL_CLASS>,
      "args": {
        <ARG_NAME>: <ARG_VALUE>,
        ...
      }
    },
    <FIELD_NAME>: {
      "class": <WEIGHTING_MODEL_CLASS>
      "args": {
        <ARG_NAME>: <ARG_VALUE>,
        ...
      }
    }
  }
}
```

`default` is the weighting instance to use for fields not specified in the field names.

- `<FIELD_NAME>`: The field name.
- `<WEIGHTING_MODEL_CLASS>`: The weighting model class.
- `<ARG_NAME>`: The argument name to use constructing the weighting model.
- `<ARG_VALUE>`: The argument value to use constructing the weighting model.

For example, defines weighting model as following:

```
weighting:
  default:
    class: whoosh.scoring.BM25F
    args:
      B: 0.75
      K1: 1.2
  title:
    class: whoosh.scoring.TF_IDF
  text:
    class: whoosh.scoring.PL2
    args:
      c: 1.0
```

```
{
  "weighting": {
    "default": {
      "class": "whoosh.scoring.BM25F",
      "args": {
        "B": 0.75,
        "K1": 1.2
```

(continues on next page)

(continued from previous page)

```
      }
    },
    "title": {
      "class": "whoosh.scoring.TF_IDF"
    },
    "text": {
      "class": "whoosh.scoring.PL2",
      "args": {
        "c": 1.0
      }
    }
  }
}
```

4.3.3 Example

Refer to the example for how to define schema.

- YAML example: https://github.com/mosuka/cockatrice/blob/master/example/weighting.yaml
- JSON example: https://github.com/mosuka/cockatrice/blob/master/example/weighting.json

4.3.4 More information

See documents for more information.

- https://whoosh.readthedocs.io/en/latest/api/scoring.html

4.4 Cluster

Cockatrice includes the ability to set up a cluster of servers that combines fault tolerance and high availability.

4.4.1 Bring up a cluster

You already know how to start Cockatrice in standalone mode, but that is not fault tolerant. If you need to increase the fault tolerance, bring up a cluster.

You can easily bring up 3-node cluster with dynamic membership by following commands:

```
$ cockatrice start indexer --port=7070 --snapshot-file=/tmp/cockatrice/node1/index.
↪zip --data-dir=/tmp/cockatrice/node1/index --grpc-port 5050 --http-port=8080
$ cockatrice start indexer --port=7071 --snapshot-file=/tmp/cockatrice/node2/index.
↪zip --data-dir=/tmp/cockatrice/node2/index --grpc-port 5051 --http-port=8081 --peer-
↪addr=127.0.0.1:7070
$ cockatrice start indexer --port=7072 --snapshot-file=/tmp/cockatrice/node3/index.
↪zip --data-dir=/tmp/cockatrice/node3/index --grpc-port 5052 --http-port=8082 --peer-
↪addr=127.0.0.1:7070
```

Start by specifying the existing node in the cluster with the `--peer-addr` parameter.

Now you have a 3-nodes cluster. Then you can tolerate the failure of 1 node.

Above example shows each Cockatrice node running on the same host, so each node must listen on different ports. This would not be necessary if each node ran on a different host.

4.4.2 Get Cluster state

You will be wondering if the cluster is working properly. In such a case you can retrieve the cluster state with the following command;

```
$ curl -s -X GET http://localhost:8080/cluster
```

The result of the above command can be seen in the JSON format as follows:


```
{
  "cluster": {
    "version": "0.3.4",
    "revision": "2c8a3263d0dbe3f8d7b8a03e93e86d385c1de558",
    "self": "localhost:7070",
    "state": 2,
    "leader": "localhost:7070",
    "partner_nodes_count": 2,
    "partner_node_status_server_localhost:7071": 2,
    "partner_node_status_server_localhost:7072": 2,
    "readonly_nodes_count": 0,
    "unknown_connections_count": 0,
    "log_len": 4,
    "last_applied": 4,
    "commit_idx": 4,
    "raft_term": 1,
    "next_node_idx_count": 2,
    "next_node_idx_server_localhost:7071": 5,
    "next_node_idx_server_localhost:7072": 5,
    "match_idx_count": 2,
    "match_idx_server_localhost:7071": 4,
    "match_idx_server_localhost:7072": 4,
    "leader_commit_idx": 4,
    "uptime": 29,
    "self_code_version": 0,
    "enabled_code_version": 0
  },
  "time": 5.91278076171875e-05,
  "status": {
    "code": 200,
    "phrase": "OK",
    "description": "Request fulfilled, document follows"
  }
}
```


It is recommended to set an odd number of 3 or more for the number of nodes to bring up the cluster. In failure scenarios, data loss is inevitable, so avoid deploying single node.

Once the cluster is created, you can request that any node in the cluster be created index. The following command request to create index named `myindex` to `localhost:8080`:

```
$ curl -s -X PUT -H "Content-type: application/yaml" --data-binary @./conf/schema.
↪yaml http://localhost:8080/indices/myindex
```

If the above command succeeds, same index will be created on all the nodes in the cluster. Check your index on each nodes like follows:

```
$ curl -s -X GET http://localhost:8080/indices/myindex
$ curl -s -X GET http://localhost:8081/indices/myindex
$ curl -s -X GET http://localhost:8082/indices/myindex
```

Similarly, you can request to add any document to any node in the cluster. The following command requests to index documents in the index named `myindex` via `localhost:8080`:

```
$ curl -s -X PUT -H "Content-Type:application/json" http://localhost:8080/indices/
↪myindex/documents/1 -d @./example/doc1.json
```

If the above command succeeds, same document will be indexed on all the nodes in the cluster. Check your document on each nodes like follows:

```
$ curl -s -X GET http://localhost:8080/indices/myindex/documents/1
$ curl -s -X GET http://localhost:8081/indices/myindex/documents/1
$ curl -s -X GET http://localhost:8082/indices/myindex/documents/1
```

4.5 Monitoring

The `/metrics` endpoint provides access to all the metrics. Cockatrice outputs metrics in Prometheus exposition format.

4.5.1 Get metrics

You can get metrics by the following command:

```
$ curl -s -X GET http://localhost:8080/metrics
```

You can see the result in Prometheus exposition format. The result of the above command is:

```
# HELP cockatrice_http_requests_total The number of requests.
# TYPE cockatrice_http_requests_total counter
cockatrice_http_requests_total{endpoint="/myindex",method="PUT",status_code="202"} 1.0
cockatrice_http_requests_total{endpoint="/myindex/_docs",method="PUT",status_code="202
↪"} 1.0
# HELP cockatrice_http_requests_bytes_total A summary of the invocation requests␣
↪bytes.
# TYPE cockatrice_http_requests_bytes_total counter
cockatrice_http_requests_bytes_total{endpoint="/myindex",method="PUT"} 7376.0
cockatrice_http_requests_bytes_total{endpoint="/myindex/_docs",method="PUT"} 3909.0
# HELP cockatrice_http_responses_bytes_total A summary of the invocation responses␣
↪bytes.
# TYPE cockatrice_http_responses_bytes_total counter
cockatrice_http_responses_bytes_total{endpoint="/myindex",method="PUT"} 135.0
cockatrice_http_responses_bytes_total{endpoint="/myindex/_docs",method="PUT"} 137.0
# HELP cockatrice_http_requests_duration_seconds The invocation duration in seconds.
# TYPE cockatrice_http_requests_duration_seconds histogram
cockatrice_http_requests_duration_seconds_bucket{endpoint="/myindex",le="0.005",
↪method="PUT"} 0.0
cockatrice_http_requests_duration_seconds_bucket{endpoint="/myindex",le="0.01",method=
↪"PUT"} 0.0
```

(continues on next page)

(continued from previous page)

```
cockatrice_http_requests_duration_seconds_bucket{endpoint="/myindex",le="0.025",
↪method="PUT"} 0.0
cockatrice_http_requests_duration_seconds_bucket{endpoint="/myindex",le="0.05",method=
↪"PUT"} 0.0
cockatrice_http_requests_duration_seconds_bucket{endpoint="/myindex",le="0.075",
↪method="PUT"} 0.0
cockatrice_http_requests_duration_seconds_bucket{endpoint="/myindex",le="0.1",method=
↪"PUT"} 0.0
cockatrice_http_requests_duration_seconds_bucket{endpoint="/myindex",le="0.25",method=
↪"PUT"} 1.0
cockatrice_http_requests_duration_seconds_bucket{endpoint="/myindex",le="0.5",method=
↪"PUT"} 1.0
cockatrice_http_requests_duration_seconds_bucket{endpoint="/myindex",le="0.75",method=
↪"PUT"} 1.0
cockatrice_http_requests_duration_seconds_bucket{endpoint="/myindex",le="1.0",method=
↪"PUT"} 1.0
cockatrice_http_requests_duration_seconds_bucket{endpoint="/myindex",le="2.5",method=
↪"PUT"} 1.0
cockatrice_http_requests_duration_seconds_bucket{endpoint="/myindex",le="5.0",method=
↪"PUT"} 1.0
cockatrice_http_requests_duration_seconds_bucket{endpoint="/myindex",le="7.5",method=
↪"PUT"} 1.0
cockatrice_http_requests_duration_seconds_bucket{endpoint="/myindex",le="10.0",method=
↪"PUT"} 1.0
cockatrice_http_requests_duration_seconds_bucket{endpoint="/myindex",le="+Inf",method=
↪"PUT"} 1.0
cockatrice_http_requests_duration_seconds_count{endpoint="/myindex",method="PUT"} 1.0
cockatrice_http_requests_duration_seconds_sum{endpoint="/myindex",method="PUT"} 0.
↪22063422203063965
cockatrice_http_requests_duration_seconds_bucket{endpoint="/myindex/_docs",le="0.005",
↪method="PUT"} 1.0
cockatrice_http_requests_duration_seconds_bucket{endpoint="/myindex/_docs",le="0.01",
↪method="PUT"} 1.0
cockatrice_http_requests_duration_seconds_bucket{endpoint="/myindex/_docs",le="0.025",
↪method="PUT"} 1.0
cockatrice_http_requests_duration_seconds_bucket{endpoint="/myindex/_docs",le="0.05",
↪method="PUT"} 1.0
cockatrice_http_requests_duration_seconds_bucket{endpoint="/myindex/_docs",le="0.075",
↪method="PUT"} 1.0
cockatrice_http_requests_duration_seconds_bucket{endpoint="/myindex/_docs",le="0.1",
↪method="PUT"} 1.0
cockatrice_http_requests_duration_seconds_bucket{endpoint="/myindex/_docs",le="0.25",
↪method="PUT"} 1.0
cockatrice_http_requests_duration_seconds_bucket{endpoint="/myindex/_docs",le="0.5",
↪method="PUT"} 1.0
cockatrice_http_requests_duration_seconds_bucket{endpoint="/myindex/_docs",le="0.75",
↪method="PUT"} 1.0
cockatrice_http_requests_duration_seconds_bucket{endpoint="/myindex/_docs",le="1.0",
↪method="PUT"} 1.0
cockatrice_http_requests_duration_seconds_bucket{endpoint="/myindex/_docs",le="2.5",
↪method="PUT"} 1.0
cockatrice_http_requests_duration_seconds_bucket{endpoint="/myindex/_docs",le="5.0",
↪method="PUT"} 1.0
cockatrice_http_requests_duration_seconds_bucket{endpoint="/myindex/_docs",le="7.5",
↪method="PUT"} 1.0
cockatrice_http_requests_duration_seconds_bucket{endpoint="/myindex/_docs",le="10.0",
↪method="PUT"} 1.0
```

(continues on next page)

(continued from previous page)

```
cockatrice_http_requests_duration_seconds_bucket{endpoint="/myindex/_docs",le="+Inf",
↪method="PUT"} 1.0
cockatrice_http_requests_duration_seconds_count{endpoint="/myindex/_docs",method="PUT
↪"} 1.0
cockatrice_http_requests_duration_seconds_sum{endpoint="/myindex/_docs",method="PUT"}␣
↪0.0020329952239990234
# HELP cockatrice_index_documents The number of documents.
# TYPE cockatrice_index_documents gauge
cockatrice_index_documents{index_name="myindex"} 5.0
```

4.6 Health check

Cockatrice provides a health check endpoint which returns 200 if Cockatrice is live or ready to response to queries.

4.6.1 Liveness probe

To get the current liveness probe is following:

```
$ curl -s -X GET http://localhost:8080/liveness
```

You can see the result in JSON format. The result of the above command is:

```
{
  "liveness": true,
  "time": 7.152557373046875e-06,
  "status": {
    "code": 200,
    "phrase": "OK",
    "description": "Request fulfilled, document follows"
  }
}
```

4.6.2 Readiness probe

To get the current readiness probe is following:

```
$ curl -s -X GET http://localhost:8080/readiness
```

You can see the result in JSON format. The result of the above command is:

```
{
  "readiness": true,
  "time": 1.6927719116210938e-05,
  "status": {
    "code": 200,
    "phrase": "OK",
    "description": "Request fulfilled, document follows"
  }
}
```

4.7 RESTful API Reference

4.7.1 Index APIs

The Index API is used to manage individual indices.

Put Index API

The Create Index API is used to manually create an index in Cockatrice. The most basic usage is the following:

```
PUT /indices/<INDEX_NAME>?sync=<SYNC>&output=<OUTPUT>
---
schema:
  id:
    field_type: id
    args:
      unique: true
      stored: true
...
```

- `<INDEX_NAME>`: The index name.
- `<SYNC>`: Specifies whether to execute the command synchronously or asynchronously. If `True` is specified, command will execute synchronously. Default is `False`, command will execute asynchronously.
- `<OUTPUT>`: The output format. `json` or `yaml`. Default is `json`.
- Request Body: JSON or YAML formatted schema definition.

Get Index API

The Get Index API allows to retrieve information about the index. The most basic usage is the following:

```
GET /indices/<INDEX_NAME>?output=<OUTPUT>
```

- `<INDEX_NAME>`: The index name.
- `<OUTPUT>`: The output format. `json` or `yaml`. Default is `json`.

Delete Index API

The Delete Index API allows to delete an existing index. The most basic usage is the following:

```
DELETE /indices/<INDEX_NAME>?sync=<SYNC>&output=<OUTPUT>
```

- `<INDEX_NAME>`: The index name.
- `<SYNC>`: Specifies whether to execute the command synchronously or asynchronously. If `True` is specified, command will execute synchronously. Default is `False`, command will execute asynchronously.
- `<OUTPUT>`: The output format. `json` or `yaml`. Default is `json`.

4.7.2 Document APIs

Get Document API

```
GET /indices/<INDEX_NAME>/documents/<DOC_ID>?output=<OUTPUT>
```

- <INDEX_NAME>: The index name.
- <DOC_ID>: The document ID to retrieve.
- <OUTPUT>: The output format. `json` or `yaml`. Default is `json`.

Put Document API

```
PUT /indices/<INDEX_NAME>/documents/<DOC_ID>?sync=<SYNC>&output=<OUTPUT>
{
  "name": "Cockatrice",
  ...
}
```

- <INDEX_NAME>: The index name.
- <DOC_ID>: The document ID to index.
- <SYNC>: Specifies whether to execute the command synchronously or asynchronously. If `True` is specified, command will execute synchronously. Default is `False`, command will execute asynchronously.
- <OUTPUT>: The output format. `json` or `yaml`. Default is `json`.
- Request Body: JSON or YAML formatted fields definition.

Delete Document API

```
DELETE /indices/<INDEX_NAME>/documents/<DOC_ID>?sync=<SYNC>&output=<OUTPUT>
```

- <INDEX_NAME>: The index name.
- <DOC_ID>: The document ID to delete.
- <SYNC>: Specifies whether to execute the command synchronously or asynchronously. If `True` is specified, command will execute synchronously. Default is `False`, command will execute asynchronously.
- <OUTPUT>: The output format. `json` or `yaml`. Default is `json`.

Put Documents API

```
PUT /indices/<INDEX_NAME>/documents?sync=<SYNC>&output=<OUTPUT>
[
  {
    "id": "1",
    "name": "Cockatrice"
  },
  {
    "id": "2",
  ...
]
```

- <INDEX_NAME>: The index name.
- <SYNC>: Specifies whether to execute the command synchronously or asynchronously. If `True` is specified, command will execute synchronously. Default is `False`, command will execute asynchronously.
- <OUTPUT>: The output format. `json` or `yaml`. Default is `json`.
- Request Body: JSON or YAML formatted documents definition.

Delete Documents API

```
DELETE /indices/<INDEX_NAME>/documents?sync=<SYNC>&output=<OUTPUT>
[
  "1",
  "2",
  ...
]
```

- <INDEX_NAME>: The index name.
- <SYNC>: Specifies whether to execute the command synchronously or asynchronously. If `True` is specified, command will execute synchronously. Default is `False`, command will execute asynchronously.
- <OUTPUT>: The output format. `json` or `yaml`. Default is `json`.
- Request Body: JSON or YAML formatted document ids definition.

4.7.3 Search APIs

Search API

```
GET /indices/<INDEX_NAME>/search?query=<QUERY>&search_field=<SEARCH_FIELD>&page_num=
↪<PAGE_NUM>&page_len=<PAGE_LEN>&output=<OUTPUT>
```

- <INDEX_NAME>: The index name to search.
- <QUERY>: The unicode string to search index.
- <SEARCH_FIELD>: Uses this as the field for any terms without an explicit field.
- <PAGE_NUM>: The page number to retrieve, starting at `1` for the first page.
- <PAGE_LEN>: The number of results per page.
- <OUTPUT>: The output format. `json` or `yaml`. Default is `json`.

4.7.4 Cluster APIs

Get Cluster API

```
GET /cluster?output=<OUTPUT>
```

- <OUTPUT>: The output format. `json` or `yaml`. Default is `json`.

Add Node API

```
PUT /cluster/<NODE_NAME>?output=<OUTPUT>
```

- <NODE_NAME>: The node name.
- <OUTPUT>: The output format. `json` or `yaml`. Default is `json`.

Delete Node API

```
DELETE /cluster/<NODE_NAME>?output=<OUTPUT>
```

- <NODE_NAME>: The node name.
- <OUTPUT>: The output format. `json` or `yaml`. Default is `json`.

4.7.5 Snapshot APIs

Get Snapshot API

```
GET /snapshot
```

Create Snapshot API

```
PUT /snapshot?output=<OUTPUT>
```

- <OUTPUT>: The output format. `json` or `yaml`. Default is `json`.

4.8 Wipepedia example

This section discusses how to index Wikipedia dump.

4.8.1 Downloading Wikipedia dump

```
$ curl -o ~/tmp/enwiki-20190101-pages-articles.xml.bz2 https://dumps.wikimedia.org/
↪enwiki/20190101/enwiki-20190101-pages-articles.xml.bz2
```

4.8.2 Installing Wikiextractor

```
$ git clone git@github.com:attardi/wikiextractor.git
$ cd wikiextractor
```

4.8.3 Extracting Wikipedia data

```
$ ./WikiExtractor.py --output ~/tmp/enwiki --bytes 200K --json ~/tmp/enwiki-20190101-
↪pages-articles.xml.bz2
```

4.8.4 Starting Cockatrice

```
$ cockatrice start indexer
```

4.8.5 Creating index

```
$ curl -s -X GET https://raw.githubusercontent.com/mosuka/cockatrice/master/example/
↪enwiki_schema.yaml | xargs -0 cockatrice create index enwiki
```

4.8.6 Indexing Wikipedia

```
$ for FILE in $(find ./tmp/enwiki -type f -name '*' | sort)
  do
    echo ${FILE}
    cat ${FILE} | jq  . | jq -s '.' | xargs -0 cockatrice put documents enwiki
  done
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search