Rally (esrally): Elasticsearch benchmarking and stress testing tool investigation

1. Introduction

ElasticSearch (es) officially uses Rally (known as esrally) to test its performance. esrally is a Python3-based open-source stress testing tool, that provides a large variety of benchmarking metrics for es nodes. It is a macro-benchmarking tool, which treats elasticsearch node/cluster as a black-box system and test it from user's perspective, rather than a unit testing or integration testing tool that are from code's perspective. It can help you with the following tasks:

- Setup and teardown of an Elasticsearch cluster for benchmarking
- Customize test dataset and operations based on existing indicies
- Management of benchmark data and specifications even across Elasticsearch versions
- Running benchmarks and recording results
- Finding performance problems by attaching so-called telemetry devices
- Comparing performance results

Terms explanation

Terms	
pipeline	the process of a stress testing. There are 4 available pipelines: `from-sources-complete`, `from-sources-skip-build`, `from-distribution`, `benchmark-only`. Check with `$ esrally list pipelines`
track	dataset + test policies: json files that include dataset and test plans. esrally provides 10 example tracks, size ranging from 104.9MB to 109.2GB, format ranging from structured data to unstructured data. Check with `$ esrally list tracks`. Tracks will be automatically downloaded when running a race by specifying track name in the command line as a parameter (default is the *geonames* track). Users can also create their own tracks using the script provided by esrally. Refer to *3.4 Customized track test* or this link for creating custom tracks.
distribution	an elasticsearch instance downloaded from elasticsearch repo
race	one benchmark test. Check with `$ esrally list races`
car / team	es instances with different heap configurations. Check with `$ esrally list cars`
tournament	compare two race results. Check with `$ esrally compare --baseline=[race1 Id] --contender=[race2 Id]`

2. Install and configure esrally

To get started, some prerequisites are required `Python3.8+`, `pip3`, `JDK11` and `JDK15`, and `git1.9+`. Refer to this link https://esrally.readthedocs.io/en/stable/install.html for prerequisites installation.

esrally Installation

`pip3 install esrally` (add `--user` at the end if having *permission denied* errors). Current latest version is `2.2.1`, stable version is `2.2.0`.

```
$ pip3 install esrally
```

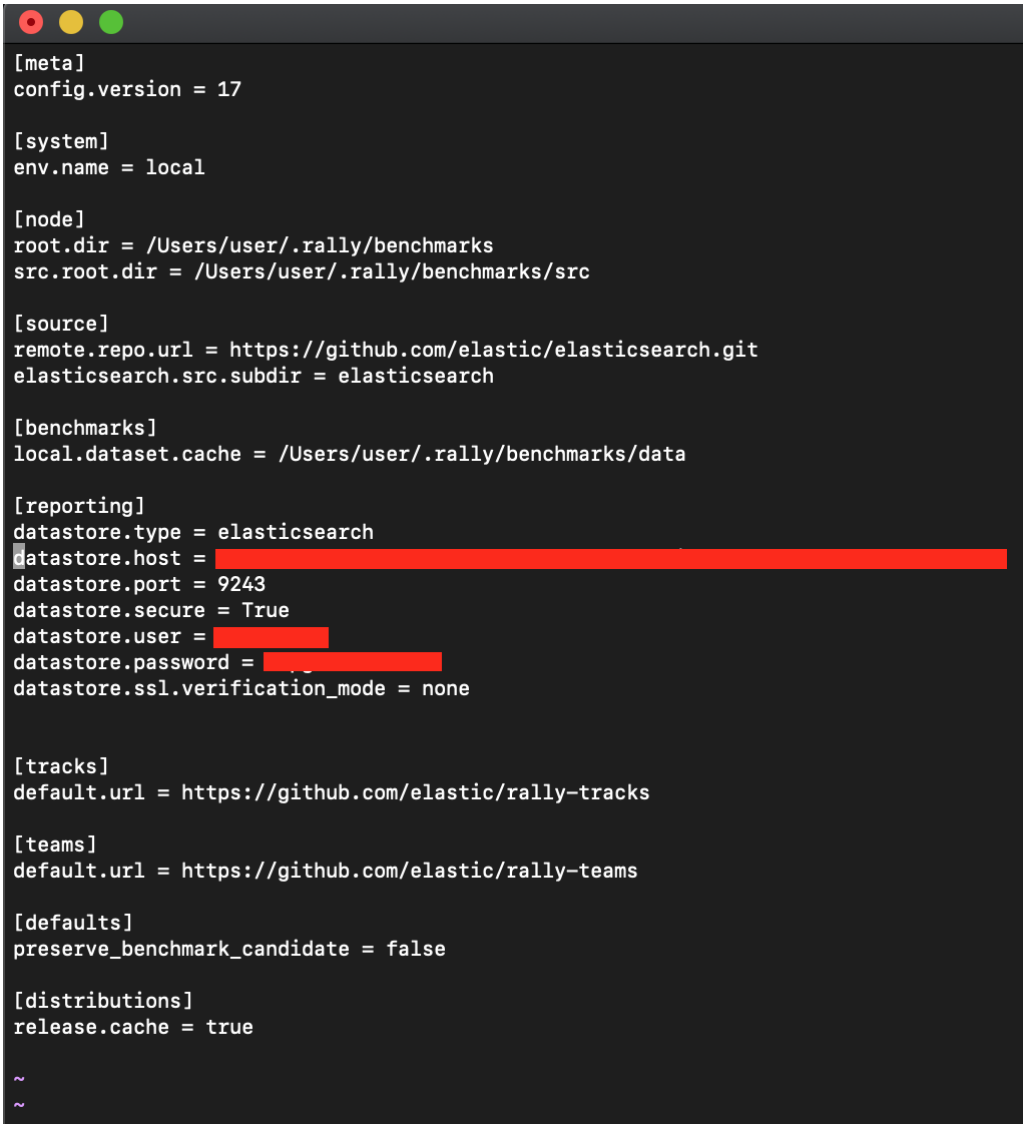
Configuration for reporting options

In our case, we want to report all test results back to elasticsearch cluster for further analysis in Kibana. Can run `esrally configure --advanced-config` to do a step-by-step advanced configuration: make changes to root directory, project directory, elasticsearch repository url, reporting mode, git repository, etc.

Or, it is easier to edit the rally.ini file for advanced configuration.

```
$ vim ~/.rally/rally.ini
```

```
[reporting]
datastore.type = elasticsearch
datastore.host = bd696a4c24ef4f43b0ea52f396da54e7.elastic-
nonprod.****.****.***.**
datastore.port = 9243
datastore.secure = True
datastore.user = ****
datastore.password = ****
datastore.ssl.verification_mode = none
```



Note: change datastore.type from *in-memory* to *elasticsearch* followed by other details will point test results to elasticsearch cluster, instead of keeping in memory of the local machine.

3. Use esrally

3.1 Explore esrally using sample tracks

The most important folder is `~/.rally/benchmarks`. Track data and their configuration files, cars data, races data will be downloaded or saved to the subfolders ot this folder. Try to allow enough space (for example, >5GB if running the default `geonames` track), otherwise, or if having *No Space on Disk* errors, consider run --advanced-config to point to another directory.

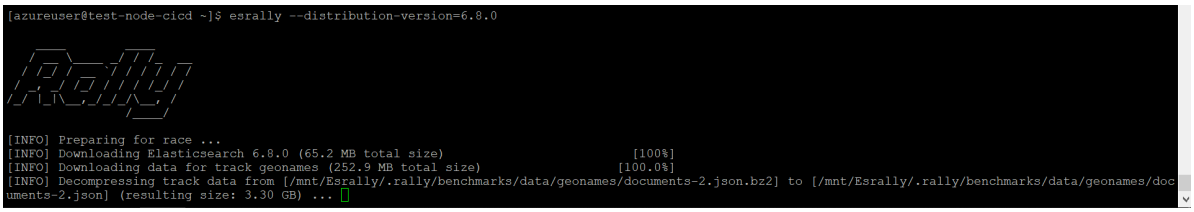
The most important file is `~/.rally/rally.ini`, which stores all environment and usage configurations of the esrally tool.

Start esrally

Now we can start the first race (benchmarking test) by running command (*Time Warning*: about 1 hour if running like this):

```
$ esrally race --distribution-version=6.8.0 --track=geonames
```

Once started, should look similar to:



This is an online test, which means esrally will first download an elasticsearch instance (6.8.0 in this example) and complete all test steps.

esrally **CANNOT** be run/started using `root`, have to be run with user account.

For testing purposes, add `--test-mode` flag to use the provided 1K size test dataset instead of the full size dataset.

Can also do an offline test. Download the tracks first following this link (usually will take a long time), unzip to directory `~/.rally/benchmarks/data`, then specify with the parameters `--offline`. Offline mode can also be used for customized tracks.

```
$ esrally race --distribution-version=7.11.0 --track=geonames --test-mode --offline
```

Sample tracks

Sample tracks will be automatically downloaded when running rally race with pipeline from-distribution. the gitlab repo will not provide the link for downloading. Instead, the repo provides a create_a_track sample dataset for the purpose of creating our own track (please refer to *3.4 Customized track* test).

View available tracks provided by esrally:

```
$ esrally list tracks
```

Track	Compressed size	Decompressed size	Number of document	Data type
geonames	259 MB	3.3 GB	11396505	structured data
geopoints	482 MB	2.3 GB	60844404	geo queries
http_logs	1.2 GB	31 GB	247249096	web server logs
nested	663 MB	3.3 GB	11203029	nested documents
noaa	947 MB	9 GB	33659481	range fields
nyc_taxis	4.5 GB	74 GB	165346692	highly structured data
percolator	103 KB	105 MB	2000000	percolation queries
pmc	5.5 GB	22 GB	574199	full text search

For these provided sample tracks, one track consists of 5 json files:

- ~/.rally/benchmarks/data//documents.json (datasets are downloaded here)
- ~/.rally/benchmarks/tracks//track.json (define datasets, and invoke test details and test plans)
- ~/.rally/benchmarks/tracks//index.json (define data model)
- ~/.rally/benchmarks/tracks//operations/default.json (define test details - detail operations)
- ~/.rally/benchmarks/tracks//challenges/default.json (define test plans - the combination and sequence of detail operations)

Here is a sample track.json example:

```
{
  "short-description": "Http_logs benchmark",
  "description": "This benchmark indexes HTTP server log data from the 1998
world cup.",
  "data-url":
"http://benchmarks.elasticsearch.org.s3.amazonaws.com/corpora/logging",
  "indices": [
    {
      "name": "logs-181998",
      "types": [
        {
          "name": "type",
          "mapping": "mappings.json",
          "documents": "documents-181998.json.bz2",
          "document-count": 2708746,
          "compressed-bytes": 13815456,
          "uncompressed-bytes": 363512754
        }
      ]
    },
    {
      "name": "logs-191998",
      "types": [
        {
          "name": "type",
          "mapping": "mappings.json",
          "documents": "documents-191998.json.bz2",
          "document-count": 9697882,
          "compressed-bytes": 49439633,
          "uncompressed-bytes": 1301732149
        }
      ]
    }
  ],
  "operations": [
    {{ rally.collect(parts="operations/*.json") }}
  ],
  "challenges": [
    {{ rally.collect(parts="challenges/*.json") }}
  ]
}
```

This json file consists of below components：

- description and short-description: track description

- data-url: url address, indicate the root path of where to download dataset. Concatenate with "documents" in "indices" (see below) to get the full download address
- indices: define the indices that this track can work on, including create, update, delete etc. Refer to this link
- operations: define the detail operations, e.g index, force-merge, segment, search, etcRefer to this link
- challenges: define the process of a benchmarking by combining operations Refer to this link

Note that the bottom 2 sections of the track file, "operations" and "challenges" are pointing to another location, which are "operations/default.json" and "challenges/default.json". It is also OK not to point out, but write operations and challenges inline inside track.json file. See the 2 examples in 3.3 Customized track.

3.2 esrally parameters

Some commonly used parameters that suit our use case - benchmark an existing elasticsearch cluster using customized track:

Parameters	
`--pipeline`	`from-sources-complete`: Compile es source code and run es, do benchma `from-sources-skip-build`: Skip es compiling and run es, do benchmarking `distribution`: Download es distribution version, do benchmarking and ge `only`: Run on existing elasticsearch instance, not downloading any elastics benchmarking and generate report
`--target-hosts`	Specify running elasticsearch node location, e.g `--target-hosts=<IP addre` connect to the dev elasticsearch cluster, use `--target-hosts=https://bd696a4c24ef4f43b0ea52f396da54e7.elastic-nonprod.*`
`--client-options`	customize Rally's internal Elasticsearch client, accepts a list of comma-sepa example, HTTP compression, basic authorization. If X-Pack Security installe specify TLS/SSL certificate verification options in here.

Parameters	
`--track`	If the track folder is under benchmarks/tracks/default/ path, directly specif test on its dataset with pre-defined operations and/or challenges, which ca operations/default.json and challenges/default.json files
`--track-path`	If the tracks are customized, or downloaded, that are not under benchmar track-path=/path/to/your/track (Note: --track-path and --track cannot be us
`--user-tag`	key-value pairs in double quotes, e.g. --user-tag="version:7.11.0". Suggest t makes it easier to find it afterwards from `esrally list races` results
`--offline`	offline test mode without downloading datasets if tracks are already down enviornment with no Internet connection
`--report-format`	Specify the output format for the command line report: `markdown` or `csv`
`--report-file`	name the report, e.g `--report-file=results.md`, which will be saved unde

Here is an example of a full race command used to benchmark the dev elasticsearch cluster, which used a customized track by speicifying the track path:

```
$ esrally race --pipeline=benchmark-only --target-
hosts=https://bd696a4c24ef4f43b0ea52f396da54e7.elastic-
nonprod.****.****.***.**:9243 --client-
options="use_ssl:true,verify_certs:false,basic_auth_user:'****',basic_auth_passw
ord:'****',timeout:5" --track-
path=/Users/user/.rally/benchmarks/tracks/custom_tracks/dev_cust_cst_cunm --
offline --user-tag=“version7.11.1:dev_cst_cunm”
```

For a full list of available parameters, refer to this link (command line reference).

3.3 Customized tracks

A track includes 2 parts, the document dataset (json files) and the benchmarking scenarios (the operations). There are 2 options to create a track:

1. Create a track from scratch

   Use python script to convert a text file to document datasets, then compose the test scenario (operations) file by creating a track.json file. This option is not covered in this report. Refer to Appendix A for detailed steps.

2. Create a track from data in an existing elasticsearch cluster

   Use subcommand `create-track` with parameter `--indicies` to create the document dataset from that index, then compose the test scenario (operations) file by editing the provided track.json file.

Example 1

Below example created a track using the clk_bat_b sample index in the dev elasticsearch cluster

- **Create a track**

  For example, we want to create a track from elasticsearch index `stp_clk_cust_cunm_v4`,

  We can create the track using command:

  ```
  $ esrally create-track --track=dev_clk_cust_cst_cunm --target-
  hosts=https://bd696a4c24ef4f43b0ea52f396da54e7.elastic-nonprod...***.**:9243
  --client-
  options="use_ssl:true,verify_certs:false,basic_auth_user:'****',basic_auth_p
  assword:'****', timeout:5" --indices=dev_clk_cust_cst_cunm --output-
  path=~/.rally/benchmarks/tracks/custom_tracks
  ```

  Note: use `--output-path` to specify where to put the customized track. Rally will create a folder with the index name to be the folder name (dev_clk_cust_cst_cunm in this example). The output folder includes the datasets (`*documents\*.json`, `*.bz2` and `*.offset`), 1 mapping file and 1 track definition file `track.json`.

  ```
  user@Songs-MBP .rally % ls -l ~/.rally/benchmarks/tracks/custom_tracks/dev_clk_cust_cst_cunm
  total 54808
  -rw-r--r--  1 user  staff    249627 21 Jun 14:57 dev_clk_cust_cst_cunm-documents-1k.json
  -rw-r--r--  1 user  staff     22007 21 Jun 14:57 dev_clk_cust_cst_cunm-documents-1k.json.bz2
  -rw-r--r--  1 user  staff  25034896 21 Jun 14:58 dev_clk_cust_cst_cunm-documents.json
  -rw-r--r--  1 user  staff   1937363 21 Jun 14:58 dev_clk_cust_cst_cunm-documents.json.bz2
  -rw-r--r--  1 user  staff      6218 21 Jun 14:57 dev_clk_cust_cst_cunm.json
  -rw-r--r--  1 user  staff      1396 21 Jun 14:58 track.json
  ```

  Refer to this link for details of creating a track from data in an existing cluster.

- **Edit track.json** to define operations

  For this example, suppose we are intetested in running 5 queries: `match_all, match_phrase, terms, range, aggregation` on this trac, which are listed below:

They can be defined into track.json as below:

```
# open track.json file
$ vim
~/.rally/benchmarks/tracks/custom_tracks/dev_clk_cust_cst_cunm/track.json
```

```
{% import "rally.helpers" as rally with context %}
{
  "version": 2,
  "description": "Tracker-generated track for dev_clk_cust_cst_cunm",
  "indices": [
    {
      "name": "dev_clk_cust_cst_cunm",
      "body": "dev_clk_cust_cst_cunm.json"
    }
  ],
  "corpora": [
```

```
    {
      "name": "dev_clk_cust_cst_cunm",
      "documents": [
        {
          "target-index": "dev_clk_cust_cst_cunm",
          "source-file": "dev_clk_cust_cst_cunm-documents.json.bz2",
          "document-count": 100374,
          "compressed-bytes": 1937363,
          "uncompressed-bytes": 25034896
        }
      ]
    }
  ],
  "schedule": [
    {
      "operation": {
      "name": "query-match-all",
      "operation-type": "search",
      "body": {
        "query": {
          "match_all": {}
          }
        }
      },
      "clients": 4,
      "warmup-time-period": 20,
      "time-period": 140
    },
    {
      "operation":{
        "name": "match_phrase_FirstName",
        "operation-type": "search",
        "body": {
          "size": 0,
          "query": {
            "match_phrase": {
              "FIRST_NAME": "Chris"
            }
          }
        }
      }
    },
    {
      "operation": {
        "name": "terms_TITLE",
        "operation-type": "search",
        "body": {
          "query": {
            "terms": {
              "TITLE": ["Miss","Mrs.","Ms."]
            }
          }
        }
      },
      "clients": 4,
      "warmup-time-period": 20,
      "time-period": 140,
      "target-throughput": 400
```

```
    },
    {
      "operation": {
        "name": "range",
        "operation-type": "search",
        "body": {
          "query": {
            "range": {
              "DOB": {
                "gte": 19290101,
                "lte": 19991231
              }
            }
          }
        }
      }
    },
    {
      "operation": {
        "name": "aggregation",
        "operation-type": "search",
         "body": {
           "size": 0,
             "aggs": {
               "agg_by_gender": {
                 "terms": {
                   "field": "SEX_CODE"
                 }
               }
             }
           }
        }
      }
  ]
}
```

Write the 5 queries into each `body` element, together with name and operation-type, as 5 operations directly under the `scedule` element. Here `Match_all` and `terms` are defined with 4 clients, other 3 operations did not define clients, which will be 1 client by default.

Some useful task properties inside `schedule` element to control the test operations

- `clients` (optional, defaults to 1): The number of clients that should execute a task concurrently
- `warmup-time-period` (optional, defaults to 0): A time period in seconds that Rally considers for warmup of the benchmark candidate. All response data captured during warmup will not show up in the measurement results.
- `time-period` (optional): A time period in seconds that Rally considers for measurement. Note that for bulk indexing you should usually not define this time period
- `target-throughput` (optional): Defines the benchmark mode. If it is not defined, Rally assumes this is a throughput benchmark and will run the task as fast as it can. This is mostly needed for batch-style operations where it is more important to achieve the best throughput instead of an acceptable latency. If it is defined, it specifies the number of requests per second over all clients. E.g. if you specify `target-throughput: 1000` with 8 clients, it means that each client will issue 125 (= 1000 / 8) requests per second. In total,

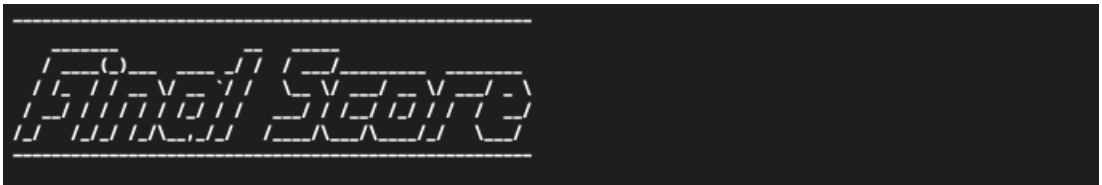
all clients will issue 1000 requests each second. If Rally reports less than the specified throughput then Elasticsearch simply cannot reach it
    - `warmup-iterations` (optional, defaults to 0): Number of iterations that each client should execute to warmup the benchmark candidate. Warmup iterations will not show up in the measurement results
    - `iterations` (optional, defaults to 1): Number of measurement iterations that each client executes. The command line report will automatically adjust the percentile numbers based on this number (i.e. if you just run 5 iterations you will not get a 99.9th percentile because we need at least 1000 iterations to determine this value precisely).

  All tasks in the `schedule` list are executed sequentially in the order in which they have been defined. However, it is also possible to execute multiple tasks concurrently, by wrapping them in a `parallel` element, as shown in Example 2.

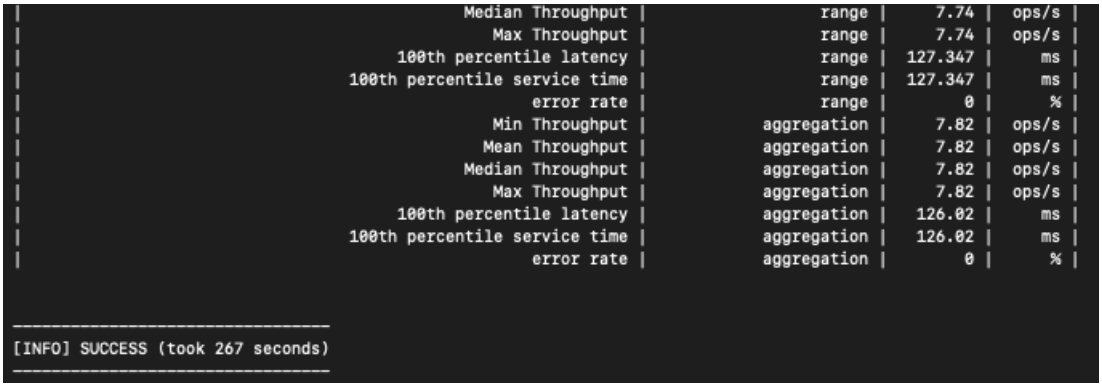
- Run

```
$ esrally race --pipeline=benchmark-only --target-
hosts=https://bd696a4c24ef4f43b0ea52f396da54e7.elastic-
nonprod.****.****.***.**:9243 --client-
options="use_ssl:true,verify_certs:false,basic_auth_user:'****',basic_auth_p
assword:'****',timeout:5" --track-
path=~/.rally/benchmarks/tracks/custom_tracks/dev_clk_cust_cst_cunm --
offline --user-tag="parallel:4+time140"
```

- Result



Metric	Task	Value	Unit
Cumulative indexing time of primary shards		20.334	min
Min cumulative indexing time across primary shards		0	min
Median cumulative indexing time across primary shards		0	min
Max cumulative indexing time across primary shards		9.5383	min
Cumulative indexing throttle time of primary shards		0	min
Min cumulative indexing throttle time across primary shards		0	min
Median cumulative indexing throttle time across primary shards		0	min
Max cumulative indexing throttle time across primary shards		0	min
Cumulative merge time of primary shards		11.5707	min
Cumulative merge count of primary shards		1192	
Min cumulative merge time across primary shards		0	min
Median cumulative merge time across primary shards		0	min
Max cumulative merge time across primary shards		5.14323	min
Cumulative merge throttle time of primary shards		0.60465	min
Min cumulative merge throttle time across primary shards		0	min
Median cumulative merge throttle time across primary shards		0	min
Max cumulative merge throttle time across primary shards		0.360167	min
Cumulative refresh time of primary shards		5.02117	min
Cumulative refresh count of primary shards		13279	
Min cumulative refresh time across primary shards		0	min
Median cumulative refresh time across primary shards		0	min
Max cumulative refresh time across primary shards		2.72852	min
Cumulative flush time of primary shards		2.93543	min
Cumulative flush count of primary shards		1405	
Min cumulative flush time across primary shards		0	min
Median cumulative flush time across primary shards		0	min
Max cumulative flush time across primary shards		0.81405	min
Total Young Gen GC time		0.396	s
Total Young Gen GC count		17	
Total Old Gen GC time		0	s
Total Old Gen GC count		0	
Store size		39.7094	GB
Translog size		0.487049	GB
Heap used for segments		9.18917	MB
Heap used for doc values		0.473284	MB
Heap used for terms		4.89389	MB
Heap used for norms		0.640869	MB
Heap used for points		0	MB
Heap used for stored fields		3.18113	MB
Segment count		1035	
Min Throughput	query-match-all	57.96	ops/s
Mean Throughput	query-match-all	58.59	ops/s
Median Throughput	query-match-all	58.63	ops/s
Max Throughput	query-match-all	59.45	ops/s
50th percentile latency	query-match-all	85193.2	ms
90th percentile latency	query-match-all	141270	ms
99th percentile latency	query-match-all	153956	ms
99.9th percentile latency	query-match-all	155159	ms
99.99th percentile latency	query-match-all	155361	ms
100th percentile latency	query-match-all	155377	ms
50th percentile service time	query-match-all	32.3865	ms
90th percentile service time	query-match-all	35.745	ms
99th percentile service time	query-match-all	59.6649	ms
99.9th percentile service time	query-match-all	95.4121	ms
99.99th percentile service time	query-match-all	194.503	ms
100th percentile service time	query-match-all	257.6	ms
error rate	query-match-all	0	%
Min Throughput	match_phrase_FirstName	7.41	ops/s
Mean Throughput	match_phrase_FirstName	7.41	ops/s
Median Throughput	match_phrase_FirstName	7.41	ops/s
Max Throughput	match_phrase_FirstName	7.41	ops/s
100th percentile latency	match_phrase_FirstName	133.072	ms
100th percentile service time	match_phrase_FirstName	133.072	ms
error rate	match_phrase_FirstName	0	%
Min Throughput	terms_TITLE	54.62	ops/s
Mean Throughput	terms_TITLE	55.22	ops/s
Median Throughput	terms_TITLE	55.29	ops/s
Max Throughput	terms_TITLE	55.77	ops/s
50th percentile latency	terms_TITLE	34.2227	ms
90th percentile latency	terms_TITLE	38.4738	ms
99th percentile latency	terms_TITLE	62.7456	ms
99.9th percentile latency	terms_TITLE	108.146	ms
99.99th percentile latency	terms_TITLE	140.311	ms
100th percentile latency	terms_TITLE	157.119	ms
50th percentile service time	terms_TITLE	34.2227	ms
90th percentile service time	terms_TITLE	38.4738	ms
99th percentile service time	terms_TITLE	62.7456	ms
99.9th percentile service time	terms_TITLE	108.146	ms
99.99th percentile service time	terms_TITLE	140.311	ms
100th percentile service time	terms_TITLE	157.119	ms
error rate	terms_TITLE	0	%
Min Throughput	range	7.74	ops/s
Mean Throughput	range	7.74	ops/s



Example 2

- Create track from index `stp_clk_cust_cunm_v4` by running below command:

```
$ esrally create-track --track=stp_clk_cust_cunm_v4 --target-
hosts=https://bd696a4c24ef4f43b0ea52f396da54e7.elastic-nonprod...***.**:9243
--client-
options="use_ssl:true,verify_certs:false,basic_auth_user:'****',basic_auth_p
assword:'****', timeout:5" --indices=stp_clk_cust_cunm_v4 --output-
path=~/.rally/benchmarks/tracks/custom_tracks
```

- Edit track.json to add below query:

```
{"size":500,"query":{"bool":{"must":[{"match_phrase":{"FIRST_NAME.clean":
{"query":"STANLEY","slop":0,"zero_terms_query":"NONE","boost":1.0}}},
{"match_phrase":{"SURNAME.clean":
{"query":"SMITH","slop":0,"zero_terms_query":"NONE","boost":1.0}}},
{"match_phrase":{"DOB":{"query":"1945-07-
05","slop":0,"zero_terms_query":"NONE","boost":1.0}}},{"term":{"CUMI":
{"value":"0","boost":1.0}}},{"term":{"CST":{"value":"0","boost":1.0}}},
{"term":{"CUDL":{"value":"0","boost":1.0}}},{"term":{"LD":
{"value":"0","boost":1.0}}},{"term":{"LNK":
{"value":"0","boost":1.0}}}],"adjust_pure_negative":true,"boost":1.0}}}
```

  Open track.json,

```
# open track.json file
$ vim
~/.rally/benchmarks/tracks/custom_tracks/stp_clk_cust_cunm_v4/track.json
```

  Replace the operation section with the given query like below. This time, the entire query is wrapped inside a `parallel tasks` section, to stay align with the actual dev elasticsearch environment.

```
{% import "rally.helpers" as rally with context %}
{
  "version": 2,
  "description": "Tracker-generated track for stp_clk_cust_cunm_v4",
  "indices": [
    {
      "name": "stp_clk_cust_cunm_v4",
      "body": "stp_clk_cust_cunm_v4.json"
```

```
    }
  ],
  "corpora": [
    {
      "name": "stp_clk_cust_cunm_v4",
      "documents": [
        {
          "target-index": "stp_clk_cust_cunm_v4",
          "source-file": "stp_clk_cust_cunm_v4-documents.json.bz2",
          "document-count": 1539010,
          "compressed-bytes": 25981065,
          "uncompressed-bytes": 434519237
        }
      ]
    }
  ],
  "schedule": [
    {
        "parallel": {
          "tasks": [
            {
              "operation": {
                "name": "query_bool_must",
                "operation-type": "search",
                "body": {
                  "query": {
                    "bool": {
                        "adjust_pure_negative": true,
                        "boost": 1.0,
                        "must": [
                          {
                            "match_phrase": {
                                "FIRST_NAME.clean": {
                                    "boost": 1.0,
                                    "query": "STANLEY",
                                    "slop": 0,
                                    "zero_terms_query": "NONE"
                                }
                            }
                        },
                          {
                            "match_phrase": {
                                "SURNAME.clean": {
                                    "boost": 1.0,
                                    "query": "SMITH",
                                    "slop": 0,
                                    "zero_terms_query": "NONE"
                                }
                            }
                        },
                          {
                            "match_phrase": {
                                "DOB": {
                                    "boost": 1.0,
                                    "query": "1945-07-05",
                                    "slop": 0,
                                    "zero_terms_query": "NONE"
                                }
```

```
                            }
                        },
                          {
                            "term": {
                                "CUMI": {
                                    "boost": 1.0,
                                    "value": "0"
                                }
                            }
                        },
                          {
                            "term": {
                                "CST": {
                                    "boost": 1.0,
                                    "value": "0"
                                }
                            }
                        },
                          {
                            "term": {
                                "CUDL": {
                                    "boost": 1.0,
                                    "value": "0"
                                }
                            }
                        },
                          {
                            "term": {
                                "LD": {
                                    "boost": 1.0,
                                    "value": "0"
                                }
                            }
                        },
                          {
                            "term": {
                                "LNK": {
                                    "boost": 1.0,
                                    "value": "0"
                                }
                            }
                        }
                        ]
                    }
                },
                "size": 500
                }
            },
            "clients": 2,
            "time-period": 200
        }
      ]
    }

  }
 ]
}
```

- Start the benchmarking by running command:

```
$ esrally race --pipeline=benchmark-only --target-
hosts=https://bd696a4c24ef4f43b0ea52f396da54e7.elastic-
nonprod.****.****.***.**:9243 --client-
options="use_ssl:true,verify_certs:false,basic_auth_user:'****',basic_auth_p
assword:'****',timeout:5" --track-
path=~/.rally/benchmarks/tracks/custom_tracks/stp_clk_cust_cunm_v4 --offline
--user-tag="parallel:2+time200"
```

- Results

```
------------------------------------------------------
    ______               __  _____
   / ____(_)___  ____ _/ /  / ___/_________ _________
  / /_  / / __ \/ __ `/ /   \__ \/ ___/ __ \/ ___/ _ \
 / __/ / / / / / /_/ / /   ___/ / /__/ /_/ / /  /  __/
/_/   /_/_/ /_/\__,_/_/   /____/\___/\____/_/   \___/
------------------------------------------------------

|                                                         Metric |            Task |    Value |   Unit |
|---------------------------------------------------------------:|----------------:|---------:|-------:|
|                     Cumulative indexing time of primary shards |                 |  20.4001 |    min |
|             Min cumulative indexing time across primary shards |                 |        0 |    min |
|          Median cumulative indexing time across primary shards |                 |        0 |    min |
|             Max cumulative indexing time across primary shards |                 |  9.59547 |    min |
|            Cumulative indexing throttle time of primary shards |                 |        0 |    min |
|    Min cumulative indexing throttle time across primary shards |                 |        0 |    min |
| Median cumulative indexing throttle time across primary shards |                 |        0 |    min |
|    Max cumulative indexing throttle time across primary shards |                 |        0 |    min |
|                        Cumulative merge time of primary shards |                 |   11.592 |    min |
|                      Cumulative merge count of primary shards |                 |     1221 |        |
|                 Min cumulative merge time across primary shards |                 |        0 |    min |
|              Median cumulative merge time across primary shards |                 |        0 |    min |
|                 Max cumulative merge time across primary shards |                 |  5.15592 |    min |
|               Cumulative merge throttle time of primary shards |                 |  0.60465 |    min |
|       Min cumulative merge throttle time across primary shards |                 |        0 |    min |
|    Median cumulative merge throttle time across primary shards |                 |        0 |    min |
|       Max cumulative merge throttle time across primary shards |                 | 0.360167 |    min |
|                      Cumulative refresh time of primary shards |                 |  5.06757 |    min |
|                    Cumulative refresh count of primary shards |                 |    13628 |        |
|               Min cumulative refresh time across primary shards |                 |        0 |    min |
|            Median cumulative refresh time across primary shards |                 |        0 |    min |
|               Max cumulative refresh time across primary shards |                 |  2.72852 |    min |
|                         Cumulative flush time of primary shards |                 |  2.97147 |    min |
|                       Cumulative flush count of primary shards |                 |     1420 |        |
|                 Min cumulative flush time across primary shards |                 |        0 |    min |
|              Median cumulative flush time across primary shards |                 |        0 |    min |
|                 Max cumulative flush time across primary shards |                 | 0.827533 |    min |
|                                       Total Young Gen GC time |                 |    0.012 |      s |
|                                      Total Young Gen GC count |                 |        1 |        |
|                                         Total Old Gen GC time |                 |        0 |      s |
|                                        Total Old Gen GC count |                 |        0 |        |
|                                                   Store size |                 |  39.7189 |     GB |
|                                                Translog size |                 | 0.375365 |     GB |
|                                     Heap used for segments |                 |  9.23507 |     MB |
|                                   Heap used for doc values |                 | 0.479948 |     MB |
|                                        Heap used for terms |                 |  4.92874 |     MB |
|                                        Heap used for norms |                 | 0.640076 |     MB |
|                                       Heap used for points |                 |        0 |     MB |
|                                Heap used for stored fields |                 |  3.18631 |     MB |
|                                              Segment count |                 |     1046 |        |
|                                                 Min Throughput | query_bool_must |    49.86 |  ops/s |
|                                                Mean Throughput | query_bool_must |    59.92 |  ops/s |
|                                              Median Throughput | query_bool_must |    60.54 |  ops/s |
|                                                 Max Throughput | query_bool_must |    60.82 |  ops/s |
|                                      50th percentile latency | query_bool_must |  6435.01 |     ms |
|                                      90th percentile latency | query_bool_must |    11629 |     ms |
|                                      99th percentile latency | query_bool_must |  12783.9 |     ms |
|                                    99.9th percentile latency | query_bool_must |  13009.8 |     ms |
|                                     100th percentile latency | query_bool_must |  13027.2 |     ms |
|                                 50th percentile service time | query_bool_must |   31.656 |     ms |
|                                 90th percentile service time | query_bool_must |  34.3956 |     ms |
|                                 99th percentile service time | query_bool_must |  40.1181 |     ms |
|                               99.9th percentile service time | query_bool_must |  120.598 |     ms |
|                                100th percentile service time | query_bool_must |  123.978 |     ms |
|                                            error rate | query_bool_must |        0 |      % |


--------------------------------
[INFO] SUCCESS (took 77 seconds)
--------------------------------
```

4. Report

4.1 Test results

Direct test results are printed on the screen after each run, as shown in previous examples.

Export results to a local file

Test results can be directly exported to a local file by adding parameters `--report-format=<markdown or csv>` and `--report-file=<filename.md or .csv>`. Supported formats are markdown and csv.

```
$ esrally race --distribution-version=7.11.0 --report-format=markdown --report-file=/path/to/your/report.md
```

Export to Elasticsearch and Kibana

If testing an existing Elasticsearch cluster, reports can also be exported to the cluster, and further to Kibana where provides better analytical abilities, by specifying the location in the esrally configuration file `rally.ini`

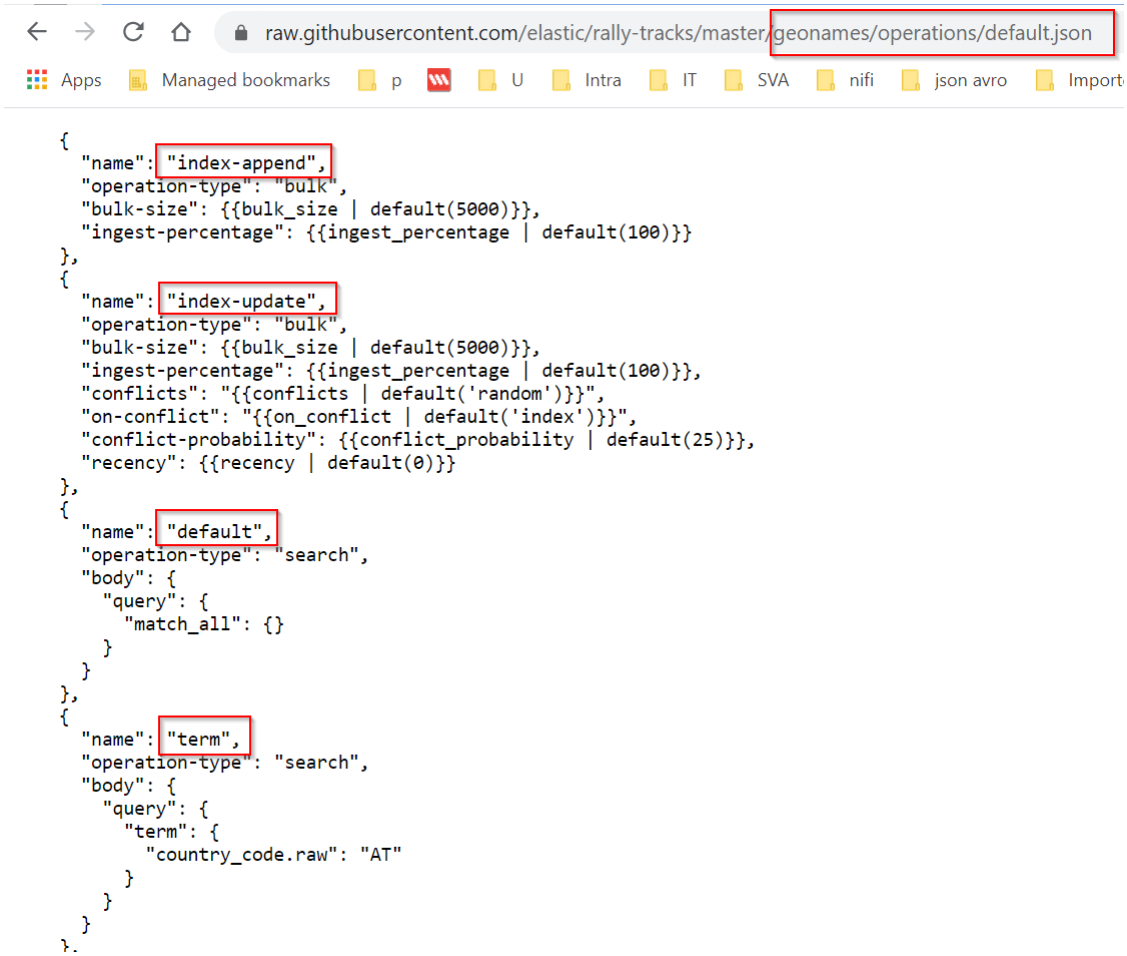
```
[reporting]
datastore.type = elasticsearch # put "elasticsearch" here instead of "in-memory"
datastore.host = https://bd696a4c24ef4f43b0ea52f396da54e7.elastic-nonprod.****.****.***.** # give the location of the es node
datastore.port = 9243 # the port number, default 9200
datastore.secure = True
datastore.user = ****
datastore.password = ****
datastore.ssl.verification_mode = none
```

Result interpretation

At the end of each race, esrally shows a summary report. Refer to this link for Summary Report.

`Task`

- operation, same contents as what're defined in (use geonames track as example) `tracks/default/geonames/operations/default.json` file

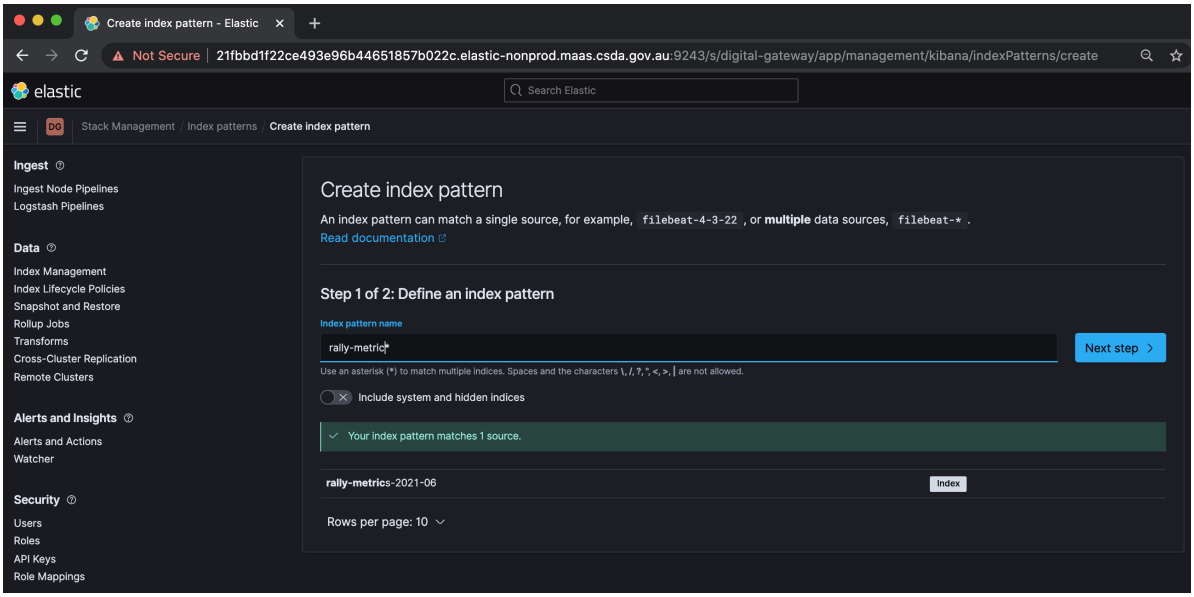


`Metric`

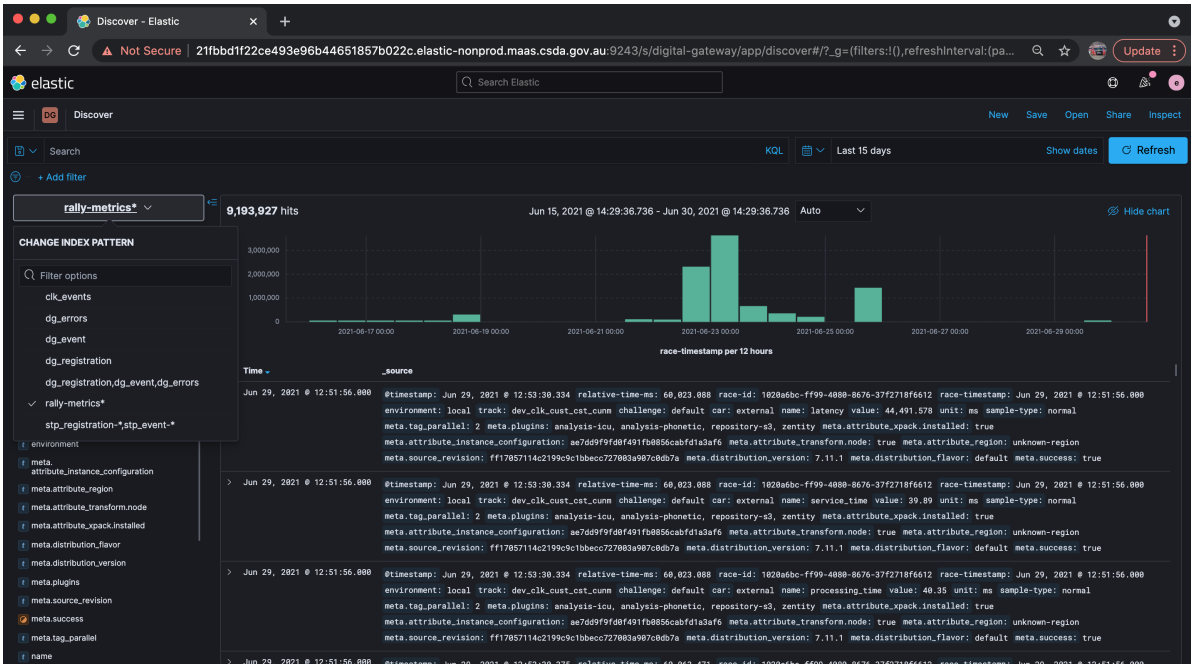
- where `Task` is blank, it is the overall summaries of corresponding metrics:
  - time: index/merge/refresh/flush - the overall time of that operation
  - GC: young/total - time consumed by GC
  - index size/total written - total index size after this test and total written volume
  - heap: segment/doc/term/norm/point/stored field - heap memory consumption
- where `Task` is not blank (has an operation), it is the calculated summaries of each operation, such as min, median, max:
  - Throughput - how many operations (queries) is handled by the tested Elasticsearch cluster per second
  - Latency - current opreation's latency (from the request reached es till request reponsed back, including queuing time)
  - Service Time - current opreation's service time (the actual time processing the request by es)

4.2 View results in Kibana

As we have configured the output of esrally tests to be sent to Elasticsearch, next we just need to create an Index Pattern by searching index name rally-metric* in Kibana.



Once created, should be able to see the incoming results in `Discover` after each run, and be able to query the index in `Dev_Tools` or create visualizations in `Visualize` and `Dashboard`.

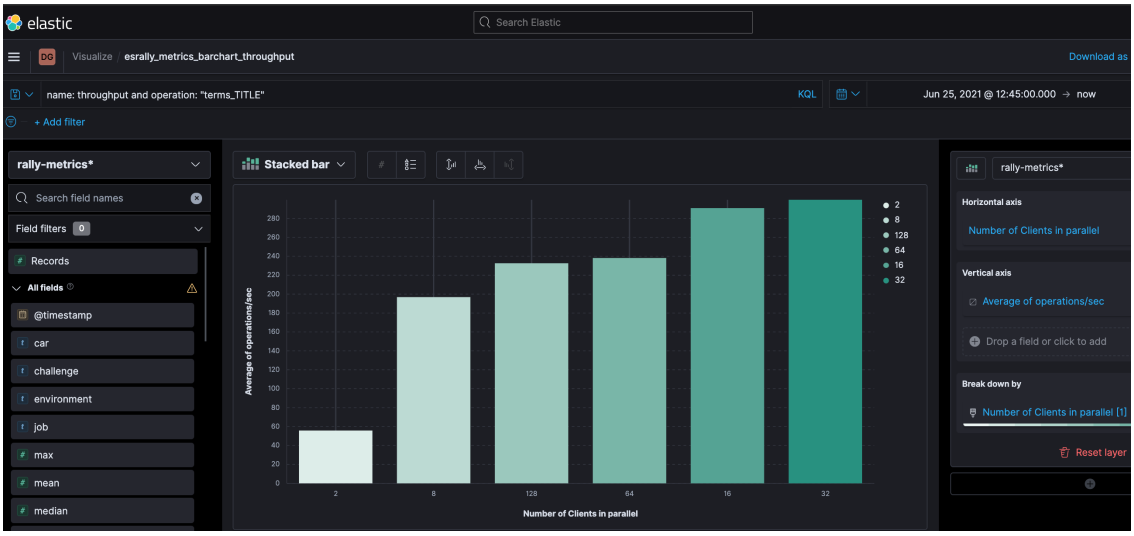


3 examples

In this case, we are more interested in the *search* performance of the Elasticsearch cluster, rather than indexing.

1. *Search* throughput 1

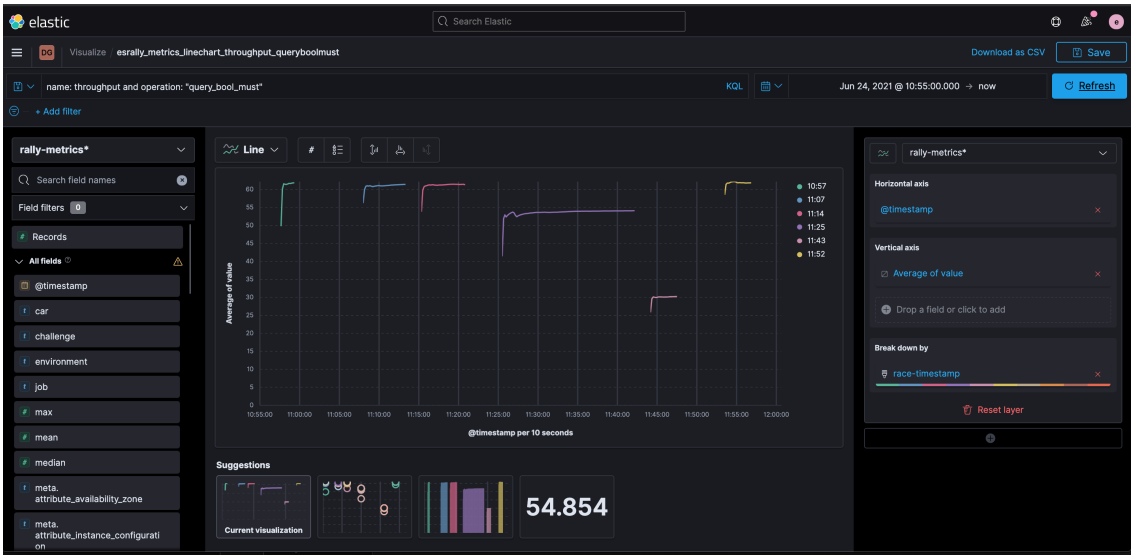
   Average throughtput of search operation across different parallel clients. Write KQL to define the metric name and operation of interest to be visualised, for example: `name: throughput and operation: "terms_TITLE"`



As parallel client number increases from 2 to 128, the highest average throughput (280 operations/second) occurred at 32 parallel clients, indicating the Elasticsearch cluster can guarantee its best search performance to up to 32 clients searching in parallel. When parallel clients increase to 64 or 128, the performance dropped.

2. *Search* throughput 2

   Change `Break down by` to Date Histogram of race-timestamp, the test history will be displayed by each race timestamp, where it is easier to compare the metrics across several races.



This example compares throughput test results across different time of the day. This can be achieved by setting the visualization to `throughput` by `@timestamp` breaking down by `race-timestamp`. This ability is useful to compare the impacts of any changes made to the test case configuration (the `track.json` file). For example, change query details, or change test duration time.

3. *Search* Latency

   Change KQL to `name: latency and not operation: "terms" and not operation: "query-match-all"`

The search latency (in milliseconds) comparasion indicates that `range` type of query has slightly higher latency than `aggregation` type of query than `match phrase` type of query, although the overall latency levels are not significantly different.

4.3 Compare two race results

Esrally can generate a comparasion report of two races. This becomes handy and useful when we want to compare two tests under different track configurations. First list out the race history, then run a tournament to compare two results.

```
# list the Race ID
$ esrally list races
```

```
[user@Songs-MBP .rally % esrally list races

    ____            ____
   / __ \___  ____ _/ / /_  __
  / /_/ / __ `/ / / / / / /
 / _, _/ /_/ / / / / /_/ /
/_/ |_|\__,_/_/_/_/\__, /
                  /____/

[WARNING] No Internet connection detected. Automatic download of track data sets etc. is disabled.

Recent races:

Race ID                              Race Timestamp    Track                Track Parameters    Challenge    Car       User Tags             Track Revision    Team Revi
ion
-----------------------------------  ----------------  -------------------  ------------------  -----------  --------  --------------------  ----------------  ---------
-----
e0c69e65-75d5-4664-8d3a-30d6a56b00dd  20210625T035949Z  dev_clk_cust_cst_cunm                                  external  parallel=128
3ee14c2f-8b2d-4b7e-a0da-7de8880fb3e2  20210625T034332Z  dev_clk_cust_cst_cunm                                  external  parallel=64
504de257-43a1-4dd2-a669-d48f4f13255e  20210625T033405Z  dev_clk_cust_cst_cunm                                  external  parallel=32
a3899cff-bda2-4004-8fe0-11bd9611ea1c  20210625T032912Z  dev_clk_cust_cst_cunm                                  external  parallel=16
f3638ee3-19a7-4734-9828-d8a05466593e  20210625T032355Z  dev_clk_cust_cst_cunm                                  external  parallel=8
d91b9d23-2ce2-47d7-b5ae-cfb4ef2b3467  20210625T031840Z  dev_clk_cust_cst_cunm                                  external  parallel=2
b7e0c79d-77cc-48d3-aa64-180f40a9b45d  20210625T031319Z  stp_clk_cust_cunm_v4                                   external  parallel=128+time200
cc7370b6-2564-4a93-a7eb-4163262223f5  20210625T023429Z  stp_clk_cust_cunm_v4                                   external  parallel=2+time200
ca8da776-07f2-462a-a823-fc98c08199f6  20210624T070905Z  stp_clk_cust_cunm_v4                                   external  parallel=2+time200
7bb65d3b-aedc-4e84-9384-08d62c0ee850  20210624T064851Z  dev_clk_cust_cst_cunm                                  external  parallel=5

[INFO] SUCCESS (took 8 seconds)
```

```
# compare by Race ID
$ esrally compare --baseline=[race1 Id] --contender=[race2 Id]
```

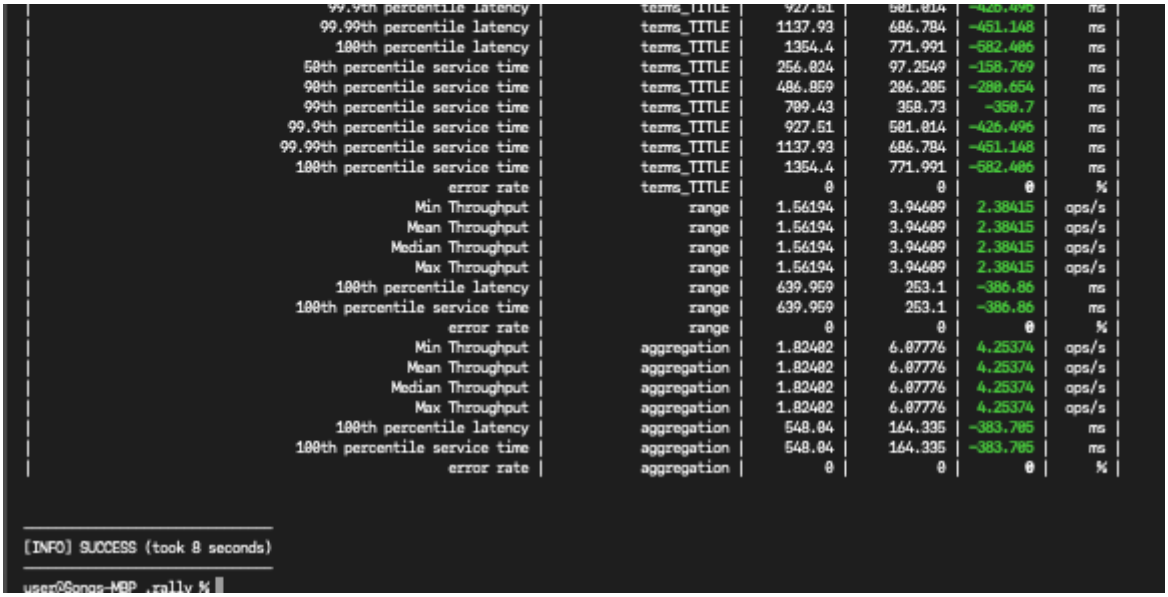
```
user@Songs-MBP .rally % esrally compare --baseline=3ee14c2f-8b2d-4b7e-a8da-7de8880fb3e2 --contender=504de257-43a1-4dd2-a669-d48f4f13255e
```

```
[WARNING] No Internet connection detected. Automatic download of track data sets etc. is disabled.

Comparing baseline
  Race ID: 3ee14c2f-8b2d-4b7e-a8da-7de8880fb3e2
  Race timestamp: 2021-06-25 03:43:32
  Car: external
  User tags: parallel=64

with contender
  Race ID: 504de257-43a1-4dd2-a669-d48f4f13255e
  Race timestamp: 2021-06-25 03:34:05
  Car: external
  User tags: parallel=32
```

Metric	Task	Baseline	Contender	Diff	Unit
Cumulative indexing time of primary shards		21.7685	21.4984	-0.27008	min
Min cumulative indexing time across primary shard		0	0	0	min
Median cumulative indexing time across primary shard		0	0	0	min
Max cumulative indexing time across primary shard		10.8761	10.6066	-0.26947	min
Cumulative indexing throttle time of primary shards		0	0	0	min
Min cumulative indexing throttle time across primary shard		0	0	0	min
Median cumulative indexing throttle time across primary shard		0	0	0	min
Max cumulative indexing throttle time across primary shard		0	0	0	min
Cumulative merge time of primary shards		12.356	12.0522	-0.30385	min
Cumulative merge count of primary shards		1580	1575	-5	
Min cumulative merge time across primary shard		0	0	0	min
Median cumulative merge time across primary shard		0	0	0	min
Max cumulative merge time across primary shard		5.76757	5.4672	-0.30037	min
Cumulative merge throttle time of primary shards		0.73575	0.60465	-0.1311	min
Min cumulative merge throttle time across primary shard		0	0	0	min
Median cumulative merge throttle time across primary shard		0	0	0	min
Max cumulative merge throttle time across primary shard		0.491267	0.360167	-0.1311	min
Cumulative refresh time of primary shards		5.64752	5.61275	-0.03477	min
Cumulative refresh count of primary shards		17510	17446	-64	
Min cumulative refresh time across primary shard		0	0	0	min
Median cumulative refresh time across primary shard		0	0	0	min
Max cumulative refresh time across primary shard		2.73273	2.73273	0	min
Cumulative flush time of primary shards		3.3688	3.36372	-0.00508	min
Cumulative flush count of primary shards		1501	1497	-4	
Min cumulative flush time across primary shard		0	0	0	min
Median cumulative flush time across primary shard		0	0	0	min
Max cumulative flush time across primary shard		0.8675	0.865883	-0.00162	min
Total Young Gen GC time		3.292	2.446	-0.846	s
Total Young Gen GC count		63	61	-2	
Total Old Gen GC time		0	0	0	s
Total Old Gen GC count		0	0	0	
Store size		39.9459	39.8663	-0.07954	GB
Translog size		0.853266	0.886335	0.03307	GB
Heap used for segments		9.14887	9.14002	-0.00885	MB
Heap used for doc values		0.464533	0.464735	0.0002	MB
Heap used for terms		4.8689	4.86115	-0.00775	MB
Heap used for norms		0.634033	0.634033	0	MB
Heap used for points		0	0	0	MB
Heap used for stored fields		3.1814	3.18091	-0.0005	MB
Segment count		1035	1034	-1	
Min Throughput	query-match-all	148.482	178.164	29.6815	ops/s
Mean Throughput	query-match-all	198.771	199.096	0.32511	ops/s
Median Throughput	query-match-all	199.618	199.672	0.05362	ops/s
Max Throughput	query-match-all	200.087	199.937	-0.14932	ops/s
50th percentile latency	query-match-all	236.762	105.12	-131.643	ms
90th percentile latency	query-match-all	550.145	269.098	-281.047	ms
99th percentile latency	query-match-all	997.012	617.056	-379.956	ms
99.9th percentile latency	query-match-all	1342.42	943.118	-399.307	ms
99.99th percentile latency	query-match-all	1590.09	1180.06	-410.033	ms
100th percentile latency	query-match-all	1736.07	1301.02	-435.059	ms
50th percentile service time	query-match-all	203.708	90.8183	-112.89	ms
90th percentile service time	query-match-all	391.352	192.718	-198.635	ms
99th percentile service time	query-match-all	603.519	310.424	-293.095	ms
99.9th percentile service time	query-match-all	813.279	463.01	-350.269	ms
99.99th percentile service time	query-match-all	1010.75	597.947	-412.799	ms
100th percentile service time	query-match-all	1025.29	872.649	-152.638	ms
error rate	query-match-all	0	0	0	%
Min Throughput	match_phrase_FirstName	1.18388	7.30523	6.12135	ops/s
Mean Throughput	match_phrase_FirstName	1.18388	7.30523	6.12135	ops/s
Median Throughput	match_phrase_FirstName	1.18388	7.30523	6.12135	ops/s
Max Throughput	match_phrase_FirstName	1.18388	7.30523	6.12135	ops/s
100th percentile latency	match_phrase_FirstName	844.486	136.697	-707.789	ms
100th percentile service time	match_phrase_FirstName	844.486	136.697	-707.789	ms
error rate	match_phrase_FirstName	0	0	0	%
Min Throughput	terms_TITLE	142.537	241.601	99.0637	ops/s
Mean Throughput	terms_TITLE	238.191	299.729	61.5378	ops/s
Median Throughput	terms_TITLE	243.134	294.801	51.6666	ops/s
Max Throughput	terms_TITLE	249.79	338.175	88.3848	ops/s
50th percentile latency	terms_TITLE	256.003	97.2638	-158.739	ms
90th percentile latency	terms_TITLE	486.847	206.201	-280.646	ms
99th percentile latency	terms_TITLE	709.434	358.75	-350.684	ms



The differences between the contender test against the baseline test are shown in the `Diff` column, with green coloured values indicating performance improvement and red items as decrese.

It is useful to add `--user-tag="your-key:your-value"`, e.g `--user-tag="parallel:2"` in the race command for eaiser identification of which Race ID we are looking for.

5. Other scenario

Provision and benchmarking a single Elasticsearch node

Apart from acting as a load-generator, esrally can also provision the Elasticsearch environment then perform the benchmarking tasks. This will be useful if the objective is to test different versions of Elasticsearch, or to test the impact of changing Elasticsearch configuration parameters without impacting the existing running cluster.

First, install an es node on local VM:

```
$ esrally install --quiet --distribution-version=6.8.0 --node-name="rally-node-0" --network-host="127.0.0.1" --http-port=39200 --master-nodes="rally-node-0" --seed-hosts="127.0.0.1:39300"
```

Make a note of the **installation-id**, which will be used later.

After the installation, start the node (use the installation-id from above step) using below command. To tie all metrics of a benchmark together, Rally needs a consistent race id across all invocations. Suggested generating a UUID `uuidgen`.

```
# generate a unique race id (use the same id from now on)
$ export RACE_ID=$(uuidgen)
$ esrally start --installation-id="93d6ef1d-7cb2-45c5-89ad-8759a01b644e" --race-id="${RACE_ID}"
```

After node started, run a benchmark:

```
$ esrally race --pipeline=benchmark-only --target-host=127.0.0.1:39200 --
track=geonames --challenge=append-no-conflicts --on-error=abort --race-
id=${RACE_ID} --test-mode
```

After benchmarking, can stop the node:

```
$ esrally stop --installation-id="93d6ef1d-7cb2-45c5-89ad-8759a01b644e"
```

6. Security control

esrally provides `client-options` to control user access and transport layer security to elasticsearch cluster.

If want to connect the elasticsearch cluster that has enabled TLS and basic authentication, need to enable basic authentication: `--client-options="basic_auth_user:'user',basic_auth_password:'password'"`. Avoid the characters `'`, `,` and `:` in user name and password as Rally's parsing of these options is currently really simple and there is no possibility to escape characters.

```
$ esrally --pipeline=benchmark-only --track=geonames --challenge=append-no-
conflicts-index-only --target-host=http://localhost:9200 --client-
options="use_ssl:true,basic_auth_user:'user',basic_auth_password:'password',veri
fy_certs:false"
```

Need to modify basic_auth_user and basic_auth_password accordingly.

7. Summary

esrally designs a complete and reproducible test process for elasticsearch based on configuration files, which significantly reduces the test complexicity. Its test results are well integrated with ELK products (Elasticsearch, Logstash, Kibana) that makes the result storage as well as analysis and visualization easy to manage.

Suggest to allow some time to setup the environment configuration. Another point that needs to draw attention is some of the provided tracks are huge and take long time to download. Download them first then run offline mode benchmarking, or customize tracks using existing data can overcome this drawback.

Appendix A

Create a track from scratch

- Sample dataset: Geonames provides sample geo data: allCountries.zip. Download create_a_track to directory `.rally/benchmarks/tracks/myTrack/` (instead of the `/mnt/Esrally/.rally/benchmarks/tracks/default/` folder), extract all files and inspect `allCountires.txt`
- Under the same directory, invoke the `toJSON.py` script, which will convert the allCountries.txt to `documents.json`, which is the testing dataset. Use below commands to check document and dataset size:

  ```
  $ wc -l documents.json
  $ stat -f "%z" documents.json
  ```

  and update them into the track.json values of:

  "document-count":
  "uncompressed-bytes":
- `track.json` and `index.json` are the configuration files of this track
- With above 3 files, documents.json, track.json, index.json, a track is now created. Refer to this link for more details. Now we can run a race on the created track by specifying its location:

  ```
  $ esrally --distribution-version=6.8.0 --track-
  path=/mnt/Esrally/.rally/benchmarks/tracks/myTrack --user-
  tag="track:customized"
  ```