Rally Documentation

Release 2.0.2

Daniel Mitterdorfer

Oct 27, 2020

Getting Started with Rally

You want to benchmark Elasticsearch? Then Rally is for you. It can help you with the following tasks:

- Setup and teardown of an Elasticsearch cluster for benchmarking
- 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

We have also put considerable effort in Rally to ensure that benchmarking data are reproducible.

CHAPTER 1

Getting Help or Contributing to Rally

- Use our Discuss forum to provide feedback or ask questions about Rally.
- See our contribution guide on guidelines for contributors.

CHAPTER 2

Source Code

Rally's source code is available on Github. You can also check the changelog and the roadmap there.

2.1 Quickstart

Rally is developed for Unix and is actively tested on Linux and MacOS. Rally supports benchmarking Elasticsearch clusters running on Windows but Rally itself needs to be installed on machines running Unix.

2.1.1 Install

Install Python 3.8+ including `pip3`, git 1.9+ and an appropriate JDK to run Elasticsearch Be sure that `JAVA_HOME` points to that JDK. Then run the following command, optionally prefixed by `sudo` if necessary:

```
pip3 install esrally
```

If you have any trouble or need more detailed instructions, look in the *detailed installation guide*.

2.1.2 Configure

Just invoke `esrally configure`.

For more detailed instructions and a detailed walkthrough see the *configuration guide*.

2.1.3 Run your first race

Now we're ready to run our first *race*:

```
esrally --distribution-version=6.5.3
```

This will download Elasticsearch 6.5.3 and run Rally's default *track* - the geonames track - against it. After the race, a *summary report* is written to the command line::

```
------------------------------------------------------

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

|   Lap |                                                          Metric |          ␣
↪         Task |     Value |    Unit |
|------:|----------------------------------------------------------------:|-----------
↪-------------:|----------:|--------:|
|   All |                      Cumulative indexing time of primary shards |          ␣
↪              |   54.5878 |     min |
|   All |              Min cumulative indexing time across primary shards |          ␣
↪              |   10.7519 |     min |
|   All |           Median cumulative indexing time across primary shards |          ␣
↪              |   10.9219 |     min |
|   All |              Max cumulative indexing time across primary shards |          ␣
↪              |   11.1754 |     min |
|   All |             Cumulative indexing throttle time of primary shards |          ␣
↪              |         0 |     min |
|   All |     Min cumulative indexing throttle time across primary shards |          ␣
↪              |         0 |     min |
|   All |  Median cumulative indexing throttle time across primary shards |          ␣
↪              |         0 |     min |
|   All |     Max cumulative indexing throttle time across primary shards |          ␣
↪              |         0 |     min |
|   All |                         Cumulative merge time of primary shards |          ␣
↪              |   20.4128 |     min |
|   All |                        Cumulative merge count of primary shards |          ␣
↪              |       136 |         |
|   All |                 Min cumulative merge time across primary shards |          ␣
↪              |   3.82548 |     min |
|   All |              Median cumulative merge time across primary shards |          ␣
↪              |    4.1088 |     min |
|   All |                 Max cumulative merge time across primary shards |          ␣
↪              |   4.38148 |     min |
|   All |                Cumulative merge throttle time of primary shards |          ␣
↪              |   1.17975 |     min |
|   All |        Min cumulative merge throttle time across primary shards |          ␣
↪              |    0.1169 |     min |
|   All |     Median cumulative merge throttle time across primary shards |          ␣
↪              |   0.26585 |     min |
|   All |        Max cumulative merge throttle time across primary shards |          ␣
↪              |  0.291033 |     min |
|   All |                       Cumulative refresh time of primary shards |          ␣
↪              |    7.0317 |     min |
|   All |                      Cumulative refresh count of primary shards |          ␣
↪              |       420 |         |
|   All |               Min cumulative refresh time across primary shards |          ␣
↪              |   1.37088 |     min |
|   All |            Median cumulative refresh time across primary shards |          ␣
↪              |    1.4076 |     min |
|   All |               Max cumulative refresh time across primary shards |          ␣
↪              |   1.43343 |     min |
```

(continues on next page)

(continued from previous page)

```
|   All |                       Cumulative flush time of primary shards |          ␣
↪             |  0.599417 |    min |
|   All |                      Cumulative flush count of primary shards |          ␣
↪             |        10 |        |
|   All |               Min cumulative flush time across primary shards |          ␣
↪             | 0.0946333 |    min |
|   All |            Median cumulative flush time across primary shards |          ␣
↪             |  0.118767 |    min |
|   All |               Max cumulative flush time across primary shards |          ␣
↪             |   0.14145 |    min |
|   All |                                              Median CPU usage |          ␣
↪             |     284.4 |      % |
|   All |                                       Total Young Gen GC time |          ␣
↪             |    12.868 |      s |
|   All |                                      Total Young Gen GC count |          ␣
↪             |        17 |        |
|   All |                                         Total Old Gen GC time |          ␣
↪             |     3.803 |      s |
|   All |                                        Total Old Gen GC count |          ␣
↪             |         2 |        |
|   All |                                                    Store size |          ␣
↪             |   3.17241 |     GB |
|   All |                                                 Translog size |          ␣
↪             |   2.62736 |     GB |
|   All |                                                    Index size |          ␣
↪             |   5.79977 |     GB |
|   All |                                                 Total written |          ␣
↪             |   22.8536 |     GB |
|   All |                                        Heap used for segments |          ␣
↪             |   18.8885 |     MB |
|   All |                                      Heap used for doc values |          ␣
↪             | 0.0322647 |     MB |
|   All |                                           Heap used for terms |          ␣
↪             |   17.7184 |     MB |
|   All |                                           Heap used for norms |          ␣
↪             | 0.0723877 |     MB |
|   All |                                          Heap used for points |          ␣
↪             |  0.277171 |     MB |
|   All |                                   Heap used for stored fields |          ␣
↪             |  0.788307 |     MB |
|   All |                                                 Segment count |          ␣
↪             |        94 |        |
|   All |                                                Min Throughput |          ␣
↪index-append |   38089.5 |  docs/s |
|   All |                                             Median Throughput |          ␣
↪index-append |   38613.9 |  docs/s |
|   All |                                                Max Throughput |          ␣
↪index-append |   40693.3 |  docs/s |
|   All |                                       50th percentile latency |          ␣
↪index-append |   803.417 |      ms |
|   All |                                       90th percentile latency |          ␣
↪index-append |    1913.7 |      ms |
|   All |                                       99th percentile latency |          ␣
↪index-append |   3591.23 |      ms |
|   All |                                     99.9th percentile latency |          ␣
↪index-append |   6176.23 |      ms |
|   All |                                      100th percentile latency |          ␣
↪index-append |   6642.97 |      ms |
```

(continues on next page)

(continued from previous page)

```
|   All |                                        50th percentile service time |           ␣
↪index-append |   803.417 |       ms |
|   All |                                        90th percentile service time |           ␣
↪index-append |    1913.7 |       ms |
|   All |                                        99th percentile service time |           ␣
↪index-append |   3591.23 |       ms |
|   All |                                      99.9th percentile service time |           ␣
↪index-append |   6176.23 |       ms |
|   All |                                       100th percentile service time |           ␣
↪index-append |   6642.97 |       ms |
|   All |                                                          error rate |           ␣
↪index-append |         0 |        % |
|   All |                                                                 ... |           ␣
↪         ... |       ... |      ... |
|   All |                                                                 ... |           ␣
↪         ... |       ... |      ... |
|   All |                                                      Min Throughput | large_
↪prohibited_terms |         2 |   ops/s |
|   All |                                                   Median Throughput | large_
↪prohibited_terms |         2 |   ops/s |
|   All |                                                      Max Throughput | large_
↪prohibited_terms |         2 |   ops/s |
|   All |                                             50th percentile latency | large_
↪prohibited_terms |   344.429 |      ms |
|   All |                                             90th percentile latency | large_
↪prohibited_terms |   353.187 |      ms |
|   All |                                             99th percentile latency | large_
↪prohibited_terms |    377.22 |      ms |
|   All |                                            100th percentile latency | large_
↪prohibited_terms |   392.918 |      ms |
|   All |                                        50th percentile service time | large_
↪prohibited_terms |   341.177 |      ms |
|   All |                                        90th percentile service time | large_
↪prohibited_terms |   349.979 |      ms |
|   All |                                        99th percentile service time | large_
↪prohibited_terms |   374.958 |      ms |
|   All |                                       100th percentile service time | large_
↪prohibited_terms |    388.62 |      ms |
|   All |                                                          error rate | large_
↪prohibited_terms |         0 |       % |


----------------------------------
[INFO] SUCCESS (took 1862 seconds)
----------------------------------
```

2.1.4 Next steps

Now you can check *how to run benchmarks*, get a better understanding how to interpret the numbers in the *summary report* or start to *create your own tracks*. Be sure to check also some *tips and tricks* to help you understand how to solve specific problems in Rally.

Also run `esrally --help` to see what options are available and keep the *command line reference* handy for more detailed explanations of each option.

2.2 Installation

This is the detailed installation guide for Rally. If you are in a hurry you can check the *quickstart guide*.

2.2.1 Hardware Requirements

Use an SSD on the load generator machine. If you run bulk-indexing benchmarks, Rally will read one or more data files from disk. Usually, you will configure multiple clients and each client reads a portion of the data file. To the disk this appears as a random access pattern where spinning disks perform poorly. To avoid an accidental bottleneck on client-side you should therefore use an SSD on each load generator machine.

2.2.2 Prerequisites

Rally does not support Windows and is only actively tested on MacOS and Linux. Install the following packages first.

Python

- Python 3.8 or better available as `python3` on the path. Verify with: `python3 --version`.
- Python3 header files (included in the Python3 development package).
- `pip3` available on the path. Verify with `pip3 --version`.

We recommend to use pyenv to manage installation of Python. For details refer to their installation instructions and **ensure that all of** pyenv's prerequisites are installed.

Once `pyenv` is installed, install a compatible Python version:

```
# Install Python
pyenv install 3.8.0

# select that version for the current user
# see https://github.com/pyenv/pyenv/blob/master/COMMANDS.md#pyenv-global for details
pyenv global 3.8.0

# Upgrade pip
python3 -m pip install --user --upgrade pip
```

git

Git is not required if **all** of the following conditions are met:

- You are using Rally only as a load generator (`--pipeline=benchmark-only`) or you are referring to Elasticsearch configurations with `--team-path`.
- You create your own tracks and refer to them with `--track-path`.

In all other cases, Rally requires `git 1.9` or better. Verify with `git --version`.

Debian / Ubuntu

```
sudo apt-get install git
```

Red Hat / CentOS / Amazon Linux

```
sudo yum install git
```

Note: If you use RHEL, install a recent version of git via the Red Hat Software Collections.

MacOS

`git` is already installed on MacOS.

pbzip2

It is strongly recommended to install `pbzip2` to speed up decompressing the corpora of Rally standard tracks. If you have created *custom tracks* using corpora compressed with `gzip` instead of `bzip2`, it's also advisable to install `pigz` to speed up the process.

Debian / Ubuntu

```
sudo apt-get install pbzip2
```

Red Hat / CentOS / Amazon Linux

`pbzip` is available via the EPEL repository.

```
sudo yum install pbzip2
```

MacOS

Install via Homebrew:

```
brew install pbzip2
```

JDK

A JDK is required on all machines where you want to launch Elasticsearch. If you use Rally just as a load generator to *benchmark remote clusters*, no JDK is required. For details on how to install a JDK check your operating system's documentation pages.

To find the JDK, Rally expects the environment variable `JAVA_HOME` to be set on all targeted machines. To have more specific control, for example when you want to benchmark across a wide range of Elasticsearch releases, you can also set `JAVAx_HOME` where `x` is the major version of a JDK (e.g. `JAVA8_HOME` would point to a JDK 8 installation). Rally will then choose the highest supported JDK per version of Elasticsearch that is available.

Note: If you have Rally download, install and benchmark a local copy of Elasticsearch (i.e., the default Rally behavior) be sure to configure the Operating System (OS) of your Rally server with the recommended kernel settings

2.2.3 Installing Rally

1. Ensure `~/.local/bin` is in your `$PATH`.
2. Install Rally: `python3 -m pip install --user esrally`.

If you get errors during installation, it is probably due to the installation of `psutil` which we use to gather system metrics like CPU utilization. Ensure that you have installed the Python development package as documented in the prerequisites section above.

2.2.4 VirtualEnv Install

You can also use Virtualenv to install Rally into an isolated Python environment without sudo.

1. Set up a new virtualenv environment in a directory with `python3 -m venv .`
2. Activate the environment with `source /path/to/virtualenv/dir/bin/activate`
3. Install Rally with `python3 -m pip install esrally`

Whenever you want to use Rally, run the activation script (step 2 above) first. When you are done, simply execute `deactivate` in the shell to exit the virtual environment.

2.2.5 Docker

Docker images of Rally can be found in DockerHub.

Please refer to *Running Rally with Docker* for detailed instructions.

2.2.6 Offline Install

If you are in a corporate environment using Linux servers that do not have any access to the Internet, you can use Rally's offline installation package. Follow these steps to install Rally:

1. Install all prerequisites as documented above.
2. Download the offline installation package for the latest release and copy it to the target machine(s).
3. Decompress the installation package with `tar -xzf esrally-dist-linux-*.tar.gz`.
4. Run the install script with `sudo ./esrally-dist-linux-*/install.sh`.

2.2.7 Next Steps

After you have installed Rally, you need to configure it. Just run `esrally configure` or follow the *configuration help page* for more guidance.

2.3 Running Rally with Docker

Rally is available as a Docker image.

2.3.1 Limitations

The following Rally functionality isn't supported when using the Docker image:

- *Distributing the load test driver* to apply load from multiple machines.
- Using other *pipelines* apart from `benchmark-only`.

2.3.2 Quickstart

You can test the Rally Docker image by first issuing a simple command to list the available tracks:

```
$ docker run elastic/rally list tracks

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

Available tracks:

Name           Description                                                           ␣
↪                                                                                    ␣
↪                          Documents    Compressed Size    Uncompressed Size    ␣
↪Default Challenge        All Challenges
-------------  -----------------------------------------------------------------------
↪-------------------------------------------------------------------------------------
↪----------------------  -----------  -----------------  -------------------  -------
↪----------------  ------------------------------------------------------------------
↪---------------------------------------------------------
geopoint       Point coordinates from PlanetOSM                                      ␣
↪                                                                                    ␣
↪                          60,844,404   481.9 MB           2.3 GB              append-
↪no-conflicts      append-no-conflicts,append-no-conflicts-index-only,append-fast-
↪with-conflicts
eventdata      This benchmark indexes HTTP access logs generated based sample logs␣
↪from the elastic.co website using the generator available in https://github.com/
↪elastic/rally-eventdata-track  20,000,000   755.1 MB           15.3 GB             ␣
↪append-no-conflicts      append-no-conflicts
nested         StackOverflow Q&A stored as nested docs                               ␣
↪                                                                                    ␣
↪                          11,203,029   663.1 MB           3.4 GB              nested-
↪search-challenge  nested-search-challenge,index-only
so             Indexing benchmark using up to questions and answers from␣
↪StackOverflow                                                                       ␣
↪                                       36,062,278   8.9 GB             33.1 GB      ␣
↪      append-no-conflicts      append-no-conflicts
geoshape       Shapes from PlanetOSM                                                 ␣
↪                                                                                    ␣
↪                          60,523,283   13.4 GB            45.4 GB             append-
↪no-conflicts      append-no-conflicts
http_logs      HTTP server log data                                                  ␣
↪                                                                                    ␣
↪                          247,249,096  1.2 GB             31.1 GB             append-
↪no-conflicts      append-no-conflicts,append-no-conflicts-index-only,append-sorted-
↪no-conflicts,append-index-only-with-ingest-pipeline,update
geonames       POIs from Geonames                                                    ␣
↪                                                                                    ␣
↪                          11,396,505   252.4 MB           3.3 GB              append-
↪no-conflicts      append-no-conflicts,append-no-conflicts-index-only,append-sorted-
↪no-conflicts,append-fast-with-conflicts
noaa           Global daily weather measurements from NOAA                           ␣
↪                                                                                    ␣
↪                          33,659,481   947.3 MB           9.0 GB              append-
↪no-conflicts      append-no-conflicts,append-no-conflicts-index-only
```

(continues on next page)

(continued from previous page)

```
percolator     Percolator benchmark based on AOL queries                                  ␣
↪                                                                                          ␣
↪                    2,000,000    102.7 kB          104.9 MB             append-
↪no-conflicts      append-no-conflicts
nyc_taxis      Taxi rides in New York in 2015                                             ␣
↪                                                                                          ␣
↪                    165,346,692  4.5 GB            74.3 GB              append-
↪no-conflicts      append-no-conflicts,append-no-conflicts-index-only,append-sorted-
↪no-conflicts-index-only,update,append-ml
geopointshape  Point coordinates from PlanetOSM indexed as geoshapes                      ␣
↪                                                                                          ␣
↪                    60,844,404   470.5 MB          2.6 GB               append-
↪no-conflicts      append-no-conflicts,append-no-conflicts-index-only,append-fast-
↪with-conflicts
metricbeat     Metricbeat data                                                            ␣
↪                                                                                          ␣
↪                    1,079,600    87.6 MB           1.2 GB               append-
↪no-conflicts      append-no-conflicts
pmc            Full text benchmark with academic papers from PMC                          ␣
↪                                                                                          ␣
↪                    574,199      5.5 GB            21.7 GB              append-
↪no-conflicts      append-no-conflicts,append-no-conflicts-index-only,append-sorted-
↪no-conflicts,append-fast-with-conflicts

-------------------------------
[INFO] SUCCESS (took 3 seconds)
-------------------------------
```

As a next step, we assume that Elasticsearch is running on `es01:9200` and is accessible from the host where you are running the Rally Docker image. Run the `nyc_taxis` track in `test-mode` using:

```
$ docker run elastic/rally --track=nyc_taxis --test-mode --pipeline=benchmark-only --
↪target-hosts=es01:9200
```

Note: We didn't need to explicitly specify `esrally` as we'd normally do in a normal CLI invocation; the entrypoint in the Docker image does this automatically.

Now you are able to use all regular *Rally commands*, bearing in mind the aforementioned *limitations*.

2.3.3 Configuration

The Docker image ships with a default configuration file under `/rally/.rally/rally.ini`. To customize Rally you can create your own `rally.ini` and bind mount it using:

```
docker run -v /home/<myuser>/custom_rally.ini:/rally/.rally/rally.ini elastic/rally ..
↪.
```

2.3.4 Persistence

It is highly recommended to use a local bind mount (or a named volume) for the directory `/rally/.rally` in the container. This will ensure you have persistence across invocations and any tracks downloaded and extracted can be reused, reducing the startup time. You need to ensure the UID is `1000` (or GID is `0` especially in OpenShift) so that Rally can write to the bind-mounted directory.

If your local bind mount doesn't contain a `rally.ini` the container will create one for you during the first run.

Example:

```
mkdir myrally
sudo chgrp 0 myrally

# First run will generate the rally.ini
docker run --rm -v $PWD/myrally:/rally/.rally elastic/rally --track=nyc_taxis --test-
↪mode --pipeline=benchmark-only --target-hosts=es01:9200

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

Running simple configuration. Run the advanced configuration with:

  esrally configure --advanced-config

* Setting up benchmark root directory in /rally/.rally/benchmarks
* Setting up benchmark source directory in /rally/.rally/benchmarks/src/elasticsearch

Configuration successfully written to /rally/.rally/rally.ini. Happy benchmarking!

More info about Rally:

* Type esrally --help
* Read the documentation at https://esrally.readthedocs.io/en/latest/
* Ask a question on the forum at https://discuss.elastic.co/tags/c/elastic-stack/
↪elasticsearch/rally

# now run our benchmark
docker run --rm -v $PWD/myrally:/rally/.rally elastic/rally --track=nyc_taxis --test-
↪mode --pipeline=benchmark-only --target-hosts=es01:9200

...

# inspect results
$ tree myrally/benchmarks/races/
myrally/benchmarks/races/
└── 1d81930a-4ebe-4640-a09b-3055174bce43
    └── race.json

1 directory, 1 file
```

In case you forgot to bind mount a directory, the Rally Docker image will create an anonymous volume for `/rally/.rally` to ensure logs and results get persisted even after the container has terminated.

For example, after executing our earlier quickstart example `docker run elastic/rally --track=nyc_taxis --test-mode --pipeline=benchmark-only --target-hosts=es01:9200`, `docker volume ls` shows a volume:

```
$ docker volume ls
DRIVER              VOLUME NAME
local               96256462c3a1f61120443e6d69d9cb0091b28a02234318bdabc52b6801972199
```

To further examine the contents we can bind mount it from another image e.g.:

```
$ docker run --rm -i -
↪v=96256462c3a1f61120443e6d69d9cb0091b28a02234318bdabc52b6801972199:/rallyvolume -ti␣
↪python:3.8.2-slim /bin/bash
root@9a7dd7b3d8df:/# cd /rallyvolume/
root@9a7dd7b3d8df:/rallyvolume# ls
root@9a7dd7b3d8df:/rallyvolume/.rally# ls
benchmarks  logging.json  logs      rally.ini
# head -4 benchmarks/races/1d81930a-4ebe-4640-a09b-3055174bce43/race.json
{
 "rally-version": "1.2.1.dev0",
 "environment": "local",
 "race-id": "1d81930a-4ebe-4640-a09b-3055174bce43",
```

2.3.5 Specifics about the image

Rally runs as user `1000` and its files are installed with uid:gid `1000:0` (to support Arbitrary User IDs).

2.3.6 Extending the Docker image

You can also create your own customized Docker image on top of the existing one. The example below shows how to get started:

```
FROM elastic/rally:1.2.1
COPY --chown=1000:0 rally.ini /rally/.rally/
```

You can then build and test the image with:

```
docker build --tag=custom-rally .
docker run -ti custom-rally list tracks
```

2.4 Configuration

Rally has to be configured once after installation. If you just run `esrally` after installing Rally, it will detect that the configuration file is missing and asks you a few questions.

If you want to reconfigure Rally at any later time, just run `esrally configure` again.

2.4.1 Simple Configuration

By default, Rally will run a simpler configuration routine and autodetect as much settings as possible or choose defaults for you. If you need more control you can run Rally with `esrally configure --advanced-config`.

Rally can build Elasticsearch either from sources or use an official binary distribution. If you have Rally build Elasticsearch from sources, it can only be used to benchmark Elasticsearch 5.0 and above. The reason is that with Elasticsearch 5.0 the build tool switched from Maven to Gradle. As Rally utilizes the Gradle Wrapper, it is limited to Elasticsearch 5.0 and above.

Let's go through an example step by step: First run `esrally`:

```
dm@io:~ $ esrally

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

Running simple configuration. Run the advanced configuration with:

  esrally configure --advanced-config

* Setting up benchmark root directory in /Users/dm/.rally/benchmarks
* Setting up benchmark source directory in /Users/dm/.rally/benchmarks/src/
↪elasticsearch

Configuration successfully written to /Users/dm/.rally/rally.ini. Happy benchmarking!

More info about Rally:

* Type esrally --help
* Read the documentation at https://esrally.readthedocs.io/en/latest/
* Ask a question on the forum at https://discuss.elastic.co/tags/c/elastic-stack/
↪elasticsearch/rally
```

Congratulations! Time to *run your first benchmark*.

2.4.2 Advanced Configuration

If you need more control over a few variables or want to store your metrics in a dedicated Elasticsearch metrics store, then you should run the advanced configuration routine. You can invoke it at any time with `esrally configure --advanced-config`.

Prerequisites

When using the advanced configuration, you can choose that Rally stores its metrics not in-memory but in a dedicated Elasticsearch instance. Therefore, you will also need the following software installed:

- Elasticsearch: a dedicated Elasticsearch instance which acts as the metrics store for Rally. If you don't want to set it up yourself you can also use Elastic Cloud.
- Optional: Kibana (also included in Elastic Cloud).

Preparation

First install Elasticsearch 6.0 or higher. A simple out-of-the-box installation with a single node will suffice. Rally uses this instance to store metrics data. It will setup the necessary indices by itself. The configuration procedure of Rally will you ask for host and port of this cluster.

Note: Rally will choose the port 39200 (HTTP) and 39300 (transport) for the benchmark cluster, so do not use these ports for the metrics store.

Optional but recommended is to install also Kibana. However, note that Kibana will not be auto-configured by Rally.

Configuration Options

Rally will ask you a few more things in the advanced setup:

- **Benchmark root directory**: Rally stores all benchmark related data in this directory which can take up to several tens of GB. If you want to use a dedicated partition, you can specify a different root directory here.
- **Elasticsearch project directory**: This is the directory where the Elasticsearch sources are located. If you don't actively develop on Elasticsearch you can just leave the default but if you want to benchmark local changes you should point Rally to your project directory. Note that Rally will run builds with the Gradle Wrapper in this directory (it runs `./gradlew clean` and `./gradlew :distribution:tar:assemble`).
- **Metrics store type**: You can choose between `in-memory` which requires no additional setup or `elasticsearch` which requires that you start a dedicated Elasticsearch instance to store metrics but gives you much more flexibility to analyse results.
- **Metrics store settings** (only for metrics store type `elasticsearch`): Provide the connection details to the Elasticsearch metrics store. This should be an instance that you use just for Rally but it can be a rather small one. A single node cluster with default setting should do it. When using self-signed certificates on the Elasticsearch metrics store, certificate verification can be turned off by setting the `datastore.ssl.verification_mode` setting to `none`. Alternatively you can enter the path to the certificate authority's signing certificate in `datastore.ssl.certificate_authorities`. Both settings are optional.
- **Name for this benchmark environment** (only for metrics store type `elasticsearch`): You can use the same metrics store for multiple environments (e.g. local, continuous integration etc.) so you can separate metrics from different environments by choosing a different name.
- whether or not Rally should keep the Elasticsearch benchmark candidate installation including all data by default. This will use lots of disk space so you should wipe `~/.rally/benchmarks/races` regularly.

2.4.3 Proxy Configuration

Rally downloads all necessary data automatically for you:

- Elasticsearch distributions from elastic.co if you specify `--distribution-version=SOME_VERSION_NUMBER`
- Elasticsearch source code from Github if you specify a revision number e.g. `--revision=952097b`
- Track meta-data from Github
- Track data from an S3 bucket

Hence, it needs to connect via http(s) to the outside world. If you are behind a corporate proxy you need to configure Rally and git. As many other Unix programs, Rally relies that the HTTP proxy URL is available in the environment variable `http_proxy` (note that this is in lower-case). Hence, you should add this line to your shell profile, e.g. `~/.bash_profile`:

```
export http_proxy=http://proxy.acme.org:8888/
```

Afterwards, source the shell profile with `source ~/.bash_profile` and verify that the proxy URL is correctly set with `echo $http_proxy`.

Finally, you can set up git (see also the Git config documentation):

```
git config --global http.proxy $http_proxy
```

Verify that the proxy setup for git works correctly by cloning any repository, e.g. the `rally-tracks` repository:

```
git clone https://github.com/elastic/rally-tracks.git
```

If the configuration is correct, git will clone this repository. You can delete the folder `rally-tracks` after this verification step.

To verify that Rally will connect via the proxy server you can check the log file. If the proxy server is configured successfully, Rally will log the following line on startup:

```
Rally connects via proxy URL [http://proxy.acme.org:3128/] to the Internet (picked up␣
↪from the environment variable [http_proxy]).
```

Note: Rally will use this proxy server only for downloading benchmark-related data. It will not use this proxy for the actual benchmark.

2.4.4 Logging

Logging in Rally is configured in `~/.rally/logging.json`. For more information about the log file format please refer to the following documents:

- Python logging cookbook provides general tips and tricks.
- The Python reference documentation on the logging configuration schema explains the file format.
- The logging handler documentation describes how to customize where log output is written to.

By default, Rally will log all output to `~/.rally/logs/rally.log`.

The log file will not be rotated automatically as this is problematic due to Rally's multi-process architecture. Setup an external tool like logrotate to achieve that. See the following example as a starting point for your own `logrotate` configuration and ensure to replace the path `/home/user/.rally/logs/rally.log` with the proper one:

```
/home/user/.rally/logs/rally.log {
        # rotate daily
        daily
        # keep the last seven log files
        rotate 7
        # remove logs older than 14 days
        maxage 14
        # compress old logs ...
        compress
        # ... after moving them
        delaycompress
        # ignore missing log files
        missingok
        # don't attempt to rotate empty ones
        notifempty
}
```

Example

With the following configuration Rally will log all output to standard error:

```
{
  "version": 1,
  "formatters": {
    "normal": {
      "format": "%(asctime)s,%(msecs)d %(actorAddress)s/PID:%(process)d %(name)s
↪%(levelname)s %(message)s",
      "datefmt": "%Y-%m-%d %H:%M:%S",
      "()": "esrally.log.configure_utc_formatter"
    }
  },
  "filters": {
    "isActorLog": {
      "()": "thespian.director.ActorAddressLogFilter"
    }
  },
  "handlers": {
    "console_log_handler": {
        "class": "logging.StreamHandler",
        "formatter": "normal",
        "filters": ["isActorLog"]
    }
  },
  "root": {
    "handlers": ["console_log_handler"],
    "level": "INFO"
  },
  "loggers": {
    "elasticsearch": {
      "handlers": ["console_log_handler"],
      "level": "WARNING",
      "propagate": false
    }
  }
}
```

2.5 Run a Benchmark: Races

2.5.1 Definition

A “race” in Rally is the execution of a benchmarking experiment. You can choose different benchmarking scenarios (called *tracks*) for your benchmarks.

2.5.2 List Tracks

Start by finding out which tracks are available:

```
esrally list tracks
```

This will show the following list:

```
Name        Description                                          Documents ␣
↪Compressed Size    Uncompressed Size    Default Challenge        All Challenges
----------  -------------------------------------------------  -----------  ----------
↪
```

(continues on next page)

(continued from previous page)

```
geonames    POIs from Geonames                                  11396505  252.4 MB ␣
↪           3.3 GB               append-no-conflicts      append-no-conflicts,appe...
geopoint    Point coordinates from PlanetOSM                    60844404  481.9 MB ␣
↪           2.3 GB               append-no-conflicts      append-no-conflicts,appe...
http_logs   HTTP server log data                               247249096  1.2 GB   ␣
↪           31.1 GB              append-no-conflicts      append-no-conflicts,appe...
nested      StackOverflow Q&A stored as nested docs             11203029  663.1 MB ␣
↪           3.4 GB               nested-search-challenge  nested-search-challenge,...
noaa        Global daily weather measurements from NOAA         33659481  947.3 MB ␣
↪           9.0 GB               append-no-conflicts      append-no-conflicts,appe...
nyc_taxis   Taxi rides in New York in 2015                     165346692  4.5 GB   ␣
↪           74.3 GB              append-no-conflicts      append-no-conflicts,appe...
percolator  Percolator benchmark based on AOL queries            2000000  102.7 kB ␣
↪           104.9 MB             append-no-conflicts      append-no-conflicts,appe...
pmc         Full text benchmark with academic papers from PMC     574199  5.5 GB   ␣
↪           21.7 GB              append-no-conflicts      append-no-conflicts,appe...
```

The first two columns show the name and a description of each track. A track also specifies one or more challenges which describe the workload to run.

2.5.3 Starting a Race

Note: Do not run Rally as root as Elasticsearch will refuse to start with root privileges.

To start a race you have to define the track and challenge to run. For example:

```
esrally --distribution-version=6.0.0 --track=geopoint --challenge=append-fast-with-
↪conflicts
```

Rally will then start racing on this track. If you have never started Rally before, it should look similar to the following output:

```
dm@io:~ $ esrally --distribution-version=6.0.0 --track=geopoint --challenge=append-
↪fast-with-conflicts

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

[INFO] Racing on track [geopoint], challenge [append-fast-with-conflicts] and car [
↪'defaults'] with version [6.0.0].
[INFO] Downloading Elasticsearch 6.0.0 ... [OK]
[INFO] Rally will delete the benchmark candidate after the benchmark
[INFO] Downloading data from [http://benchmarks.elasticsearch.org.s3.amazonaws.com/
↪corpora/geopoint/documents.json.bz2] (482 MB) to [/Users/dm/.rally/benchmarks/data/
↪geopoint/documents.json.bz2] ... [OK]
[INFO] Decompressing track data from [/Users/dm/.rally/benchmarks/data/geopoint/
↪documents.json.bz2] to [/Users/dm/.rally/benchmarks/data/geopoint/documents.json]␣
↪(resulting size: 2.28 GB) ... [OK]
[INFO] Preparing file offset table for [/Users/dm/.rally/benchmarks/data/geopoint/
↪documents.json] ... [OK]
```

(continues on next page)

(continued from previous page)

```
Running index-update                                                            [   0%␣
↪done]
```

The benchmark will take a while to run, so be patient.

When the race has finished, Rally will show a summary on the command line:

```
|                          Metric |          Task |      Value |   Unit |
|--------------------------------:|--------------:|-----------:|-------:|
|             Total indexing time |               |    124.712 |    min |
|                Total merge time |               |    21.8604 |    min |
|              Total refresh time |               |    4.49527 |    min |
|       Total merge throttle time |               |   0.120433 |    min |
|                Median CPU usage |               |      546.5 |      % |
|         Total Young Gen GC time |               |     72.078 |      s |
|        Total Young Gen GC count |               |         43 |        |
|           Total Old Gen GC time |               |      3.426 |      s |
|          Total Old Gen GC count |               |          1 |        |
|                      Index size |               |    2.26661 |     GB |
|                   Total written |               |     30.083 |     GB |
|          Heap used for segments |               |    10.7148 |     MB |
|        Heap used for doc values |               |  0.0135536 |     MB |
|             Heap used for terms |               |    9.22965 |     MB |
|            Heap used for points |               |    0.78789 |     MB |
|     Heap used for stored fields |               |   0.683708 |     MB |
|                   Segment count |               |        115 |        |
|                  Min Throughput |  index-update |    59210.4 | docs/s |
|               Median Throughput |  index-update |    65276.2 | docs/s |
|                  Max Throughput |  index-update |    76516.6 | docs/s |
|       50.0th percentile latency |  index-update |    556.269 |     ms |
|       90.0th percentile latency |  index-update |    852.779 |     ms |
|       99.0th percentile latency |  index-update |    1854.31 |     ms |
|       99.9th percentile latency |  index-update |    2972.96 |     ms |
|      99.99th percentile latency |  index-update |    4106.91 |     ms |
|        100th percentile latency |  index-update |    4542.84 |     ms |
|  50.0th percentile service time |  index-update |    556.269 |     ms |
|  90.0th percentile service time |  index-update |    852.779 |     ms |
|  99.0th percentile service time |  index-update |    1854.31 |     ms |
|  99.9th percentile service time |  index-update |    2972.96 |     ms |
| 99.99th percentile service time |  index-update |    4106.91 |     ms |
|   100th percentile service time |  index-update |    4542.84 |     ms |
|                  Min Throughput |   force-merge |   0.221067 |  ops/s |
|               Median Throughput |   force-merge |   0.221067 |  ops/s |
|                  Max Throughput |   force-merge |   0.221067 |  ops/s |
|        100th percentile latency |   force-merge |    4523.52 |     ms |
|   100th percentile service time |   force-merge |    4523.52 |     ms |


----------------------------------
[INFO] SUCCESS (took 1624 seconds)
----------------------------------
```

Note: You can save this report also to a file by using `--report-file=/path/to/your/report.md` and save it as CSV with `--report-format=csv`.

What did Rally just do?

- It downloaded and started Elasticsearch 6.0.0
- It downloaded the relevant data for the geopoint track
- It ran the actual benchmark
- And finally it reported the results

If you are curious about the operations that Rally has run, inspect the geopoint track specification or start to *write your own tracks*. You can also configure Rally to *store all data samples in Elasticsearch* so you can analyze the results with Kibana. Finally, you may want to *change the Elasticsearch configuration*.

2.6 Compare Results: Tournaments

Suppose, we want to analyze the impact of a performance improvement.

First, we need a baseline measurement. For example:

```
esrally --track=pmc --revision=latest --user-tag="intention:baseline_github_1234"
```

Above we run the baseline measurement based on the latest source code revision of Elasticsearch. We can use the command line parameter `--user-tag` to provide a key-value pair to document the intent of a race.

Then we implement our changes and finally we want to run another benchmark to see the performance impact of the change. In that case, we do not want Rally to change our source tree and thus specify the pseudo-revision `current`:

```
esrally --track=pmc --revision=current --user-tag="intention:reduce_alloc_1234"
```

After we've run both races, we want to know about the performance impact. With Rally we can analyze differences of two given races easily. First of all, we need to find two races to compare by issuing `esrally list races`:

```
dm@io:~ $ esrally list races

    ____        ____
   / __ \____ _/ / /_  __
  / /_/ / __ `/ / / / / /
 / _, _/ /_/ / / / /_/ /
/_/ |_|\__,_/_/_/\__, /
                /____/
Recent races:

Race ID                               Race Timestamp    Track    Track Parameters   ␣
↪Challenge              Car       User Tags
------------------------------------  ----------------  -------  ------------------  -
↪-----------------   ---------  ------------------------------
beb154e4-0a05-4f45-ad9f-e34f9a9e51f7  20160518T122341Z  pmc                          ␣
↪append-no-conflicts  defaults  intention:reduce_alloc_1234
0bfd4542-3821-4c79-81a2-0858636068ce  20160518T112057Z  pmc                          ␣
↪append-no-conflicts  defaults  intention:baseline_github_1234
0cfb3576-3025-4c17-b672-d6c9e811b93e  20160518T101957Z  pmc                          ␣
↪append-no-conflicts  defaults
```

We can see that the user tag helps us to recognize races. We want to compare the two most recent races and have to provide the two race IDs in the next step:

```
dm@io:~ $ esrally compare --baseline=0bfd4542-3821-4c79-81a2-0858636068ce --
↪contender=beb154e4-0a05-4f45-ad9f-e34f9a9e51f7

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

Comparing baseline
  Race ID: 0bfd4542-3821-4c79-81a2-0858636068ce
  Race timestamp: 2016-05-18 11:20:57
  Challenge: append-no-conflicts
  Car: defaults

with contender
  Race ID: beb154e4-0a05-4f45-ad9f-e34f9a9e51f7
  Race timestamp: 2016-05-18 12:23:41
  Challenge: append-no-conflicts
  Car: defaults

------------------------------------------------------
    _______             __   _____
   / ____(_)___  ____ _/ /  / ___/_________  ________
  / /_  / / __ \/ __ `/ /   \__ \/ ___/ __ \/ ___/ _ \
 / __/ / / / / / /_/ / /   ___/ / /__/ /_/ / /  /  __/
/_/   /_/_/ /_/\__,_/_/   /____/\___/\____/_/   \___/
------------------------------------------------------
                                                  Metric    Baseline    Contender    ␣
↪          Diff
--------------------------------------------------------  ----------  -----------  ---
↪--------------
                        Min Indexing Throughput [docs/s]       19501        19118  -
↪383.00000
                     Median Indexing Throughput [docs/s]       20232      19927.5  -
↪304.45833
                        Max Indexing Throughput [docs/s]       21172        20849  -
↪323.00000
                               Total indexing time [min]     55.7989       56.335   ␣
↪+0.53603
                                  Total merge time [min]     12.9766      13.3115   ␣
↪+0.33495
                                Total refresh time [min]     5.20067      5.20097   ␣
↪+0.00030
                                  Total flush time [min]   0.0648667    0.0681833   ␣
↪+0.00332
                         Total merge throttle time [min]    0.796417     0.879267   ␣
↪+0.08285
               Query latency term (50.0 percentile) [ms]     2.10049      2.15421   ␣
↪+0.05372
               Query latency term (90.0 percentile) [ms]     2.77537      2.84168   ␣
↪+0.06630
              Query latency term (100.0 percentile) [ms]     4.52081      5.15368   ␣
↪+0.63287
        Query latency country_agg (50.0 percentile) [ms]     112.049      110.385    -
↪1.66392
```

(continues on next page)

(continued from previous page)

```
        Query latency country_agg (90.0 percentile) [ms]     128.426      124.005     -
↪4.42138
       Query latency country_agg (100.0 percentile) [ms]     155.989      133.797    -
↪22.19185
             Query latency scroll (50.0 percentile) [ms]     16.1226      14.4974     -
↪1.62519
             Query latency scroll (90.0 percentile) [ms]     17.2383      15.4079     -
↪1.83043
            Query latency scroll (100.0 percentile) [ms]     18.8419      18.4241     -
↪0.41784
 Query latency country_agg_cached (50.0 percentile) [ms]     1.70223      1.64502     -
↪0.05721
 Query latency country_agg_cached (90.0 percentile) [ms]     2.34819      2.04318     -
↪0.30500
Query latency country_agg_cached (100.0 percentile) [ms]     3.42547      2.86814     -
↪0.55732
            Query latency default (50.0 percentile) [ms]     5.89058      5.83409     -
↪0.05648
            Query latency default (90.0 percentile) [ms]     6.71282      6.64662     -
↪0.06620
           Query latency default (100.0 percentile) [ms]     7.65307       7.3701     -
↪0.28297
             Query latency phrase (50.0 percentile) [ms]     1.82687      1.83193    ␣
↪+0.00506
             Query latency phrase (90.0 percentile) [ms]     2.63714      2.46286     -
↪0.17428
            Query latency phrase (100.0 percentile) [ms]     5.39892      4.22367     -
↪1.17525
                            Median CPU usage (index) [%]     668.025       679.15   ␣
↪+11.12499
                            Median CPU usage (stats) [%]      143.75        162.4   ␣
↪+18.64999
                           Median CPU usage (search) [%]       223.1        229.2    ␣
↪+6.10000
                             Total Young Gen GC time [s]      39.447       40.456    ␣
↪+1.00900
                                Total Young Gen GC count          10           11    ␣
↪+1.00000
                               Total Old Gen GC time [s]       7.108        7.703    ␣
↪+0.59500
                                  Total Old Gen GC count          10           11    ␣
↪+1.00000
                                       Index size [GB]     3.25475      3.25098     -
↪0.00377
                                    Total written [GB]     17.8434      18.3143    ␣
↪+0.47083
                             Heap used for segments [MB]     21.7504      21.5901     -
↪0.16037
                           Heap used for doc values [MB]     0.16436      0.13905     -
↪0.02531
                                Heap used for terms [MB]     20.0293      19.9159     -
↪0.11345
                                Heap used for norms [MB]    0.105469    0.0935669     -
↪0.01190
                               Heap used for points [MB]    0.773487     0.772155     -
↪0.00133
                               Heap used for points [MB]    0.677795     0.669426     -
↪0.00837
```

(continues on next page)

(continued from previous page)

```
                                        Segment count          136           121    -
↪15.00000
                     Indices Stats(90.0 percentile) [ms]     3.16053       3.21023    ␣
↪+0.04969
                     Indices Stats(99.0 percentile) [ms]     5.29526       3.94132     -
↪1.35393
                    Indices Stats(100.0 percentile) [ms]     5.64971       7.02374    ␣
↪+1.37403
                       Nodes Stats(90.0 percentile) [ms]     3.19611       3.15251     -
↪0.04360
                       Nodes Stats(99.0 percentile) [ms]     4.44111       4.87003    ␣
↪+0.42892
                      Nodes Stats(100.0 percentile) [ms]     5.22527       5.66977    ␣
↪+0.44450
```

2.7 Setting up a Cluster

In this section we cover how to setup an Elasticsearch cluster with Rally. It is by no means required to use Rally for this and you can also use existing tooling like Ansible to achieve the same goal. The main difference between standard tools and Rally is that Rally is capable of setting up a *wide range of Elasticsearch versions*.

Warning: The following functionality is experimental. Expect the functionality and the command line interface to change significantly even in patch releases.

2.7.1 Overview

You can use the following subcommands in Rally to manage Elasticsearch nodes:

- `install` to install a single Elasticsearch node
- `start` to start a previously installed Elasticsearch node
- `stop` to stop a running Elasticsearch node and remove the installation

Each command needs to be executed locally on the machine where the Elasticsearch node should run. To setup more complex clusters remotely, we recommend using a tool like Ansible to connect to remote machines and issue these commands via Rally.

2.7.2 Getting Started: Benchmarking a Single Node

In this section we will setup a single Elasticsearch node locally, run a benchmark and then cleanup.

First we need to install Elasticearch:

```
esrally install --quiet --distribution-version=7.4.2 --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"
```

The parameter `--network-host` defines the network interface this node will bind to and `--http-port` defines which port will be exposed for HTTP traffic. Rally will automatically choose the transport port range as 100 above (39300). The parameters `--master-nodes` and `--seed-hosts` are necessary for the discovery process. Please see the respective Elasticsearch documentation on discovery for more details.

This produces the following output (the value will vary for each invocation):

```
{
  "installation-id": "69ffcfee-6378-4090-9e93-87c9f8ee59a7"
}
```

We will need the installation id in the next steps to refer to our current installation.

Note: You can extract the installation id value with jq with the following command: `jq --raw-output '.["installation-id"]'`.

After installation, we can start the node. To tie all metrics of a benchmark together, Rally needs a consistent race id across all invocations. The format of the race id does not matter but we suggest using UUIDs. You can generate a UUID on the command line with `uuidgen`. Issue the following command to start the node:

```
# generate a unique race id (use the same id from now on)
export RACE_ID=$(uuidgen)
esrally start --installation-id="69ffcfee-6378-4090-9e93-87c9f8ee59a7" --race-id="$
↪{RACE_ID}"
```

After the Elasticsearch node has started, we can run a benchmark. Be sure to pass the same race id so you can match results later in your metrics store:

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

When the benchmark has finished, we can stop the node again:

```
esrally stop --installation-id="69ffcfee-6378-4090-9e93-87c9f8ee59a7"
```

If you only want to shutdown the node but don't want to delete the node and the data, pass `--preserve-install` additionally.

2.7.3 Levelling Up: Benchmarking a Cluster

This approach of being able to manage individual cluster nodes shows its power when we want to setup a cluster consisting of multiple nodes. At the moment Rally only supports a uniform cluster architecture but with this approach we can also setup arbitrarily complex clusters. The following examples shows how to setup a uniform three node cluster on three machines with the IPs `192.168.14.77`, `192.168.14.78` and `192.168.14.79`. On each machine we will issue the following command (pick the right one per machine):

```
# on 192.168.14.77
export INSTALLATION_ID=$(esrally install --quiet --distribution-version=7.4.2 --node-
↪name="rally-node-0" --network-host="192.168.14.77" --http-port=39200 --master-nodes=
↪"rally-node-0,rally-node-1,rally-node-2" --seed-hosts="192.168.14.77:39300,192.168.
↪14.78:39300,192.168.14.79:39300" | jq --raw-output '.["installation-id"]')
# on 192.168.14.78
export INSTALLATION_ID=$(esrally install --quiet --distribution-version=7.4.2 --node-
↪name="rally-node-1" --network-host="192.168.14.78" --http-port=39200 --master-nodes=
↪"rally-node-0,rally-node-1,rally-node-2" --seed-hosts="192.168.14.77:39300,192.168.
↪14.78:39300,192.168.14.79:39300" | jq --raw-output '.["installation-id"]')
# on 192.168.14.79
export INSTALLATION_ID=$(esrally install --quiet --distribution-version=7.4.2 --node-
↪name="rally-node-2" --network-host="192.168.14.79" --http-port=39200 --master-nodes=
↪"rally-node-0,rally-node-1,rally-node-2" --seed-hosts="192.168.14.77:39300,192.168.
↪14.78:39300,192.168.14.79:39300" | jq --raw-output '.["installation-id"]')
```

(continues on next page)

(continued from previous page)

```
```

Then we pick a random race id, e.g. `fb38013d-5d06-4b81-b81a-b61c8c10f6e5` and set it on each machine (including the machine where will generate load):

```
export RACE_ID="fb38013d-5d06-4b81-b81a-b61c8c10f6e5"
```

Now we can start the cluster. Run the following command on each node:

```
esrally start --installation-id="${INSTALLATION_ID}" --race-id="${RACE_ID}"
```

Once this has finished, we can check that the cluster is up and running e.g. with the `_cat/health` API:

```
curl http://192.168.14.77:39200/_cat/health\?v
```

We should see that our cluster consisting of three nodes is up and running:

```
epoch      timestamp cluster         status node.total node.data shards pri relo init␣
↪unassign pending_tasks max_task_wait_time active_shards_percent
1574930657 08:44:17  rally-benchmark green           3         3      0   0    0    0␣
↪        0             0                  -                100.0%
```

Now we can start the benchmark on the load generator machine (remember to set the race id there):

```
esrally --pipeline=benchmark-only --target-host=192.168.14.77:39200,192.168.14.
↪78:39200,192.168.14.79:39200 --track=geonames --challenge=append-no-conflicts-index-
↪only --on-error=abort --race-id=${RACE_ID}
```

Similarly to the single-node benchmark, we can now shutdown the cluster again by issuing the following command on each node:

```
esrally stop --installation-id="${INSTALLATION_ID}"
```

2.8 Tips and Tricks

This section covers various tips and tricks in a recipe-style fashion.

2.8.1 Benchmarking an Elastic Cloud cluster

Note: We assume in this recipe, that Rally is already properly *configured*.

Benchmarking an Elastic Cloud cluster with Rally is similar to *benchmarking any other existing cluster*. In the following example we will run a benchmark against a cluster reachable via the endpoint `https://abcdef123456.europe-west1.gcp.cloud.es.io:9243` by the user `elastic` with the password `changeme`:

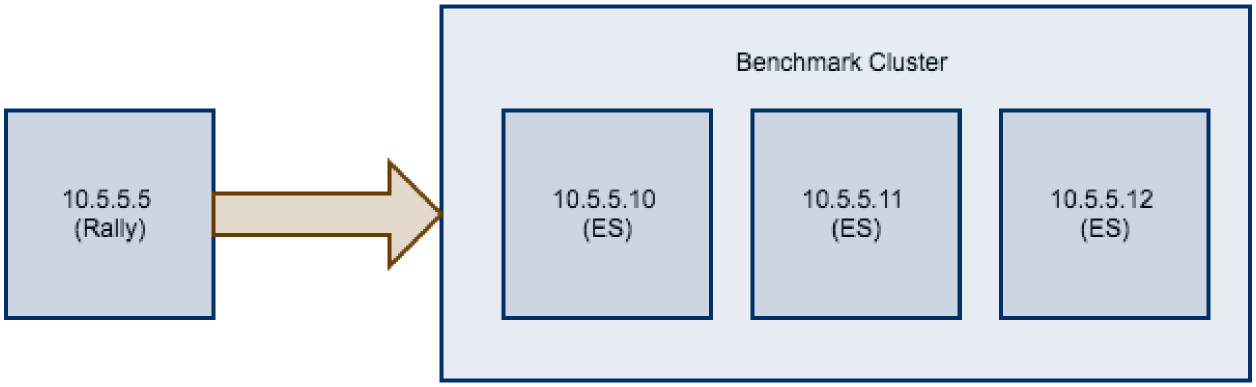
```
esrally --track=pmc --target-hosts=abcdef123456.europe-west1.gcp.cloud.es.io:9243 --
↪pipeline=benchmark-only --client-options="timeout:60,use_ssl:true,verify_certs:true,
↪basic_auth_user:'elastic',basic_auth_password:'changeme'"
```

2.8.2 Benchmarking an existing cluster

Warning: If you are just getting started with Rally and don't understand how it works, do NOT run it against any production or production-like cluster. Besides, benchmarks should be executed in a dedicated environment anyway where no additional traffic skews results.

Note: We assume in this recipe, that Rally is already properly *configured*.

Consider the following configuration: You have an existing benchmarking cluster, that consists of three Elasticsearch nodes running on `10.5.5.10`, `10.5.5.11`, `10.5.5.12`. You've setup the cluster yourself and want to benchmark it with Rally. Rally is installed on `10.5.5.5`.



First of all, we need to decide on a track. So, we run `esrally list tracks`:

```
Name         Description                                          Documents ␣
↪Compressed Size    Uncompressed Size    Default Challenge        All Challenges
----------   -------------------------------------------------   -----------   ----------
↪-------   -------------------   -----------------------   ---------------------------
geonames     POIs from Geonames                                    11396505   252.4 MB ␣
↪           3.3 GB              append-no-conflicts      append-no-conflicts,appe...
geopoint     Point coordinates from PlanetOSM                      60844404   481.9 MB ␣
↪           2.3 GB              append-no-conflicts      append-no-conflicts,appe...
http_logs    HTTP server log data                                 247249096   1.2 GB   ␣
↪           31.1 GB             append-no-conflicts      append-no-conflicts,appe...
nested       StackOverflow Q&A stored as nested docs               11203029   663.1 MB ␣
↪           3.4 GB              nested-search-challenge  nested-search-challenge,...
noaa         Global daily weather measurements from NOAA           33659481   947.3 MB ␣
↪           9.0 GB              append-no-conflicts      append-no-conflicts,appe...
nyc_taxis    Taxi rides in New York in 2015                       165346692   4.5 GB   ␣
↪           74.3 GB             append-no-conflicts      append-no-conflicts,appe...
percolator   Percolator benchmark based on AOL queries              2000000   102.7 kB ␣
↪           104.9 MB            append-no-conflicts      append-no-conflicts,appe...
pmc          Full text benchmark with academic papers from PMC       574199   5.5 GB   ␣
↪           21.7 GB             append-no-conflicts      append-no-conflicts,appe...
```

We're interested in a full text benchmark, so we'll choose to run `pmc`. If you have your own data that you want to use for benchmarks *create your own track* instead; the metrics you'll gather will be more representative and useful than some default track.

Next, we need to know which machines to target which is easy as we can see that from the diagram above.

Finally we need to check which *pipeline* to use. For this case, the `benchmark-only` pipeline is suitable as we don't want Rally to provision the cluster for us.

Now we can invoke Rally:

```
esrally --track=pmc --target-hosts=10.5.5.10:9200,10.5.5.11:9200,10.5.5.12:9200 --
↪pipeline=benchmark-only
```

If you have X-Pack Security enabled, then you'll also need to specify another parameter to use https and to pass credentials:

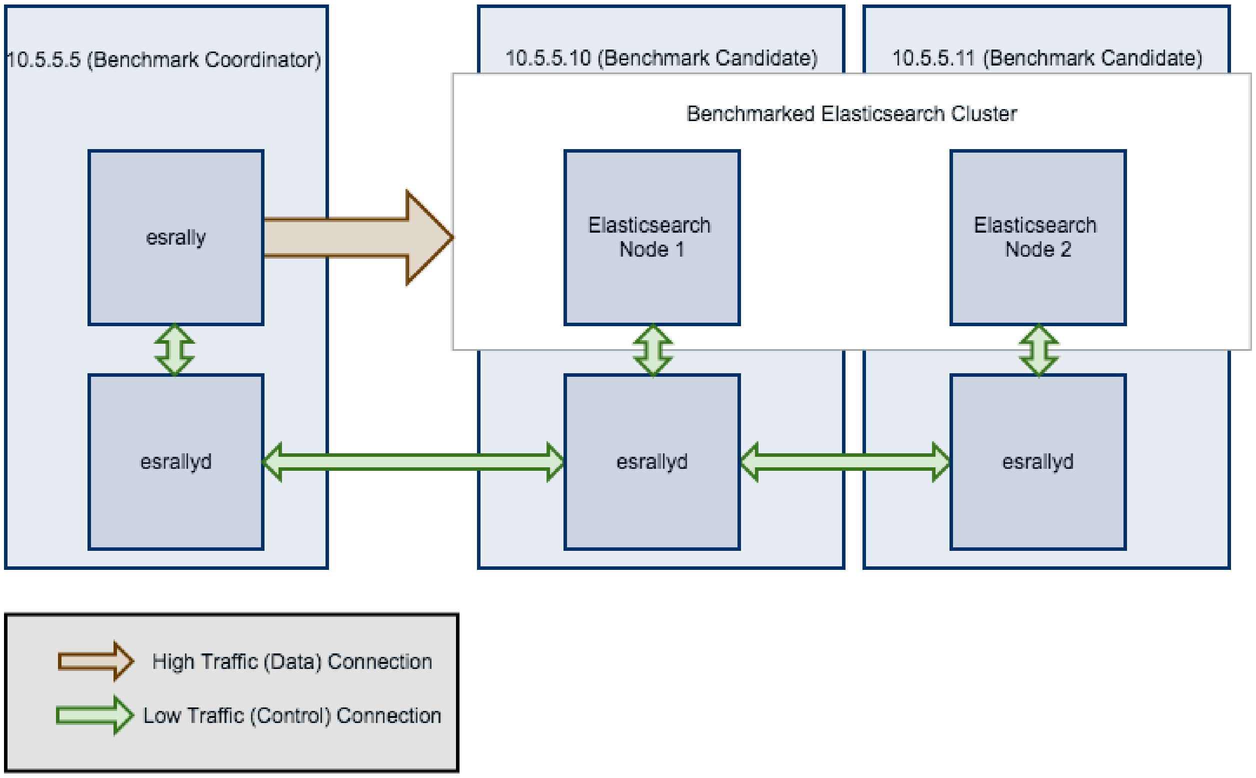
```
esrally --track=pmc --target-hosts=10.5.5.10:9243,10.5.5.11:9243,10.5.5.12:9243 --
↪pipeline=benchmark-only --client-options="use_ssl:true,verify_certs:true,basic_auth_
↪user:'elastic',basic_auth_password:'changeme'"
```

2.8.3 Benchmarking a remote cluster

Contrary to the previous recipe, you want Rally to provision all cluster nodes.

We will use the following configuration for the example:

- You will start Rally on `10.5.5.5`. We will call this machine the "benchmark coordinator".
- Your Elasticsearch cluster will consist of two nodes which run on `10.5.5.10` and `10.5.5.11`. We will call these machines the "benchmark candidate"s.


10.5.5.5 (Benchmark Coordinator)
10.5.5.10 (Benchmark Candidate)
10.5.5.11 (Benchmark Candidate)
Benchmarked Elasticsearch Cluster
esrally
Elasticsearch Node 1
Elasticsearch Node 2
esrallyd
esrallyd
esrallyd
High Traffic (Data) Connection
Low Traffic (Control) Connection

Note: All `esrallyd` nodes form a cluster that communicates via the "benchmark coordinator". For aesthetic reasons we do not show a direct connection between the "benchmark coordinator" and all nodes.

To run a benchmark for this scenario follow these steps:

1. *Install* and *configure* Rally on all machines. Be sure that the same version is installed on all of them and fully *configured*.
2. Start the *Rally daemon* on each machine. The Rally daemon allows Rally to communicate with all remote machines. On the benchmark coordinator run `esrallyd start --node-ip=10.5.5.5 --coordinator-ip=10.5.5.5` and on the benchmark candidate machines run `esrallyd start --node-ip=10.5.5.10 --coordinator-ip=10.5.5.5` and `esrallyd start --node-ip=10.5.5.11 --coordinator-ip=10.5.5.5` respectively. The `--node-ip` parameter tells Rally the IP of the machine on which it is running. As some machines have more than one network interface, Rally will not attempt to auto-detect the machine IP. The `--coordinator-ip` parameter tells Rally the IP of the benchmark coordinator node.
3. Start the benchmark by invoking Rally as usual on the benchmark coordinator, for example: `esrally --distribution-version=5.0.0 --target-hosts=10.5.5.10:39200,10.5.5.11:39200`. Rally will derive from the `--target-hosts` parameter that it should provision the nodes `10.5.5.10` and `10.5.5.11`.
4. After the benchmark has finished you can stop the Rally daemon again. On the benchmark coordinator and on the benchmark candidates run `esrallyd stop`.

Note: Logs are managed per machine, so all relevant log files and also telemetry output is stored on the benchmark candidates but not on the benchmark coordinator.

Now you might ask yourself what the differences to benchmarks of existing clusters are. In general you should aim to give Rally as much control as possible as benchmark are easier reproducible and you get more metrics. The following table provides some guidance on when to choose which option:

Your requirement	Recommendation
You want to use Rally's telemetry devices	Use Rally daemon, as it can provision the remote node for you
You want to benchmark a source build of Elasticsearch	Use Rally daemon, as it can build Elasticsearch for you
You want to tweak the cluster configuration yourself	Use Rally daemon with a *custom configuration* or set up the cluster by yourself and use `--pipeline=benchmark-only`
You need to run a benchmark with plugins	Use Rally daemon if the *plugins* are supported or set up the cluster by yourself and use `--pipeline=benchmark-only`
You need to run a benchmark against multiple nodes	Use Rally daemon if all nodes can be configured identically. For more complex cases, set up the cluster by yourself and use `--pipeline=benchmark-only`

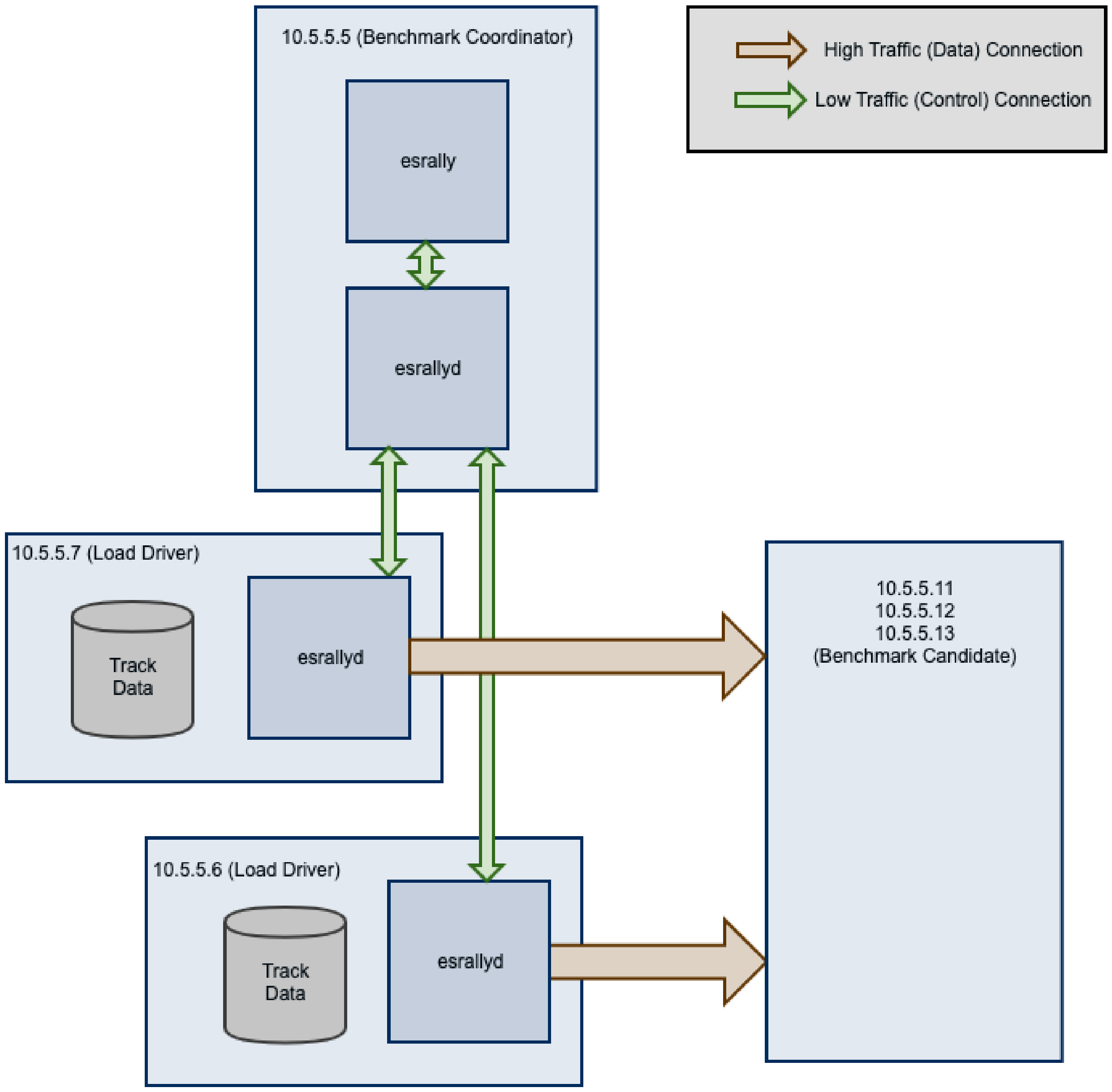
Rally daemon will be able to cover most of the cases described above in the future so there should be almost no case where you need to use the `benchmark-only` pipeline.

2.8.4 Distributing the load test driver

By default, Rally will generate load on the same machine where you start a benchmark. However, when you are benchmarking larger clusters, a single load test driver machine may not be able to generate sufficient load. In these cases, you should use multiple load driver machines. We will use the following configuration for the example:

- You will start Rally on `10.5.5.5`. We will call this machine the "benchmark coordinator".
- You will start two load drivers on `10.5.5.6` and `10.5.5.7`. Note that one load driver will simulate multiple clients. Rally will simply assign clients to load driver machines in a round-robin fashion.

- Your Elasticsearch cluster will consist of three nodes which run on `10.5.5.11`, `10.5.5.12` and `10.5.5.13`. We will call these machines the "benchmark candidate". For simplicity, we will assume an externally provisioned cluster but you can also use Rally to setup the cluster for you (see above).



1. *Install* and *configure* Rally on all machines. Be sure that the same version is installed on all of them and fully *configured*.
2. Start the *Rally daemon* on each machine. The Rally daemon allows Rally to communicate with all remote machines. On the benchmark coordinator run `esrallyd start --node-ip=10.5.5.5 --coordinator-ip=10.5.5.5` and on the load driver machines run `esrallyd start --node-ip=10.5.5.6 --coordinator-ip=10.5.5.5` and `esrallyd start --node-ip=10.5.5.7 --coordinator-ip=10.5.5.5` respectively. The `--node-ip` parameter tells Rally the IP of the machine on which it is running. As some machines have more than one network interface, Rally will not attempt to auto-detect the machine IP. The `--coordinator-ip` parameter tells Rally the IP of the benchmark coordinator node.

3. Start the benchmark by invoking Rally on the benchmark coordinator, for example: `esrally --pipeline=benchmark-only --load-driver-hosts=10.5.5.6,10.5.5.7 --target-hosts=10.5.5.11:9200,10.5.5.12:9200,10.5.5.13:9200`.
4. After the benchmark has finished you can stop the Rally daemon again. On the benchmark coordinator and on the load driver machines run `esrallyd stop`.

Note: Rally neither distributes code (i.e. *custom runners* or *parameter sources*) nor data automatically. You should place all tracks and their data on all machines in the same directory before starting the benchmark. Alternatively, you can store your track in a custom track repository.

Note: As indicated in the diagram, track data will be downloaded by each load driver machine separately. If you want to avoid that, you can run a benchmark once without distributing the load test driver (i.e. do not specify `--load-driver-hosts`) and then copy the contents of `~/.rally/benchmarks/data` to all load driver machines.

2.8.5 Changing the default track repository

Rally supports multiple track repositories. This allows you for example to have a separate company-internal repository for your own tracks that is separate from Rally's default track repository. However, you always need to define `--track-repository=my-custom-repository` which can be cumbersome. If you want to avoid that and want Rally to use your own track repository by default you can just replace the default track repository definition in `~./rally/rally.ini`. Consider this example:

```
...
[tracks]
default.url = git@github.com:elastic/rally-tracks.git
teamtrackrepo.url = git@example.org/myteam/my-tracks.git
```

If `teamtrackrepo` should be the default track repository, just define it as `default.url`. E.g.:

```
...
[tracks]
default.url = git@example.org/myteam/my-tracks.git
old-rally-default.url=git@github.com:elastic/rally-tracks.git
```

Also don't forget to rename the folder of your local working copy as Rally will search for a track repository with the name `default`:

```
cd ~/.rally/benchmarks/tracks/
mv default old-rally-default
mv teamtrackrepo default
```

From now on, Rally will treat your repository as default and you need to run Rally with `--track-repository=old-rally-default` if you want to use the out-of-the-box Rally tracks.

2.8.6 Testing Rally against CCR clusters using a remote metric store

Testing Rally features (such as the `ccr-stats` telemetry device) requiring Elasticsearch clusters configured for cross-cluster replication can be a time consuming process. Use recipes/ccr in Rally's repository to test a simple but complete example.

Running the `start.sh` script requires Docker locally installed and performs the following actions:

1. Starts a single node (512MB heap) Elasticsearch cluster locally, to serve as a *metrics store*. It also starts Kibana attached to the Elasticsearch metric store cluster.
2. Creates a new configuration file for Rally under `~/.rally/rally-metricstore.ini` referencing Elasticsearch from step 1.
3. Starts two additional local Elasticsearch clusters with 1 node each, (version `7.3.2` by default) called `leader` and `follower` listening at ports 32901 and 32902 respectively. Each node uses 1GB heap.
4. Accepts the trial license.
5. Configures `leader` on the `follower` as a remote cluster.
6. Sets an auto-follow pattern on the follower for every index on the leader to be replicated as `<leader-index-name>-copy`.
7. Runs the geonames track, append-no-conflicts-index-only challenge challenge, ingesting only 20% of the corpus using 3 primary shards. It also enables the `ccr-stats` *telemetry device* with a sample rate interval of `1s`.

Rally will push metrics to the metric store configured in 1. and they can be visualized by accessing Kibana at http://locahost:5601.

To tear down everything issue `./stop.sh`.

It is possible to specify a different version of Elasticsearch for step 3. by setting `export ES_VERSION=<the_desired_version>`.

2.8.7 Identifying when errors have been encountered

Custom track development can be error prone especially if you are testing a new query. A number of reasons can lead to queries returning errors.

Consider a simple example Rally operation:

```
{
  "name": "geo_distance",
  "operation-type": "search",
  "index": "logs-*",
  "body": {
    "query": {
       "geo_distance": {
          "distance": "12km",
          "source.geo.location": "40,-70"
       }
    }
  }
}
```

This query requires the field `source.geo.location` to be mapped as a `geo_point` type. If incorrectly mapped, Elasticsearch will respond with an error.

Rally will not exit on errors (unless fatal e.g. ECONNREFUSED) by default, instead reporting errors in the summary report via the *Error Rate* statistic. This can potentially leading to misleading results. This behavior is by design and consistent with other load testing tools such as JMeter i.e. In most cases it is desirable that a large long running benchmark should not fail because of a single error response.

This behavior can also be changed, by invoking Rally with the *–on-error* switch e.g.:

```
esrally --track=geonames --on-error=abort
```

Errors can also be investigated if you have configured a *dedicated Elasticsearch metrics store*.

2.8.8 Checking Queries and Responses

As described above, errors can lead to misleading benchmarking results. Some issues, however, are more subtle and the result of queries not behaving and matching as intended.

Consider the following simple Rally operation:

```
{
  "name": "geo_distance",
  "operation-type": "search",
  "index": "logs-*",
  "body": {
    "query": {
      "term": {
        "http.request.method": {
          "value": "GET"
        }
      }
    }
  }
}
```

For this term query to match the field `http.request.method` needs to be type `keyword`. Should this field be dynamically mapped, its default type will be `text` causing the value `GET` to be analyzed, and indexed as `get`. The above query will in turn return `0` hits. The field should either be correctly mapped or the query modified to match on `http.request.method.keyword`.

Issues such as this can lead to misleading benchmarking results. Prior to running any benchmarks for analysis, we therefore recommended users ascertain whether queries are behaving as intended. Rally provides several tools to assist with this.

Firstly, users can set the *log level* for the Elasticsearch client to `DEBUG` i.e.:

```
"loggers": {
  "elasticsearch": {
    "handlers": ["rally_log_handler"],
    "level": "DEBUG",
    "propagate": false
  },
  "rally.profile": {
    "handlers": ["rally_profile_handler"],
    "level": "INFO",
    "propagate": false
  }
}
```

This will in turn ensure logs include the Elasticsearch query and accompanying response e.g.:

```
2019-12-16 14:56:08,389 -not-actor-/PID:9790 elasticsearch DEBUG > {"sort":[{
↪"geonameid":"asc"}],"query":{"match_all":{}}}
2019-12-16 14:56:08,389 -not-actor-/PID:9790 elasticsearch DEBUG < {"took":1,"timed_
↪out":false,"_shards":{"total":5,"successful":5,"skipped":0,"failed":0},"hits":{
↪"total":{"value":1000,"relation":"eq"},"max_score":null,"hits":[{"_index":"geonames
↪","_type":"_doc","_id":"Lb81D28Bu7VEEZ3mXFGw","_score":null,"_source":{"geonameid":
↪2986043, "name": "Pic de Font Blanca", "asciiname": "Pic de Font Blanca",
↪"alternatenames": "Pic de Font Blanca,Pic du Port", "feature_class": "T", "feature_
↪code": "PK", "country_code": "AD", "admin1_code": "00", "population": 0, "dem":
↪"2860", "timezone": "Europe/Andorra", "location": [1.53335, 42.64991]},"sort
↪":[2986043]},
```

(continues on next page)

(continued from previous page)

Users should discard any performance metrics collected from a benchmark with `DEBUG` logging. This will likely cause a client-side bottleneck so once the correctness of the queries has been established, disable this setting and re-run any benchmarks.

The number of hits from queries can also be investigated if you have configured a *dedicated Elasticsearch metrics store*. Specifically, documents within the index pattern `rally-metrics-*` contain a `meta` field with a summary of individual responses e.g.:

```
{
  "@timestamp" : 1597681313435,
  "relative-time" : 130273374,
  "race-id" : "452ad9d7-9c21-4828-848e-89974af3230e",
  "race-timestamp" : "20200817T160412Z",
  "environment" : "Personal",
  "track" : "geonames",
  "challenge" : "append-no-conflicts",
  "car" : "defaults",
  "name" : "latency",
  "value" : 270.77871300025436,
  "unit" : "ms",
  "sample-type" : "warmup",
  "meta" : {
    "source_revision" : "757314695644ea9a1dc2fecd26d1a43856725e65",
    "distribution_version" : "7.8.0",
    "distribution_flavor" : "oss",
    "pages" : 25,
    "hits" : 11396503,
    "hits_relation" : "eq",
    "timed_out" : false,
    "took" : 110,
    "success" : true
  },
  "task" : "scroll",
  "operation" : "scroll",
  "operation-type" : "Search"
}
```

2.9 Define Custom Workloads: Tracks

2.9.1 Definition

A track describes one or more benchmarking scenarios. Its structure is described in detail in the *track reference*.

2.9.2 Creating a track from data in an existing cluster

If you already have a cluster with data in it you can use the `create-track` subcommand of Rally to create a basic Rally track. To create a Rally track with data from the indices `products` and `companies` that are hosted by a locally running Elasticsearch cluster, issue the following command:

```
esrally create-track --track=acme --target-hosts=127.0.0.1:9200 --indices="products,
↪companies" --output-path=~/tracks
```

If you want to connect to a cluster with TLS and basic authentication enabled, for example via Elastic Cloud, also specify *–client-options* and change `basic_auth_user` and `basic_auth_password` accordingly:

```
esrally create-track --track=acme --target-hosts=abcdef123.us-central-1.gcp.cloud.es.
↪io:9243 --client-options="timeout:60,use_ssl:true,verify_certs:true,basic_auth_user:
↪'elastic',basic_auth_password:'secret-password'" --indices="products,companies" --
↪output-path=~/tracks
```

The track generator will create a folder with the track's name in the specified output directory:

```
> find tracks/acme
tracks/acme
tracks/acme/companies-documents.json
tracks/acme/companies-documents.json.bz2
tracks/acme/companies-documents-1k.json
tracks/acme/companies-documents-1k.json.bz2
tracks/acme/companies.json
tracks/acme/products-documents.json
tracks/acme/products-documents.json.bz2
tracks/acme/products-documents-1k.json
tracks/acme/products-documents-1k.json.bz2
tracks/acme/products.json
tracks/acme/track.json
```

The files are organized as follows:

- `track.json` contains the actual Rally track. For details see the *track reference*.
- `companies.json` and `products.json` contain the mapping and settings for the extracted indices.
- `*-documents.json(.bz2)` contains the sources of all the documents from the extracted indices. The files suffixed with `-1k` contain a smaller version of the document corpus to support *test mode*.

2.9.3 Creating a track from scratch

We will create the track "tutorial" step by step. We store everything in the directory `~/rally-tracks/tutorial` but you can choose any other location.

First, get some data. Geonames provides geo data under a creative commons license. Download allCountries.zip (around 300MB), extract it and inspect `allCountries.txt`.

The file is tab-delimited but to bulk-index data with Elasticsearch we need JSON. Convert the data with the following script:

```
import json

cols = (("geonameid", "int", True),
        ("name", "string", True),
        ("asciiname", "string", False),
        ("alternatenames", "string", False),
        ("latitude", "double", True),
        ("longitude", "double", True),
        ("feature_class", "string", False),
        ("feature_code", "string", False),
        ("country_code", "string", True),
        ("cc2", "string", False),
        ("admin1_code", "string", False),
        ("admin2_code", "string", False),
```

(continues on next page)

(continued from previous page)

```
        ("admin3_code", "string", False),
        ("admin4_code", "string", False),
        ("population", "long", True),
        ("elevation", "int", False),
        ("dem", "string", False),
        ("timezone", "string", False))


def main():
    with open("allCountries.txt", "rt", encoding="UTF-8") as f:
        for line in f:
            tup = line.strip().split("\t")
            record = {}
            for i in range(len(cols)):
                name, type, include = cols[i]
                if tup[i] != "" and include:
                    if type in ("int", "long"):
                        record[name] = int(tup[i])
                    elif type == "double":
                        record[name] = float(tup[i])
                    elif type == "string":
                        record[name] = tup[i]
            print(json.dumps(record, ensure_ascii=False))


if __name__ == "__main__":
    main()
```

Store the script as `toJSON.py` in the tutorial directory (`~/rally-tracks/tutorial`). Invoke it with `python3 toJSON.py > documents.json`.

Then store the following mapping file as `index.json` in the tutorial directory:

```
{
  "settings": {
    "index.number_of_replicas": 0
  },
  "mappings": {
    "docs": {
      "dynamic": "strict",
      "properties": {
        "geonameid": {
          "type": "long"
        },
        "name": {
          "type": "text"
        },
        "latitude": {
          "type": "double"
        },
        "longitude": {
          "type": "double"
        },
        "country_code": {
          "type": "text"
        },
        "population": {
```

(continues on next page)

(continued from previous page)

```
          "type": "long"
        }
      }
    }
  }
}
```

Note: This tutorial assumes that you want to benchmark a version of Elasticsearch prior to 7.0.0. If you want to benchmark Elasticsearch 7.0.0 or later you need to remove the mapping type above.

For details on the allowed syntax, see the Elasticsearch documentation on mappings and the create index API.

Finally, store the track as `track.json` in the tutorial directory:

```
{
  "version": 2,
  "description": "Tutorial benchmark for Rally",
  "indices": [
    {
      "name": "geonames",
      "body": "index.json",
      "types": [ "docs" ]
    }
  ],
  "corpora": [
    {
      "name": "rally-tutorial",
      "documents": [
        {
          "source-file": "documents.json",
          "document-count": 11658903,
          "uncompressed-bytes": 1544799789
        }
      ]
    }
  ],
  "schedule": [
    {
      "operation": {
        "operation-type": "delete-index"
      }
    },
    {
      "operation": {
        "operation-type": "create-index"
      }
    },
    {
      "operation": {
        "operation-type": "cluster-health",
        "request-params": {
          "wait_for_status": "green"
        }
      }
    },
```

(continues on next page)

(continued from previous page)

```
    {
      "operation": {
        "operation-type": "bulk",
        "bulk-size": 5000
      },
      "warmup-time-period": 120,
      "clients": 8
    },
    {
      "operation": {
        "operation-type": "force-merge"
      }
    },
    {
      "operation": {
        "name": "query-match-all",
        "operation-type": "search",
        "body": {
          "query": {
            "match_all": {}
          }
        }
      },
      "clients": 8,
      "warmup-iterations": 1000,
      "iterations": 1000,
      "target-throughput": 100
    }
  ]
}
```

The numbers under the `documents` property are needed to verify integrity and provide progress reports. Determine the correct document count with `wc -l documents.json` and the size in bytes with `stat -f "%z" documents.json`.

Note: This tutorial assumes that you want to benchmark a version of Elasticsearch prior to 7.0.0. If you want to benchmark Elasticsearch 7.0.0 or later you need to remove the `types` property above.

Note: You can store any supporting scripts along with your track. However, you need to place them in a directory starting with "_", e.g. "_support". Rally loads track plugins (see below) from any directory but will ignore directories starting with "_".

Note: We have defined a JSON schema for tracks which you can use to check how to define your track. You should also check the tracks provided by Rally for inspiration.

The new track appears when you run `esrally list tracks --track-path=~/rally-tracks/tutorial`:

```
dm@io:~ $ esrally list tracks --track-path=~/rally-tracks/tutorial

    ____          ____
```

(continues on next page)

(continued from previous page)

```
   / __ \____ _/ / /_  __
  / /_/ / __ `/ / / / / /
 / _, _/ /_/ / / / /_/ /
/_/ |_|\__,_/_/_/\__, /
                /____/
Available tracks:

Name         Description                    Documents    Compressed Size  Uncompressed␣
↪Size
----------   ------------------------------ -----------  ---------------  --------------
↪---
tutorial     Tutorial benchmark for Rally      11658903  N/A               1.4 GB
```

You can also show details about your track with `esrally info --track-path=~/rally-tracks/tutorial`:

```
dm@io:~ $ esrally info --track-path=~/rally-tracks/tutorial

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

Showing details for track [tutorial]:

* Description: Tutorial benchmark for Rally
* Documents: 11,658,903
* Compressed Size: N/A
* Uncompressed Size: 1.4 GB


Schedule:
----------

1. delete-index
2. create-index
3. cluster-health
4. bulk (8 clients)
5. force-merge
6. query-match-all (8 clients)
```

Congratulations, you have created your first track! You can test it with `esrally --distribution-version=6.0.0 --track-path=~/rally-tracks/tutorial`.

2.9.4 Adding support for test mode

You can check your track very quickly for syntax errors when you invoke Rally with `--test-mode`. Rally postprocesses its internal track representation as follows:

- Iteration-based tasks run at most one warmup iteration and one measurement iteration.
- Time-period-based tasks run at most for 10 seconds without warmup.

Rally also postprocesses all data file names. Instead of `documents.json`, Rally expects `documents-1k.json` and assumes the file contains 1.000 documents. You need to prepare these data files though. Pick 1.000 documents for every data file in your track and store them in a file with the suffix `-1k`. We choose the first 1.000 with `head -n 1000 documents.json > documents-1k.json`.

2.9.5 Challenges

To specify different workloads in the same track you can use so-called challenges. Instead of specifying the `schedule` property on top-level you specify a `challenges` array:

```
{
  "version": 2,
  "description": "Tutorial benchmark for Rally",
  "indices": [
    {
      "name": "geonames",
      "body": "index.json",
      "types": [ "docs" ]
    }
  ],
  "corpora": [
    {
      "name": "rally-tutorial",
      "documents": [
        {
          "source-file": "documents.json",
          "document-count": 11658903,
          "uncompressed-bytes": 1544799789
        }
      ]
    }
  ],
  "challenges": [
    {
      "name": "index-and-query",
      "default": true,
      "schedule": [
        {
          "operation": {
            "operation-type": "delete-index"
          }
        },
        {
          "operation": {
            "operation-type": "create-index"
          }
        },
        {
          "operation": {
            "operation-type": "cluster-health",
            "request-params": {
              "wait_for_status": "green"
            }
          }
        },
        {
          "operation": {
            "operation-type": "bulk",
            "bulk-size": 5000
```

(continues on next page)

(continued from previous page)

```
          },
          "warmup-time-period": 120,
          "clients": 8
        },
        {
          "operation": {
            "operation-type": "force-merge"
          }
        },
        {
          "operation": {
            "name": "query-match-all",
            "operation-type": "search",
            "body": {
              "query": {
                "match_all": {}
              }
            }
          },
          "clients": 8,
          "warmup-iterations": 1000,
          "iterations": 1000,
          "target-throughput": 100
        }
      ]
    }
  ]
}
```

Note: If you define multiple challenges, Rally runs the challenge where `default` is set to `true`. If you want to run a different challenge, provide the command line option `--challenge=YOUR_CHALLENGE_NAME`.

When should you use challenges? Challenges are useful when you want to run completely different workloads based on the same track but for the majority of cases you should get away without using challenges:

- To run only a subset of the tasks, you can use *task filtering*, e.g. `--include-tasks="create-index,bulk"` will only run these two tasks in the track above or `--exclude-tasks="bulk"` will run all tasks except for `bulk`.
- To vary parameters, e.g. the number of clients, you can use *track parameters*

2.9.6 Structuring your track

`track.json` is the entry point to a track but you can split your track as you see fit. Suppose you want to add more challenges to the track but keep them in separate files. Create a `challenges` directory and store the following in `challenges/index-and-query.json`:

```
{
  "name": "index-and-query",
  "default": true,
  "schedule": [
    {
      "operation": {
        "operation-type": "delete-index"
```

(continues on next page)

(continued from previous page)

```
        }
      },
      {
        "operation": {
          "operation-type": "create-index"
        }
      },
      {
        "operation": {
          "operation-type": "cluster-health",
          "request-params": {
            "wait_for_status": "green"
          }
        }
      },
      {
        "operation": {
          "operation-type": "bulk",
          "bulk-size": 5000
        },
        "warmup-time-period": 120,
        "clients": 8
      },
      {
        "operation": {
          "operation-type": "force-merge"
        }
      },
      {
        "operation": {
          "name": "query-match-all",
          "operation-type": "search",
          "body": {
            "query": {
              "match_all": {}
            }
          }
        },
        "clients": 8,
        "warmup-iterations": 1000,
        "iterations": 1000,
        "target-throughput": 100
      }
    ]
}
```

Include the new file in `track.json`:

```
{
  "version": 2,
  "description": "Tutorial benchmark for Rally",
  "indices": [
    {
      "name": "geonames",
      "body": "index.json",
      "types": [ "docs" ]
    }
```

(continues on next page)

(continued from previous page)

```
  ],
  "corpora": [
    {
      "name": "rally-tutorial",
      "documents": [
        {
          "source-file": "documents.json",
          "document-count": 11658903,
          "uncompressed-bytes": 1544799789
        }
      ]
    }
  ],
  "challenges": [
    {% include "challenges/index-and-query.json" %}
  ]
}
```

We replaced the challenge content with `{% include "challenges/index-and-query.json" %}` which tells Rally to include the challenge from the provided file. You can use `include` on arbitrary parts of your track.

To reuse operation definitions across challenges, you can define them in a separate `operations` block and refer to them by name in the corresponding challenge:

```
{
  "version": 2,
  "description": "Tutorial benchmark for Rally",
  "indices": [
    {
      "name": "geonames",
      "body": "index.json",
      "types": [ "docs" ]
    }
  ],
  "corpora": [
    {
      "name": "rally-tutorial",
      "documents": [
        {
          "source-file": "documents.json",
          "document-count": 11658903,
          "uncompressed-bytes": 1544799789
        }
      ]
    }
  ],
  "operations": [
    {
      "name": "delete",
      "operation-type": "delete-index"
    },
    {
      "name": "create",
      "operation-type": "create-index"
    },
    {
      "name": "wait-for-green",
```

(continues on next page)

(continued from previous page)

```
      "operation-type": "cluster-health",
      "request-params": {
        "wait_for_status": "green"
      }
    },
    {
      "name": "bulk-index",
      "operation-type": "bulk",
      "bulk-size": 5000
    },
    {
      "name": "force-merge",
      "operation-type": "force-merge"
    },
    {
      "name": "query-match-all",
      "operation-type": "search",
      "body": {
        "query": {
          "match_all": {}
        }
      }
    }
  ],
  "challenges": [
    {% include "challenges/index-and-query.json" %}
  ]
}
```

`challenges/index-and-query.json` then becomes:

```
{
  "name": "index-and-query",
  "default": true,
  "schedule": [
    {
      "operation": "delete"
    },
    {
      "operation": "create"
    },
    {
      "operation": "wait-for-green"
    },
    {
      "operation": "bulk-index",
      "warmup-time-period": 120,
      "clients": 8
    },
    {
      "operation": "force-merge"
    },
    {
      "operation": "query-match-all",
      "clients": 8,
      "warmup-iterations": 1000,
      "iterations": 1000,
```

(continues on next page)

(continued from previous page)

```
      "target-throughput": 100
    }
  ]
}
```

Note how we reference to the operations by their name (e.g. `create`, `bulk-index`, `force-merge` or `query-match-all`).

You can also use Rally's collect helper to simplify including multiple challenges:

```
{% import "rally.helpers" as rally %}
{
  "version": 2,
  "description": "Tutorial benchmark for Rally",
  "indices": [
    {
      "name": "geonames",
      "body": "index.json",
      "types": [ "docs" ]
    }
  ],
  "corpora": [
    {
      "name": "rally-tutorial",
      "documents": [
        {
          "source-file": "documents.json",
          "document-count": 11658903,
          "uncompressed-bytes": 1544799789
        }
      ]
    }
  ],
  "operations": [
    {
      "name": "delete",
      "operation-type": "delete-index"
    },
    {
      "name": "create",
      "operation-type": "create-index"
    },
    {
      "name": "wait-for-green",
      "operation-type": "cluster-health",
      "request-params": {
        "wait_for_status": "green"
      }
    },
    {
      "name": "bulk-index",
      "operation-type": "bulk",
      "bulk-size": 5000
    },
    {
      "name": "force-merge",
      "operation-type": "force-merge"
```

(continues on next page)

(continued from previous page)

```
    },
    {
      "name": "query-match-all",
      "operation-type": "search",
      "body": {
        "query": {
          "match_all": {}
        }
      }
    }
  ],
  "challenges": [
    {{ rally.collect(parts="challenges/*.json") }}
  ]
}
```

The changes are:

1. We import helper functions from Rally by adding `{% import "rally.helpers" as rally %}` in line 1.
2. We use Rally's `collect` helper to find and include all JSON files in the `challenges` subdirectory with the statement `{{ rally.collect(parts="challenges/*.json") }}`.

Note: Rally's log file contains the fully rendered track after it has loaded it successfully.

You can even use Jinja2 variables but then you need to import the Rally helpers a bit differently. You also need to declare all variables before the `import` statement:

```
{% set clients = 16 %}
{% import "rally.helpers" as rally with context %}
```

If you use this idiom you can refer to the `clients` variable inside your snippets with `{{ clients }}`.

2.9.7 Sharing your track with others

So far the track is only available on your local machine. To share your track you could check it into version control. To avoid committing the potentially huge data file you can expose it via http (e.g. via an S3 bucket) and reference it in your track with the property `base-url`. Rally expects that the URL points to the parent path and appends the document file name automatically.

You should also compress your document corpus to save network bandwidth; pbzip2 works well, is backwards compatible with `bzip2` and makes use of all available cpu cores for compression and decompression. You can create a compressed archive with the following command:

```
pbzip2 -9 -k -m2000 -v documents.json
```

If you want to support Rally's test mode, also compress your test mode corpus with:

```
pbzip2 -9 -k -m2000 -v documents-1k.json
```

Then upload the generated archives `documents.json.bz2` and `documents-1k.json.bz2` to the remote location.

Finally, specify the compressed file name in the `source-file` property and also add the `base-url` property:

```
{
  "version": 2,
  "description": "Tutorial benchmark for Rally",
  "corpora": [
    {
      "name": "rally-tutorial",
      "documents": [
        {
          "base-url": "http://benchmarks.elasticsearch.org.s3.amazonaws.com/corpora/
↪geonames",
          "source-file": "documents.json.bz2",
          "document-count": 11658903,
          "compressed-bytes": 197857614,
          "uncompressed-bytes": 1544799789
        }
      ]
    }
  ],
  ...
}
```

Specifying `compressed-bytes` (file size of `documents.json.bz2`) and `uncompressed-bytes` (file size of `documents.json`) is optional but helps Rally to provide progress indicators and also verify integrity.

You've now mastered the basics of track development for Rally. It's time to pat yourself on the back before you dive into the advanced topics!

2.9.8 Advanced topics

Template Language

Rally uses Jinja2 as a template language so you can use Jinja2 expressions in track files.

Elasticsearch utilizes Mustache formatting in a few places, notably in search templates and Watcher templates. If you are using Mustache in your Rally tracks you must escape them properly. See *put-pipeline* for an example.

Extensions

Rally also provides a few extensions to Jinja2:

- `now`: a global variable that represents the current date and time when the template is evaluated by Rally.
- `days_ago()`: a filter that you can use for date calculations.

You can find an example in the `http_logs` track:

```
{
  "name": "range",
    "index": "logs-*",
    "type": "type",
    "body": {
      "query": {
        "range": {
          "@timestamp": {
            "gte": "now-{{'15-05-1998' | days_ago(now)}}d/d",
            "lt": "now/d"
```

(continues on next page)

(continued from previous page)

```
          }
        }
      }
    }
  }
}
```

The data set that is used in the `http_logs` track starts on 26-04-1998 but we want to ignore the first few days for this query, so we start on 15-05-1998. The expression `{{'15-05-1998' | days_ago(now)}}` yields the difference in days between now and the fixed start date and allows us to benchmark time range queries relative to now with a predetermined data set.

- `rally.collect(parts)`: a macro that you can use to join track fragments. See the *example above*.
- `rally.exists_set_param(setting_name, value, default_value=None, comma=True)`: a macro that you can use to set the value of a track parameter without having to check if it exists.

Important: To use macros you must declare `{% import "rally.helpers" as rally with context %}` at the top of your track; see *the docs* for more details and the geonames track for an example.

Example:

Suppose you need an operation that specifies the Elasticsearch transient setting `indices.recovery.max_bytes_per_sec` if and only if it has been provided as a track parameter.

Your operation could look like:

```
{
  "operation": {
    "operation-type": "raw-request",
    "method": "PUT",
    "path": "/_cluster/settings",
    "body": {
      "transient": {
        "cluster.routing.allocation.node_initial_primaries_recoveries": 8
        {{ rally.exists_set_param("indices.recovery.max_bytes_per_sec", es_snapshot_
↪restore_recovery_max_bytes_per_sec) }}
      }
    }
  }
}
```

Note the lack of a comma after the first setting `cluster.routing.allocation.node_initial_primaries_recoveries`. This is intentional since the helper will insert it if the parameter exists (this behavior can be changed using `comma=False`).

Assuming we pass `--track-params="es_snapshot_restore_recovery_max_bytes_per_sec:-1"` the helper will end up rendering the operation as:

```
{
  "operation": {
    "operation-type": "raw-request",
    "method": "PUT",
    "path": "/_cluster/settings",
    "body": {
```

(continues on next page)

(continued from previous page)

```
      "transient": {
        "cluster.routing.allocation.node_initial_primaries_recoveries": 8,"indices.
↪recovery.max_bytes_per_sec": -1
      }
    }
  }
}
```

The parameter `default_value` controls the value to use for the setting if it is undefined. If the setting is undefined and there is no default value, nothing will be added.

Custom parameter sources

Warning: Your parameter source is on a performance-critical code-path. Double-check with *Rally's profiling support* that you did not introduce any bottlenecks.

Consider the following operation definition:

```
{
  "name": "term",
  "operation-type": "search",
  "body": {
    "query": {
      "term": {
        "body": "physician"
      }
    }
  }
}
```

This query is defined statically but if you want to vary parameters, for example to search also for "mechanic" or "nurse, you can write your own "parameter source" in Python.

First, define the name of your parameter source in the operation definition:

```
{
  "name": "term",
  "operation-type": "search",
  "param-source": "my-custom-term-param-source"
  "professions": ["mechanic", "physician", "nurse"]
}
```

Rally recognizes the parameter source and looks for a file `track.py` next to `track.json`. This file contains the implementation of the parameter source:

```
import random


def random_profession(track, params, **kwargs):
    # choose a suitable index: if there is only one defined for this track
    # choose that one, but let the user always override index and type.
    if len(track.indices) == 1:
        default_index = track.indices[0].name
```

(continues on next page)

(continued from previous page)

```
        if len(track.indices[0].types) == 1:
            default_type = track.indices[0].types[0].name
        else:
            default_type = None
    else:
        default_index = "_all"
        default_type = None

    index_name = params.get("index", default_index)
    type_name = params.get("type", default_type)

    # you must provide all parameters that the runner expects
    return {
        "body": {
            "query": {
                "term": {
                    "body": "%s" % random.choice(params["professions"])
                }
            }
        },
        "index": index_name,
        "type": type_name,
        "cache": params.get("cache", False)
    }

def register(registry):
    registry.register_param_source("my-custom-term-param-source", random_profession)
```

The example above shows a simple case that is sufficient if the operation to which your parameter source is applied is idempotent and it does not matter whether two clients execute the same operation.

The function `random_profession` is the actual parameter source. Rally will bind the name "my-custom-term-param-source" to this function by calling `register`. `register` is called by Rally before the track is executed.

The parameter source function needs to declare the parameters `track`, `params` and `**kwargs`. `track` contains a structured representation of the current track and `params` contains all parameters that have been defined in the operation definition in `track.json`. We use it in the example to read the professions to choose. The third parameter is there to ensure a more stable API as Rally evolves.

We also derive an appropriate index and document type from the track's index definitions but allow the user to override this choice with the `index` or `type` parameters:

```
{
  "name": "term",
  "operation-type": "search",
  "param-source": "my-custom-term-param-source"
  "professions": ["mechanic", "physician", "nurse"],
  "index": "employee*",
  "type": "docs"
}
```

If you need more control, you need to implement a class. Below is the implementation of the same parameter source as a class:

```
import random
```

(continues on next page)

(continued from previous page)

```python
class TermParamSource:
    def __init__(self, track, params, **kwargs):
        # choose a suitable index: if there is only one defined for this track
        # choose that one, but let the user always override index and type.
        if len(track.indices) == 1:
            default_index = track.indices[0].name
            if len(track.indices[0].types) == 1:
                default_type = track.indices[0].types[0].name
            else:
                default_type = None
        else:
            default_index = "_all"
            default_type = None

        # we can eagerly resolve these parameters already in the constructor...
        self._index_name = params.get("index", default_index)
        self._type_name = params.get("type", default_type)
        self._cache = params.get("cache", False)
        # ... but we need to resolve "profession" lazily on each invocation later
        self._params = params
        # Determines whether this parameter source will be "exhausted" at some point␣
↪or
        # Rally can draw values infinitely from it.
        self.infinite = True

    def partition(self, partition_index, total_partitions):
        return self

    def params(self):
        # you must provide all parameters that the runner expects
        return {
            "body": {
                "query": {
                    "term": {
                        "body": "%s" % random.choice(self._params["professions"])
                    }
                }
            },
            "index": self._index_name,
            "type": self._type_name,
            "cache": self._cache
        }


def register(registry):
    registry.register_param_source("my-custom-term-param-source", TermParamSource)
```

In `register` you bind the name in the track specification to your parameter source implementation class similar to the previous example. `TermParamSource` is the actual parameter source and needs to fulfill a few requirements:

- The constructor needs to have the signature `__init__(self, track, params, **kwargs)`.
- `partition(self, partition_index, total_partitions)` is called by Rally to "assign" the parameter source across multiple clients. Typically you can just return `self`. If each client needs to act differently then you can provide different parameter source instances here as well.
- `params(self)`: This method returns a dictionary with all parameters that the corresponding "runner" expects. This method will be invoked once for every iteration during the race. In the example, we parameterize the query

by randomly selecting a profession from a list.

- `infinite`: This property helps Rally to determine whether to let the parameter source determine when a task should be finished (when `infinite` is `False`) or whether the task properties (e.g. `iterations` or `time-period`) determine when a task should be finished. In the former case, the parameter source needs to raise `StopIteration` to indicate when it is finished.

For cases, where you want to provide a progress indication (this is typically the case when `infinite` is `False`), you can implement a property `percent_completed` which returns a floating point value between `0.0` and `1.0`. Rally will query this value before each call to `params()` and uses it to indicate progress. However:

- Rally will not check `percent_completed` if it can derive progress in any other way.
- The value of `percent_completed` is purely informational and does not influence when Rally considers an operation to be completed.

Note: The method `params(self)` as well as the property `percent_completed` are called on a performance-critical path. Don't do anything that takes a lot of time (avoid any I/O). For searches, you should usually throttle throughput anyway and there it does not matter that much but if the corresponding operation is run without throughput throttling, double-check that your custom parameter source does not introduce a bottleneck.

Custom parameter sources can use the Python standard API but using any additional libraries is not supported.

You can also implement your parameter sources and runners in multiple Python files but the main entry point is always `track.py`. The root package name of your plugin is the name of your track.

Custom runners

Warning: Your runner is on a performance-critical code-path. Double-check with *Rally's profiling support* that you did not introduce any bottlenecks.

Runners execute an operation against Elasticsearch. Rally supports many operations out of the box already, see the *track reference* for a complete list. If you want to call any other Elasticsearch API, define a custom runner.

Consider we want to use the percolate API with an older version of Elasticsearch which is not supported by Rally. To achieve this, we implement a custom runner in the following steps.

In `track.json` set the `operation-type` to "percolate" (you can choose this name freely):

```
{
  "name": "percolator_with_content_google",
  "operation-type": "percolate",
  "body": {
    "doc": {
      "body": "google"
    },
    "track_scores": true
  }
}
```

Then create a file `track.py` next to `track.json` and implement the following two functions:

```
async def percolate(es, params):
    await es.percolate(
            index="queries",
```

(continues on next page)

(continued from previous page)

```
            doc_type="content",
            body=params["body"]
        )

def register(registry):
    registry.register_runner("percolate", percolate, async_runner=True)
```

The function `percolate` is the actual runner and takes the following parameters:

- `es`, is an instance of the Elasticsearch Python client
- `params` is a `dict` of parameters provided by its corresponding parameter source. Treat this parameter as read-only.

This function can return:

- Nothing at all. Then Rally will assume by default `1` and `"ops"` (see below).
- A tuple of `weight` and a `unit`, which is usually `1` and `"ops"`. If you run a bulk operation you might return the bulk size here, for example in number of documents or in MB. Then you'd return for example `(5000, "docs")` Rally will use these values to store throughput metrics.
- A `dict` with arbitrary keys. If the `dict` contains the key `weight` it is assumed to be numeric and chosen as weight as defined above. The key `unit` is treated similarly. All other keys are added to the `meta` section of the corresponding service time and latency metrics records.

Similar to a parameter source you also need to bind the name of your operation type to the function within `register`.

To illustrate how to use custom return values, suppose we want to implement a custom runner that calls the pending tasks API and returns the number of pending tasks as additional meta-data:

```
async def pending_tasks(es, params):
    response = await es.cluster.pending_tasks()
    return {
        "weight": 1,
        "unit": "ops",
        "pending-tasks-count": len(response["tasks"])
    }

def register(registry):
    registry.register_runner("pending-tasks", pending_tasks, async_runner=True)
```

If you need more control, you can also implement a runner class. The example above, implemented as a class looks as follows:

```
class PercolateRunner:
    async def __call__(self, es, params):
        await es.percolate(
            index="queries",
            doc_type="content",
            body=params["body"]
        )

    def __repr__(self, *args, **kwargs):
        return "percolate"

def register(registry):
    registry.register_runner("percolate", PercolateRunner(), async_runner=True)
```

The actual runner is implemented in the method `__call__` and the same return value conventions apply as for functions. For debugging purposes you should also implement `__repr__` and provide a human-readable name for your runner. Finally, you need to register your runner in the `register` function.

Runners also support Python's asynchronous context manager interface. Rally uses a new context for each request. Implementing the asynchronous context manager interface can be handy for cleanup of resources after executing an operation. Rally uses it, for example, to clear open scrolls.

If you have specified multiple Elasticsearch clusters using *target-hosts* you can make Rally pass a dictionary of client connections instead of one for the `default` cluster in the `es` parameter.

To achieve this you need to:

- Use a runner class
- Specify `multi_cluster = True` as a class attribute
- Use any of the cluster names specified in *target-hosts* as a key for the `es` dict

Example (assuming Rally has been invoked specifying `default` and `remote` in *target-hosts*):

```
class CreateIndexInRemoteCluster:
    multi_cluster = True

    async def __call__(self, es, params):
        await es["remote"].indices.create(index="remote-index")

    def __repr__(self, *args, **kwargs):
        return "create-index-in-remote-cluster"

def register(registry):
    registry.register_runner("create-index-in-remote-cluster",␣
↪CreateIndexInRemoteCluster(), async_runner=True)
```

Note: You need to implement `register` just once and register all parameter sources and runners there.

For cases, where you want to provide a progress indication, you can implement the two properties `percent_completed` which returns a floating point value between `0.0` and `1.0` and the property `completed` which needs to return `True` if the runner has completed. This can be useful in cases when it is only possible to determine progress by calling an API, for example when waiting for a recovery to finish.

Warning: Rally will still treat such a runner like any other. If you want to poll status at certain intervals then limit the number of calls by specifying the `target-throughput` property on the corresponding task.

Custom schedulers

Warning: Your scheduler is on a performance-critical code-path. Double-check with *Rally's profiling support* that you did not introduce any bottlenecks.

If you want to rate-limit execution of tasks, you can specify a `target-throughput` (in operations per second). For example, Rally attempts to run this term query 20 times per second:

```
{
  "operation": "term",
  "target-throughput": 20
}
```

By default, Rally uses a deterministic distribution to determine when to schedule the next operation. Hence it executes the term query at 0, 50ms, 100ms, 150ms and so on. The scheduler is also aware of the number of clients. Consider this example:

```
{
  "operation": "term",
  "target-throughput": 20,
  "clients": 4
}
```

If Rally would not take the number of clients into account and would still issue requests (from each of the four clients) at the same points in time (i.e. 0, 50ms, 100ms, 150ms, . . .), it would run at a target throughput of 4 * 20 = 80 operations per second. Hence, Rally will automatically reduce the rate at which each client will execute requests. Each client will issue requests at 0, 200ms, 400ms, 600ms, 800ms, 1000ms and so on. Each client issues five requests per second but as there are four of them, we still have a target throughput of 20 operations per second. You should keep this in mind, when writing your own custom schedules.

To create a custom scheduler, create a file `track.py` next to `track.json` and implement the following two functions:

```
import random

def random_schedule(current):
    return current + random.randint(10, 900) / 1000.0


def register(registry):
    registry.register_scheduler("my_random", random_schedule)
```

You can then use your custom scheduler as follows:

```
{
  "operation": "term",
  "schedule": "my_random"
}
```

The function `random_schedule` returns a floating point number which represents the next point in time when Rally should execute the given operation. This point in time is measured in seconds relative to the beginning of the execution of this task. The parameter `current` is the last return value of your function and is `0.0` for the first invocation. So, for example, this scheduler could return the following series: 0, 0.119, 0.622, 1.29, 1.343, 1.984, 2.233.

This implementation is usually not sufficient as it does not take into account the number of clients. Therefore, you typically want to implement a full-blown scheduler which can also take parameters. Below is an example for our random scheduler:

```
import random

class RandomScheduler:
    def __init__(self, params):
        # assume one client by default
        clients = params.get("clients", 1)
        # scale accordingly with the number of clients!
```

(continues on next page)

(continued from previous page)

```
        self.lower_bound = clients * params.get("lower-bound-millis", 10)
        self.upper_bound = clients * params.get("upper-bound-millis", 900)

    def next(self, current):
        return current + random.randint(self.lower_bound, self.upper_bound) / 1000.0


def register(registry):
    registry.register_scheduler("my_random", RandomScheduler)
```

This implementation achieves the same rate independent of the number of clients. Additionally, we can pass the lower and upper bound for the random function from the track:

```
{
    "operation": "term",
    "schedule": "my_random",
    "clients": 4,
    "lower-bound-millis": 50,
    "upper-bound-millis": 250
}
```

2.10 Developing Rally

2.10.1 Prerequisites

Install the following software packages:

- Pyenv installed and `eval "$(pyenv init -)"` is added to the shell configuration file. For more details please refer to the PyEnv installation instructions.
- JDK version required to build Elasticsearch. Please refer to the build setup requirements.
- Docker and on Linux additionally docker-compose.
- git 1.9 or better

Check the *installation guide* for detailed installation instructions for these packages.

Rally does not support Windows and is only actively tested on MacOS and Linux.

2.10.2 Installation Instructions for Development

```
git clone https://github.com/elastic/rally.git
cd rally
make prereq
make install
source .venv/bin/activate
./rally
```

Known Issues

If you get errors during installation, it is probably due to the installation of `psutil` which we use to gather system metrics like CPU utilization. Check the installation instructions of psutil in this case. Keep in mind that Rally is based on Python 3 and you need to install the Python 3 header files instead of the Python 2 header files on Linux.

On MacOS Mojave the step `make prereq` might fail with the following message:

```
zipimport.ZipImportError: can't decompress data; zlib not available
```

This is a known issue with `pyenv`. Please see the Github issue for workarounds.

Automatic Updates

Rally has a built-in auto-update feature when you install it from sources. By default, it will update from the remote named `origin`. If you want to auto-update from a different remote, provide `--update-from-remote=YOUR_REMOTE_NAME` as first parameter.

To work conveniently with Rally, we suggest that you add the Rally project directory to your `PATH`. In case you use a different remote, you should also define aliases in your shell's config file, e.g.:

```
alias rally='rally --update-from-remote=elastic '
alias rallyd='rallyd --update-from-remote=elastic '
```

Then you can invoke Rally or the *Rally daemon* as usual and have auto-update still work.

Also note that automatic updates are disabled in the following cases:

- There are local (uncommitted) changes in the Rally project directory
- A different branch than `master` is checked out
- You have specified `--skip-update` as the first command line parameter
- You have specified `--offline` as a command line parameter for Rally

Configuring Rally

Before we can run our first benchmark, we have to configure Rally. Just invoke `./rally configure` and Rally will automatically detect that its configuration file is missing and prompt you for some values and write them to `~/.rally/rally.ini`. After you've configured Rally, it will exit.

For more information see *configuration help page*.

2.10.3 Key Components of Rally

To get a rough understanding of Rally, it makes sense to get to know its key components:

- *Race Control*: is responsible for proper execution of the race. It sets up all components and acts as a high-level controller.
- *Mechanic*: can build and prepare a benchmark candidate for the race. It checks out the source, builds Elasticsearch, provisions and starts the cluster.
- *Track*: is a concrete benchmarking scenario, e.g. the http_logs benchmark. It defines the data set to use.
- *Challenge*: is the specification on what benchmarks should be run and its configuration (e.g. index, then run a search benchmark with 1000 iterations)
- *Car*: is a concrete system configuration for a benchmark, e.g. an Elasticsearch single-node cluster with default settings.
- *Driver*: drives the race, i.e. it is executing the benchmark according to the track specification.

- *Reporter*: A reporter tells us how the race went (currently only after the fact).

There is a dedicated *tutorial on how to add new tracks to Rally*.

2.10.4 How to contribute code

See the contributors guide. We strive to be PEP-8 compliant but don't follow it to the letter.

2.11 Elasticsearch Version Support

2.11.1 Minimum supported version

Rally 2.0.2 can benchmark Elasticsearch 2.0.0 and above.

2.11.2 End-of-life Policy

The latest version of Rally allows to benchmark all currently supported versions of Elasticsearch. Once an Elasticsearch version reaches end-of-life, Rally will support benchmarking the corresponding version for two more years. For example, Elasticsearch 1.7.x has been supported until 2017-01-16. Rally drops support for Elasticsearch 1.7.x two years after that date on 2019-01-16. Version support is dropped in the next Rally maintenance release after that date.

2.11.3 Version-specific Limitations

The following features require at least Elasticsearch 5.0.0:

- *Benchmarking with plugins*
- *Benchmarking source builds*

2.12 Command Line Reference

You can control Rally with subcommands and command line flags:

- Subcommands determine which task Rally performs.
- Command line flags are used to change Rally's behavior but not all command line flags can be used for each subcommand. To find out which command line flags are supported by a specific subcommand, just run `esrally <<subcommand>> --help`.

2.12.1 Subcommands

`race`

The `race` subcommand is used to actually run a benchmark. It is the default one and chosen implicitly if none is given.

list

The list subcommand is used to list different configuration options:

- telemetry: Will show all *telemetry devices* that are supported by Rally.
- tracks: Will show all tracks that are supported by Rally. As this *may* depend on the Elasticsearch version that you want to benchmark, you can specify `--distribution-version` and also `--distribution-repository` as additional options.
- pipelines: Will show all *pipelines* that are supported by Rally.
- races: Will show a list of the most recent races. This is needed for the *tournament mode*.
- cars: Will show all cars that are supported by Rally (i.e. Elasticsearch configurations).
- elasticsearch-plugins: Will show all Elasticsearch plugins and their configurations that are supported by Rally.

To list a specific configuration option, place it after the `list` subcommand. For example, `esrally list pipelines` will list all pipelines known to Rally.

info

The `info` subcommand prints details about a track. Example:

```
esrally info --track=noaa --challenge=append-no-conflicts
```

This will print:

```
Showing details for track [noaa]:

* Description: Global daily weather measurements from NOAA
* Documents: 33,659,481
* Compressed Size: 947.3 MB
* Uncompressed Size: 9.0 GB

================================================
Challenge [append-no-conflicts] (run by default)
================================================

Indexes the whole document corpus using Elasticsearch default settings. We only␣
↪adjust the number of replicas as we benchmark a single node cluster and Rally will␣
↪only start the benchmark if the cluster turns green and we want to ensure that we␣
↪don't use the query cache. Document ids are unique so all index operations are␣
↪append only. After that a couple of queries are run.

Schedule:
----------

1. delete-index
2. create-index
3. check-cluster-health
4. index (8 clients)
5. refresh-after-index
6. force-merge
7. refresh-after-force-merge
8. range_field_big_range
9. range_field_small_range
10. range_field_conjunction_big_range_small_term_query
```

(continues on next page)

(continued from previous page)

```
11. range_field_conjunction_small_range_small_term_query
12. range_field_conjunction_small_range_big_term_query
13. range_field_conjunction_big_range_big_term_query
14. range_field_disjunction_small_range_small_term_query
15. range_field_disjunction_big_range_small_term_query
```

It is also possible to use task filters (e.g. `--include-tasks`) or to refer to a track via its path (`--track-path`) or use a different track repository (`--track-repository`).

`compare`

This subcommand is needed for *tournament mode* and its usage is described there.

`configure`

This subcommand is needed to *configure* Rally. It is implicitly chosen if you start Rally for the first time but you can rerun this command at any time.

`create-track`

This subcommand creates a basic track from data in an existing cluster. See the *track tutorial* for a complete walk-through.

`download`

This subcommand can be used to download Elasticsearch distributions. Example:

```
esrally download --distribution-version=6.8.0 --quiet
```

This will download the OSS distribution of Elasticsearch 6.8.0. Because `--quiet` is specified, Rally will suppress all non-essential output (banners, progress messages etc.) and only return the location of the binary on the local machine after it has downloaded it:

```
{
  "elasticsearch": "/Users/dm/.rally/benchmarks/distributions/elasticsearch-oss-6.8.0.
↪tar.gz"
}
```

To download the default distribution you need to specify a license (via `--car`):

```
esrally download --distribution-version=6.8.0 --car=basic-license --quiet
```

This will show the path to the default distribution:

```
{
  "elasticsearch": "/Users/dm/.rally/benchmarks/distributions/elasticsearch-6.8.0.tar.
↪gz"
}
```

`install`

Warning: This subcommand is experimental. Expect the functionality and the command line interface to change significantly even in patch releases.

This subcommand can be used to install a single Elasticsearch node. Example:

```
esrally install --quiet --distribution-version=7.4.2 --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"
```

This will output the id of this installation:

```
{
  "installation-id": "69ffcfee-6378-4090-9e93-87c9f8ee59a7"
}
```

Please see the *cluster management tutorial* for the intended use and a complete walkthrough.

`start`

Warning: This subcommand is experimental. Expect the functionality and the command line interface to change significantly even in patch releases.

This subcommand can be used to start a single Elasticsearch node that has been previously installed with the `install` subcommand. Example:

```
esrally start --installation-id=INSTALLATION_ID --race-id=RACE_ID
```

`INSTALLATION_ID` is the installation id value as shown in the output of the `install` command. The `RACE_ID` can be chosen freely but is required to be the same for all nodes in a cluster.

Please see the *cluster management tutorial* for the intended use and a complete walkthrough.

`stop`

Warning: This subcommand is experimental. Expect the functionality and the command line interface to change significantly even in patch releases.

This subcommand can be used to stop a single Elasticsearch node that has been previously started with the `start` subcommand. Example:

```
esrally stop --installation-id=INSTALLATION_ID
```

`INSTALLATION_ID` is the installation id value as shown in the output of the `install` command.

Please see the *cluster management tutorial* for the intended use and a complete walkthrough.

2.12.2 Command Line Flags

`track-path`

Can be either a directory that contains a `track.json` file or a `.json` file with an arbitrary name that contains a track specification. `--track-path` and `--track-repository` as well as `--track` are mutually exclusive. See the *track reference* to decide whether you should use `--track-path` or `--track-repository` / `--track`.

Examples:

```
# provide a directory - Rally searches for a track.json file in this directory
# Track name is "app-logs"
esrally --track-path=~/Projects/tracks/app-logs
# provide a file name - Rally uses this file directly
# Track name is "syslog"
esrally --track-path=~/Projects/tracks/syslog.json
```

`track-repository`

Selects the track repository that Rally should use to resolve tracks. By default the `default` track repository is used, which is available in the Github project rally-tracks. See the *track reference* on how to add your own track repositories. `--track-path` and `--track-repository` as well as `--track` are mutually exclusive.

`track-revision`

Selects a specific revision in the track repository. By default, Rally will choose the most appropriate branch on its own but in some cases it is necessary to specify a certain commit. This is mostly needed when testing whether a change in performance has occurred due to a change in the workload.

`track`

Selects the track that Rally should run. By default the `geonames` track is run. For more details on how tracks work, see *adding tracks* or the *track reference*. `--track-path` and `--track-repository` as well as `--track` are mutually exclusive.

`track-params`

With this parameter you can inject variables into tracks. The supported variables depend on the track and you should check the track JSON file to see which variables can be provided.

It accepts a list of comma-separated key-value pairs or a JSON file name. The key-value pairs have to be delimited by a colon.

Examples:

Consider the following track snippet showing a single challenge:

```
{
  "name": "index-only",
  "schedule": [
     {
       "operation": {
         "operation-type": "bulk",
```

(continues on next page)

(continued from previous page)

```
          "bulk-size": {{ bulk_size|default(5000) }}
        },
        "warmup-time-period": 120,
        "clients": {{ clients|default(8) }}
      }
   ]
}
```

Rally tracks can use the Jinja templating language and the construct `{{ some_variable|default(0) }}` that you can see above is a feature of Jinja to define default values for variables.

We can see that it defines two variables:

- `bulk_size` with a default value of 5000
- `clients` with a default value of 8

When we run this track, we can override these defaults:

- `--track-params="bulk_size:2000,clients:16"` will set the bulk size to 2000 and the number of clients for bulk indexing to 16.
- `--track-params="bulk_size:8000"` will just set the bulk size to 8000 and keep the default value of 8 clients.
- `--track-params="params.json"` will read the track parameters from a JSON file (defined below)

Example JSON file:

```
{
   "bulk_size": 2000,
   "clients": 16
}
```

All track parameters are recorded for each metrics record in the metrics store. Also, when you run `esrally list races`, it will show all track parameters:

```
Race Timestamp    Track    Track Parameters          Challenge            Car      ␣
↪User Tag
----------------  -------  ------------------------- -------------------- --------  --
↪-------
20160518T122341Z  pmc      bulk_size=8000            append-no-conflicts  defaults
20160518T112341Z  pmc      bulk_size=2000,clients=16 append-no-conflicts  defaults
```

Note that the default values are not recorded or shown (Rally does not know about them).

challenge

A track consists of one or more challenges. With this flag you can specify which challenge should be run. If you don't specify a challenge, Rally derives the default challenge itself. To see the default challenge of a track, run `esrally list tracks`.

race-id

A unique identifier for this race. By default a random UUID is automatically chosen by Rally.

`installation-id`

Warning: This command line parameter is experimental. Expect the functionality and the command line interface to change significantly even in patch releases.

A unique identifier for an installation. This id is automatically generated by Rally when the `install` subcommand is invoked and needs to be provided in the `start` and `stop` subcommands in order to refer to that installation.

`node-name`

Warning: This command line parameter is experimental. Expect the functionality and the command line interface to change significantly even in patch releases.

Used to specify the current node's name in the cluster when it is setup via the `install` subcommand.

`network-host`

Warning: This command line parameter is experimental. Expect the functionality and the command line interface to change significantly even in patch releases.

Used to specify the IP address to which the current cluster node should bind to when it is setup via the `install` subcommand.

`http-port`

Warning: This command line parameter is experimental. Expect the functionality and the command line interface to change significantly even in patch releases.

Used to specify the port on which the current cluster node should listen for HTTP traffic when it is setup via the `install` subcommand. The corresponding transport port will automatically be chosen by Rally and is always the specified HTTP port + 100.

`master-nodes`

Warning: This command line parameter is experimental. Expect the functionality and the command line interface to change significantly even in patch releases.

A comma-separated list that specifies the node names of the master nodes in the cluster when a node is setup via the `install` subcommand. Example:

```
--master-nodes="rally-node-0,rally-node-1,rally-node-2"
```

This will treat the nodes named `rally-node-0`, `rally-node-1` and `rally-node-2` as initial master nodes.

seed-hosts

Warning: This command line parameter is experimental. Expect the functionality and the command line interface to change significantly even in patch releases.

A comma-separated list if IP:transport port pairs used to specify the seed hosts in the cluster when a node is setup via the `install` subcommand. See the Elasticsearch documentation for a detailed explanation of the seed hosts parameter. Example:

```
--seed-hosts="192.168.14.77:39300,192.168.14.78:39300,192.168.14.79:39300"
```

include-tasks

Each challenge consists of one or more tasks but sometimes you are only interested to run a subset of all tasks. For example, you might have prepared an index already and want only to repeatedly run search benchmarks. Or you want to run only the indexing task but nothing else.

You can use `--include-tasks` to specify a comma-separated list of tasks that you want to run. Each item in the list defines either the name of a task or the operation type of a task. Only the tasks that match will be executed. Currently there is also no command that lists the tasks of a challenge so you need to look at the track source.

Examples:

- Execute only the tasks with the name `index` and `term`: `--include-tasks="index,term"`
- Execute only tasks of type `search`: `--include-tasks="type:search"`
- You can also mix and match: `--include-tasks="index,type:search"`

exclude-tasks

Similarly to *include-tasks* when a challenge consists of one or more tasks you might be interested in excluding a single operations but include the rest.

You can use `--exclude-tasks` to specify a comma-separated list of tasks that you want to skip. Each item in the list defines either the name of a task or the operation type of a task. Only the tasks that did not match will be executed.

Examples:

- Skip any tasks with the name `index` and `term`: `--exclude-tasks="index,term"`
- Skip any tasks of type `search`: `--exclude-tasks="type:search"`
- You can also mix and match: `--exclude-tasks="index,type:search"`

team-repository

Selects the team repository that Rally should use to resolve cars. By default the `default` team repository is used, which is available in the Github project rally-teams. See the documentation about *cars* on how to add your own team repositories.

`team-revision`

Selects a specific revision in the team repository. By default, Rally will choose the most appropriate branch on its own (see the *car reference* for more details) but in some cases it is necessary to specify a certain commit. This is mostly needed when benchmarking specific historic commits of Elasticsearch which are incompatible with the current master branch of the team repository.

`team-path`

A directory that contains a team configuration. `--team-path` and `--team-repository` are mutually exclusive. See the *car reference* for the required directory structure.

Example:

```
esrally --team-path=~/Projects/es-teams
```

`target-os`

Specifies the name of the target operating system for which an artifact should be downloaded. By default this value is automatically derived based on the operating system Rally is run. This command line flag is only applicable to the `download` subcommand and allows to download an artifact for a different operating system. Example:

```
esrally download --distribution-version=7.5.1 --target-os=linux
```

`target-arch`

Specifies the name of the target CPU architecture for which an artifact should be downloaded. By default this value is automatically derived based on the CPU architecture Rally is run. This command line flag is only applicable to the `download` subcommand and allows to download an artifact for a different CPU architecture. Example:

```
esrally download --distribution-version=7.5.1 --target-arch=x86_64
```

`car`

A *car* defines the Elasticsearch configuration that will be used for the benchmark. To see a list of possible cars, issue `esrally list cars`. You can specify one or multiple comma-separated values.

Example

```
esrally --car="4gheap,ea"
```

Rally will configure Elasticsearch with 4GB of heap (`4gheap`) and enable Java assertions (`ea`).

`car-params`

Allows to override config variables of Elasticsearch. It accepts a list of comma-separated key-value pairs or a JSON file name. The key-value pairs have to be delimited by a colon.

Example

```
esrally --car="4gheap" --car-params="data_paths:'/opt/elasticsearch'"
```

The variables that are exposed depend on the car's configuration. In addition, Rally implements special handling for the variable `data_paths` (by default the value for this variable is determined by Rally).

`elasticsearch-plugins`

A comma-separated list of Elasticsearch plugins to install for the benchmark. If a plugin supports multiple configurations you need to specify the configuration after the plugin name. To see a list of possible plugins and configurations, issue `esrally list elasticsearch-plugins`.

Example:

```
esrally --elasticsearch-plugins="analysis-icu,xpack:security"
```

In this example, Rally will install the `analysis-icu` plugin and the `x-pack` plugin with the `security` configuration. See the reference documentation about *Elasticsearch plugins* for more details.

`plugin-params`

Allows to override variables of Elasticsearch plugins. It accepts a list of comma-separated key-value pairs or a JSON file name. The key-value pairs have to be delimited by a colon.

Example:

```
esrally --distribution-version=6.1.1. --elasticsearch-plugins="x-pack:monitoring-http
↪" --plugin-params="monitoring_type:'https',monitoring_host:'some_remote_host',
↪monitoring_port:10200,monitoring_user:'rally',monitoring_password:'m0n1t0r1ng'"
```

This enables the HTTP exporter of X-Pack Monitoring and exports the data to the configured monitoring host.

`pipeline`

Selects the *pipeline* that Rally should run.

Rally can autodetect the pipeline in most cases. If you specify `--distribution-version` it will auto-select the pipeline `from-distribution` otherwise it will use `from-sources`.

`enable-driver-profiling`

This option enables a profiler on all tasks that the load test driver performs. It is intended to help track authors spot accidental bottlenecks, especially if they implement their own runners or parameter sources. When this mode is enabled, Rally will enable a profiler in the load driver module. After each task and for each client, Rally will add the profile information to a dedicated profile log file. For example:

```
2017-02-09 08:23:24,35 rally.profile INFO
=== Profile START for client [0] and task [index-append-1000] ===
   16052402 function calls (15794402 primitive calls) in 180.221 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
      130    0.001    0.000  168.089    1.293 /Users/dm/Projects/rally/esrally/driver/
↪driver.py:908(time_period_based)
      129    0.260    0.002  168.088    1.303 /Users/dm/.rally/benchmarks/tracks/
↪develop/bottleneck/parameter_sources/bulk_source.py:79(params)
```

(continues on next page)

(continued from previous page)

```
   129000    0.750    0.000  167.791    0.001 /Users/dm/.rally/benchmarks/tracks/
↪develop/bottleneck/parameter_sources/randomevent.py:142(generate_event)
   516000    0.387    0.000  160.485    0.000 /Users/dm/.rally/benchmarks/tracks/
↪develop/bottleneck/parameter_sources/weightedarray.py:20(get_random)
   516000    6.199    0.000  160.098    0.000 /Users/dm/.rally/benchmarks/tracks/
↪develop/bottleneck/parameter_sources/weightedarray.py:23(__random_index)
   516000    1.292    0.000  152.289    0.000 /usr/local/Cellar/python3/3.6.0/
↪Frameworks/Python.framework/Versions/3.6/lib/python3.6/random.py:96(seed)
   516000  150.783    0.000  150.783    0.000 {function Random.seed at 0x10b7fa2f0}
   129000    0.363    0.000   45.686    0.000 /Users/dm/.rally/benchmarks/tracks/
↪develop/bottleneck/parameter_sources/randomevent.py:48(add_fields)
   129000    0.181    0.000   41.742    0.000 /Users/dm/.rally/benchmarks/tracks/
↪develop/bottleneck/parameter_sources/randomevent.py:79(add_fields)
   ....

=== Profile END for client [0] and task [index-append-1000] ===
```

In this example we can spot quickly that `Random.seed` is called excessively, causing an accidental bottleneck in the load test driver.

`test-mode`

Allows you to test a track without running it for the whole duration. This mode is only intended for quick sanity checks when creating a track. Don't rely on these numbers at all (they are meaningless).

If you write your own track you need to *prepare your track to support this mode*.

`telemetry`

Activates the provided *telemetry devices* for this race.

Example

```
esrally --telemetry=jfr,jit
```

This activates Java flight recorder and the JIT compiler telemetry devices.

`telemetry-params`

Allows to set parameters for telemetry devices. It accepts a list of comma-separated key-value pairs or a JSON file name. The key-value pairs have to be delimited by a colon. See the *telemetry devices* documentation for a list of supported parameters.

Example:

```
esrally --telemetry=jfr --telemetry-params="recording-template:'profile'"
```

This enables the Java flight recorder telemetry device and sets the `recording-template` parameter to "profile".

For more complex cases specify a JSON file. Store the following as `telemetry-params.json`:

```
{
  "node-stats-sample-interval": 10,
  "node-stats-include-indices-metrics": "completion,docs,fielddata"
}
```

and reference it when running Rally:

```
esrally --telemetry="node-stats" --telemetry-params="telemetry-params.json"
```

runtime-jdk

By default, Rally will derive the appropriate runtime JDK versions automatically per version of Elasticsearch. For example, it will choose JDK 8 or 7 for Elasticsearch 2.x but only JDK 8 for Elasticsearch 5.0.0. It will choose the highest available version.

This command line parameter sets the major version of the JDK that Rally should use to run Elasticsearch. It is required that either `JAVA_HOME` or `JAVAx_HOME` (where `x` is the major version, e.g. `JAVA8_HOME` for a JDK 8) points to the appropriate JDK.

Example:

```
# Run a benchmark with defaults (i.e. JDK 8)
esrally --distribution-version=2.4.0
# Force to run with JDK 7
esrally --distribution-version=2.4.0 --runtime-jdk=7
```

It is also possible to specify the JDK that is bundled with Elasticsearch with the special value `bundled`. The JDK is bundled from Elasticsearch 7.0.0 onwards.

revision

If you actively develop Elasticsearch and want to benchmark a source build of Elasticsearch (which Rally will create for you), you can specify the git revision of Elasticsearch that you want to benchmark. But note that Rally uses and expects the Gradle Wrapper in the Elasticsearch repository (`./gradlew`) which effectively means that it will only support this for Elasticsearch 5.0 or better. The default value is `current`.

You can specify the revision in different formats:

- `--revision=latest`: Use the HEAD revision from origin/master.
- `--revision=current`: Use the current revision (i.e. don't alter the local source tree).
- `--revision=abc123`: Where `abc123` is some git revision hash.
- `--revision=@2013-07-27T10:37:00Z`: Determines the revision that is closest to the provided date. Rally logs to which git revision hash the date has been resolved and if you use Elasticsearch as metrics store (instead of the default in-memory one), *each metric record will contain the git revision hash also in the meta-data section*.

Supported date format: If you specify a date, it has to be ISO-8601 conformant and must start with an `@` sign to make it easier for Rally to determine that you actually mean a date.

If you want to create source builds of Elasticsearch plugins, you need to specify the revision for Elasticsearch and all relevant plugins separately. Revisions for Elasticsearch and each plugin need to be comma-separated (`,`). Each revision is prefixed either by `elasticsearch` or by the plugin name and separated by a colon (`:`). As core plugins are contained in the Elasticsearch repo, there is no need to specify a revision for them (the revision would even be ignored in fact).

Examples:

- Build latest Elasticsearch and plugin "my-plugin": `--revision="elasticsearch:latest, my-plugin:latest"`

- Build Elasticsearch tag `v5.6.1` and revision `abc123` of plugin "my-plugin": `--revision="elasticsearch:v5.6.1,my-plugin:abc123"`

Note that it is still required to provide the parameter `--elasticsearch-plugins`. Specifying a plugin with `--revision` just tells Rally which revision to use for building the artifact. See the documentation on *Elasticsearch plugins* for more details.

`distribution-version`

If you want Rally to launch and benchmark a cluster using a binary distribution, you can specify the version here.

Note: Do not use `distribution-version` when benchmarking a cluster that hasn't been setup by Rally (i.e. together with `pipeline=benchmark-only`); Rally automatically derives and stores the version of the cluster in the metrics store regardless of the `pipeline` used.

Example

```
esrally --distribution-version=2.3.3
```

Rally will then benchmark the official Elasticsearch 2.3.3 distribution. Please check our *version support page* to see which Elasticsearch versions are currently supported by Rally.

`distribution-repository`

Rally does not only support benchmarking official distributions but can also benchmark snapshot builds. This is option is really just intended for our benchmarks that are run in continuous integration but if you want to, you can use it too. The only supported value out of the box is `release` (default) but you can define arbitrary repositories in `~/.rally/rally.ini`.

Example

Say, you have an in-house repository where Elasticsearch snapshot builds get published. Then you can add the following in the `distributions` section of your Rally config file:

```
in_house_snapshot.url = https://www.example.org/snapshots/elasticsearch/elasticsearch-
↪{{VERSION}}.tar.gz
in_house_snapshot.cache = false
```

The `url` property defines the URL pattern for this repository. The `cache` property defines whether Rally should always download a new archive (`cache=false`) or just reuse a previously downloaded version (`cache=true`). Rally will replace the `{{VERSION}}` placeholder of in the `url` property with the value of `distribution-version` provided by the user on the command line.

You can use this distribution repository with the name "in_house_snapshot" as follows:

```
esrally --distribution-repository=in_house_snapshot --distribution-version=7.0.0-
↪SNAPSHOT
```

This will benchmark the latest 7.0.0 snapshot build of Elasticsearch.

`report-format`

The command line reporter in Rally displays a table with key metrics after a race. With this option you can specify whether this table should be in `markdown` format (default) or `csv`.

`show-in-report`

By default, the command line reporter will only show values that are available (`available`). With `all` you can force it to show a line for every value, even undefined ones, and with `all-percentiles` it will show only available values but force output of all possible percentile values.

This command line parameter is not available for comparing races.

`report-file`

By default, the command line reporter will print the results only on standard output, but can also write it to a file.

Example

```
esrally --report-format=csv --report-file=~/benchmarks/result.csv
```

`client-options`

With this option you can customize Rally's internal Elasticsearch client.

It accepts a list of comma-separated key-value pairs. The key-value pairs have to be delimited by a colon. These options are passed directly to the Elasticsearch Python client API. See their documentation on a list of supported options.

We support the following data types:

- Strings: Have to be enclosed in single quotes. Example: `ca_certs:'/path/to/CA_certs'`
- Numbers: There is nothing special about numbers. Example: `sniffer_timeout:60`
- Booleans: Specify either `true` or `false`. Example: `use_ssl:true`

Default value: `timeout:60`

Warning: If you provide your own client options, the default value will not be magically merged. You have to specify all client options explicitly. The only exceptions to this rule is `ca_cert` (see below).

Rally recognizes the following client options in addition:

- `max_connections`: By default, Rally will choose the maximum allowed number of connections automatically (equal to the number of simulated clients but at least 256 connections). With this property it is possible to override that logic but a minimum of 256 is enforced internally.
- `enable_cleanup_closed` (default: `false`): In some cases, SSL connections might not be properly closed and the number of open connections increases as a result. When this client option is set to `true`, the Elasticsearch client will check and forcefully close these connections.

Examples

Here are a few common examples:

- Enable HTTP compression: `--client-options="http_compress:true"`
- 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.

TLS/SSL

This is applicable e.g. if you have X-Pack Security installed. Enable it with `use_ssl:true`.

TLS/SSL Certificate Verification

Server certificate verification is controlled with the `verify_certs` boolean. The default value is *true*. To disable use `verify_certs:false`. If `verify_certs:true`, Rally will attempt to verify the certificate provided by Elasticsearch. If they are private certificates, you will also need to supply the private CA certificate using `ca_certs:'/path/to/cacert.pem'`.

You can also optionally present client certificates, e.g. if Elasticsearch has been configured with `xpack.security.http.ssl.client_authentication: required` (see also Elasticsearch HTTP TLS/SSL settings). Client certificates can be presented regardless of the `verify_certs` setting, but it's strongly recommended to always verify the server certificates.

TLS/SSL Examples

- Enable SSL, verify server certificates using public CA: `--client-options="use_ssl:true,verify_certs:true"`. Note that you don't need to set `ca_cert` (which defines the path to the root certificates). Rally does this automatically for you.
- Enable SSL, verify server certificates using private CA: `--client-options="use_ssl:true,verify_certs:true,ca_certs:'/path/to/cacert.pem'"`
- Enable SSL, verify server certificates using private CA, present client certificates: `--client-options="use_ssl:true,verify_certs:true,ca_certs:'/path/to/cacert.pem',client_cert:'/path/to/client_cert.pem',client_key:'/path/to/client_key.pem'"`

`on-error`

This option controls how Rally behaves when a response error occurs. The following values are possible:

- `continue`: only records that an error has happened and will continue with the benchmark. At the end of a race, errors show up in the "error rate" metric.
- `continue-on-non-fatal` (default): Behaves as `continue` but aborts the benchmark immediately on all fatal errors. The only error that is considered fatal is "Connection Refused" (ECONNREFUSED).
- `abort`: aborts the benchmark on the first request error with a detailed error message.

`load-driver-hosts`

By default, Rally will run its load driver on the same machine where you start the benchmark. However, if you benchmark larger clusters, one machine may not be enough to generate sufficient load. Hence, you can specify a comma-separated list of hosts which should be used to generate load with `--load-driver-hosts`.

Example

```
esrally --load-driver-hosts=10.17.20.5,10.17.20.6
```

In the example, above Rally will generate load from the hosts `10.17.20.5` and `10.17.20.6`. For this to work, you need to start a Rally daemon on these machines, see *distributing the load test driver* for a complete example.

`target-hosts`

If you run the `benchmark-only` *pipeline* or you want Rally to *benchmark a remote cluster*, then you can specify a comma-delimited list of hosts:port pairs to which Rally should connect. The default value is `127.0.0.1:9200`.

Example

```
esrally --pipeline=benchmark-only --target-hosts=10.17.0.5:9200,10.17.0.
↪6:9200
```

This will run the benchmark against the hosts 10.17.0.5 and 10.17.0.6 on port 9200. See `client-options` if you use X-Pack Security and need to authenticate or Rally should use https.

You can also target multiple clusters with `--target-hosts` for specific use cases. This is described in the *Advanced topics section*.

`limit`

Allows to control the number of races returned by `esrally list races` The default value is 10.

Example

The following invocation will list the 50 most recent races:

```
esrally list races --limit=50
```

`quiet`

Suppresses some output on the command line.

`kill-running-processes`

Rally attempts to generate benchmark results that are not skewed unintentionally. Consequently, if some benchmark is running, Rally will not allow you to start another one. Instead, you should stop the current benchmark and start another one manually. This flag can be added to handle automatically this process for you.

Only one Rally benchmark is allowed to run at the same time. If any processes is running, it is going to kill them and allow Rally to continue to run a new benchmark.

The default value is `false`.

`offline`

Tells Rally that it should assume it has no connection to the Internet when checking for track data. The default value is `false`. Note that Rally will only assume this for tracks but not for anything else, e.g. it will still try to download Elasticsearch distributions that are not locally cached or fetch the Elasticsearch source tree.

`preserve-install`

Rally usually installs and launches an Elasticsearch cluster internally and wipes the entire directory after the benchmark is done. Sometimes you want to keep this cluster including all data after the benchmark has finished and that's what you can do with this flag. Note that depending on the track that has been run, the cluster can eat up a very significant amount of disk space (at least dozens of GB). The default value is configurable in the advanced configuration but usually `false`.

Note: This option does only affect clusters that are provisioned by Rally. More specifically, if you use the pipeline `benchmark-only`, this option is ineffective as Rally does not provision a cluster in this case.

`advanced-config`

This flag determines whether Rally should present additional (advanced) configuration options. The default value is `false`.

Example

```
esrally configure --advanced-config
```

`assume-defaults`

This flag determines whether Rally should automatically accept all values for configuration options that provide a default. This is mainly intended to configure Rally automatically in CI runs. The default value is `false`.

Example

```
esrally configure --assume-defaults=true
```

`user-tag`

This is only relevant when you want to run *tournaments*. You can use this flag to attach an arbitrary text to the meta-data of each metric record and also the corresponding race. This will help you to recognize a race when you run `esrally list races` as you don't need to remember the concrete timestamp on which a race has been run but can instead use your own descriptive names.

The required format is `key` ":" `value`. You can choose `key` and `value` freely.

Example

```
esrally --user-tag="intention:github-issue-1234-baseline,gc:cms"
```

You can also specify multiple tags. They need to be separated by a comma.

Example

```
esrally --user-tag="disk:SSD,data_node_count:4"
```

When you run `esrally list races`, this will show up again:

```
Race Timestamp    Track    Track Parameters   Challenge            Car       User Tag
----------------  -------  ------------------ -------------------  --------  ---------
↪---------------------------
20160518T122341Z  pmc                         append-no-conflicts  defaults ␣
↪intention:github-issue-1234-baseline
20160518T112341Z  pmc                         append-no-conflicts  defaults  disk:SSD,
↪data_node_count:4
```

This will help you recognize a specific race when running `esrally compare`.

indices

A comma-separated list of index patterns to target when generating a track with the `create-track` subcommand.

Examples

Target a single index:

```
esrally create-track --track=acme --indices="products" --target-hosts=127.0.0.1:9200 -
↪-output-path=~/tracks
```

Target multiple indices:

```
esrally create-track --track=acme --indices="products,companies" --target-hosts=127.0.
↪0.1:9200 --output-path=~/tracks
```

Use index patterns:

```
esrally create-track --track=my-logs --indices="logs-*" --target-hosts=127.0.0.1:9200␣
↪--output-path=~/tracks
```

Note: If the cluster requires authentication specify credentials via `--client-options` as described in the *command line reference*.

2.12.3 Advanced topics

target-hosts

Rally can also create client connections for multiple Elasticsearch clusters. This is only useful if you want to create *custom runners* that execute operations against multiple clusters, for example for cross cluster search or cross cluster replication.

To define the host:port pairs for additional clusters with `target-hosts` you can specify either a JSON filename (ending in `.json`) or an inline JSON string. The JSON object should be a collection of name:value pairs. `name` is string for the cluster name and there **must be** one `default`.

Examples:

- json file: `--target-hosts="target_hosts1.json"`:

  ```
  { "default": ["127.0.0.1:9200","10.127.0.3:19200"] }
  ```

- json file defining two clusters: `--target-hosts="target_hosts2.json"`:

  ```
  {
    "default": [
      {"host": "127.0.0.1", "port": 9200},
      {"host": "127.0.0.1", "port": 19200}
    ],
    "remote":[
      {"host": "10.127.0.3", "port": 9200},
      {"host": "10.127.0.8", "port": 9201}
    ]
  }
  ```

- json inline string defining two clusters:

```
--target-hosts="{\"default\":[\"127.0.0.1:9200\"],\"remote\":[\"127.0.0.1:19200\",
↪\"127.0.0.1:19201\"]}"
```

Note: **All** *built-in operations* will use the connection to the `default` cluster. However, you can utilize the client connections to the additional clusters in your *custom runners*.

`client-options`

`client-options` can optionally specify options for the Elasticsearch clients when multiple clusters have been defined with `target-hosts`. If omitted, the default is `timeout:60` for all cluster connections.

The format is similar to `target-hosts`, supporting both filenames ending in `.json` or inline JSON, however, the parameters are a collection of name:value pairs, as opposed to arrays.

Examples, assuming that two clusters have been specified with `--target-hosts`:

- json file: `--client-options="client_options1.json"`:

```
{
  "default": {
    "timeout": 60
},
  "remote": {
    "use_ssl": true,
    "verify_certs": false,
    "ca_certs": "/path/to/cacert.pem"
  }
}
```

- json inline string defining two clusters:

```
--client-options="{\"default\":{\"timeout\": 60}, \"remote\": {\"use_ssl\":true,\
↪"verify_certs\":false,\"ca_certs\":\"/path/to/cacert.pem\"}}"
```

Warning: If you use `client-options` you must specify options for **every** cluster name defined with `target-hosts`. Rally will raise an error if there is a mismatch.

2.13 Offline Usage

In some corporate environments servers do not have Internet access. You can still use Rally in such environments and this page summarizes all information that you need to get started.

2.13.1 Installation and Configuration

We provide a special offline installation package. Follow the *offline installation guide* and *configure Rally as usual* afterwards.

2.13.2 Command Line Usage

Rally will automatically detect upon startup that no Internet connection is available and print the following warning:

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

It detects this by trying to connect to `https://github.com`. If you want it to probe against a different HTTP endpoint (e.g. a company-internal git server) you need to add a configuration property named `probing.url` in the `system` section of Rally's configuration file at `~/.rally/rally.ini`. Specify `--offline` if you want to disable probing entirely.

Example of `system` section with custom probing url in `~/.rally/rally.ini`:

```
[system]
env.name = local
probing.url = https://www.company-internal-server.com/
```

2.13.3 Using tracks

A Rally track describes a benchmarking scenario. You can either write your own tracks or use the tracks that Rally provides out of the box. In the former case, Rally will work just fine in an offline environment. In the latter case, Rally would normally download the track and its associated data from the Internet. If you want to use one of Rally's standard tracks in offline mode, you need to download all relevant files first on a machine that has Internet access and copy it to the target machine(s).

Use the download script to download all data for a track on a machine that has access to the Internet. Example:

```
# downloads the script from Github
curl -O https://raw.githubusercontent.com/elastic/rally-tracks/master/download.sh
chmod u+x download.sh
# download all data for the geonames track
./download.sh geonames
```

This will download all data for the geonames track and create a tar file `rally-track-data-geonames.tar` in the current directory. Copy this file to the home directory of the user which will execute Rally on the target machine (e.g. `/home/rally-user`).

On the target machine, run:

```
cd ~
tar -xf rally-track-data-geonames.tar
```

The download script does not require a Rally installation on the machine with Internet access but assumes that `git` and `curl` are available.

After you've copied the data, you can list the available tracks with `esrally list tracks`. If a track shows up in this list, it just means that the track description is available locally but not necessarily all data files.

2.13.4 Using cars

Note: You can skip this section if you use Rally only as a load generator.

If you have Rally configure and start Elasticsearch then you also need the out-of-the-box configurations available. Run the following command on a machine with Internet access:

```
git clone https://github.com/elastic/rally-teams.git ~/.rally/benchmarks/teams/default
tar -C ~ -czf rally-teams.tar.gz .rally/benchmarks/teams/default
```

Copy that file to the target machine(s) and run on the target machine:

```
cd ~
tar -xzf rally-teams.tar.gz
```

After you've copied the data, you can list the available tracks with `esrally list cars`.

2.14 Track Reference

2.14.1 Definition

A track is a specification of one or more benchmarking scenarios with a specific document corpus. It defines for example the involved indices, data files and the operations that are invoked. Its most important attributes are:

- One or more indices, each with one or more types
- The queries to issue
- Source URL of the benchmark data
- A list of steps to run, which we'll call "challenge", for example indexing data with a specific number of documents per bulk request or running searches for a defined number of iterations.

2.14.2 Track File Format and Storage

A track is specified in a JSON file.

Ad-hoc use

For ad-hoc use you can store a track definition anywhere on the file system and reference it with `--track-path`, e.g:

```
# provide a directory - Rally searches for a track.json file in this directory
# Track name is "app-logs"
esrally --track-path=~/Projects/tracks/app-logs
# provide a file name - Rally uses this file directly
# Track name is "syslog"
esrally --track-path=~/Projects/tracks/syslog.json
```

Rally will also search for additional files like mappings or data files in the provided directory. If you use advanced features like *custom runners* or *parameter sources* we recommend that you create a separate directory per track.

Custom Track Repositories

Alternatively, you can store Rally tracks also in a dedicated git repository which we call a "track repository". Rally provides a default track repository that is hosted on Github. You can also add your own track repositories although this requires a bit of additional work. First of all, track repositories need to be managed by git. The reason is that Rally can benchmark multiple versions of Elasticsearch and we use git branches in the track repository to determine the best match for each track (based on the command line parameter `--distribution-version`). The versioning scheme is as follows:

- The *master* branch needs to work with the latest *master* branch of Elasticsearch.
- All other branches need to match the version scheme of Elasticsearch, i.e. `MAJOR.MINOR.PATCH-SUFFIX` where all parts except `MAJOR` are optional.

Rally implements a fallback logic so you don't need to define a branch for each patch release of Elasticsearch. For example:

- The branch *6.0.0-alpha1* will be chosen for the version `6.0.0-alpha1` of Elasticsearch.
- The branch *5* will be chosen for all versions for Elasticsearch with the major version 5, e.g. `5.0.0`, `5.1.3` (provided there is no specific branch).

Rally tries to use the branch with the best match to the benchmarked version of Elasticsearch.

Rally will also search for related files like mappings or custom runners or parameter sources in the track repository. However, Rally will use a separate directory to look for data files (`~/.rally/benchmarks/data/$TRACK_NAME/`). The reason is simply that we do not want to check multi-GB data files into git.

Creating a new track repository

All track repositories are located in `~/.rally/benchmarks/tracks`. If you want to add a dedicated track repository, called `private` follow these steps:

```
cd ~/.rally/benchmarks/tracks
mkdir private
cd private
git init
# add your track now
git add .
git commit -m "Initial commit"
```

If you want to share your tracks with others you need to add a remote and push it:

```
git remote add origin git@git-repos.acme.com:acme/rally-tracks.git
git push -u origin master
```

If you have added a remote you should also add it in `~/.rally/rally.ini`, otherwise you can skip this step. Open the file in your editor of choice and add the following line in the section `tracks`:

```
private.url = <<URL_TO_YOUR_ORIGIN>>
```

If you specify `--track-repository=private`, Rally will check whether there is a directory `~/.rally/benchmarks/tracks/private`. If there is none, it will use the provided URL to clone the repo. However, if the directory already exists, the property gets ignored and Rally will just update the local tracking branches before the benchmark starts.

You can now verify that everything works by listing all tracks in this track repository:

```
esrally list tracks --track-repository=private
```

This shows all tracks that are available on the `master` branch of this repository. Suppose you only created tracks on the branch `2` because you're interested in the performance of Elasticsearch 2.x, then you can specify also the distribution version:

```
esrally list tracks --track-repository=private --distribution-version=2.0.0
```

Rally will follow the same branch fallback logic as described above.

Adding an already existing track repository

If you want to add a track repository that already exists, just open `~/.rally/rally.ini` in your editor of choice and add the following line in the section `tracks`:

```
your_repo_name.url = <<URL_TO_YOUR_ORIGIN>>
```

After you have added this line, have Rally list the tracks in this repository:

```
esrally list tracks --track-repository=your_repo_name
```

When to use what?

We recommend the following path:

- Start with a simple json file. The file name can be arbitrary.
- If you need *custom runners* or *parameter sources*, create one directory per track. Then you can keep everything that is related to one track in one place. Remember that the track JSON file needs to be named `track.json`.
- If you want to version your tracks so they can work with multiple versions of Elasticsearch (e.g. you are running benchmarks before an upgrade), use a track repository.

2.14.3 Anatomy of a track

A track JSON file consists of the following sections:

- indices
- templates
- corpora
- operations
- schedule
- challenges

In the `indices` and `templates` sections you define the relevant indices and index templates. These sections are optional but recommended if you want to create indices and index templates with the help of Rally.

In the `corpora` section you define all document corpora (i.e. data files) that Rally should use for this track.

In the `operations` section you describe which operations are available for this track and how they are parametrized. This section is optional and you can also define any operations directly per challenge. You can use it, if you want to share operation definitions between challenges.

In the `schedule` section you describe the workload for the benchmark, for example index with two clients at maximum throughput while searching with another two clients with ten operations per second. The schedule either uses the operations defined in the `operations` block or defines the operations to execute inline.

In the `challenges` section you describe more than one set of operations, in the event your track needs to test more than one set of scenarios. This section is optional, and more information can be found in the *challenges section*.

Creating a track does not require all of the above sections to be used. Tracks that are used against existing data may only rely on querying `operations` and can omit the `indices`, `templates`, and `corpora` sections. An example of this can be found in the *task with a single track example*.

2.14.4 Track elements

The track elements that are described here are defined in Rally's JSON schema for tracks. Rally uses this track schema to validate your tracks when it is loading them.

Each track defines the following info attributes:

- `version` (optional): An integer describing the track specification version in use. Rally uses it to detect incompatible future track specification versions and raise an error. See the table below for a reference of valid versions.
- `description` (optional): A human-readable description of the track. Although it is optional, we recommend providing it.

Track Specification Version	Rally version
1	>=0.7.3, <0.10.0
2	>=0.9.0

The `version` property has been introduced with Rally 0.7.3. Rally versions before 0.7.3 do not recognize this property and thus cannot detect incompatible track specification versions.

Example:

```
{
    "version": 2,
    "description": "POIs from Geonames"
}
```

meta

For each track, an optional structure, called `meta` can be defined. You are free which properties this element should contain.

This element can also be defined on the following elements:

- `challenge`
- `operation`
- `task`

If the `meta` structure contains the same key on different elements, more specific ones will override the same key of more generic elements. The order from generic to most specific is:

1. track
2. challenge
3. operation
4. task

E.g. a key defined on a task, will override the same key defined on a challenge. All properties defined within the merged `meta` structure, will get copied into each metrics record.

indices

The `indices` section contains a list of all indices that are used by this track.

Each index in this list consists of the following properties:

- `name` (mandatory): The name of the index.
- `body` (optional): File name of the corresponding index definition that will be used as body in the create index API call.
- `types` (optional): A list of type names in this index. Types have been removed in Elasticsearch 7.0.0 so you must not specify this property if you want to benchmark Elasticsearch 7.0.0 or later.

Example:

```
"indices": [
    {
      "name": "geonames",
      "body": "geonames-index.json",
      "types": ["docs"]
    }
]
```

templates

The `templates` section contains a list of all index templates that Rally should create.

- `name` (mandatory): Index template name
- `index-pattern` (mandatory): Index pattern that matches the index template. This must match the definition in the index template file.
- `delete-matching-indices` (optional, defaults to `true`): Delete all indices that match the provided index pattern before start of the benchmark.
- `template` (mandatory): Index template file name

Example:

```
"templates": [
    {
        "name": "my-default-index-template",
        "index-pattern": "my-index-*",
        "delete-matching-indices": true,
        "template": "default-template.json"
    }
]
```

corpora

The `corpora` section contains all document corpora that are used by this track. Note that you can reuse document corpora across tracks; just copy & paste the respective corpora definitions. It consists of the following properties:

- `name` (mandatory): Name of this document corpus. As this name is also used by Rally in directory names, it is recommended to only use lower-case names without whitespaces for maximum compatibility across file systems.
- `documents` (mandatory): A list of documents files.

Each entry in the `documents` list consists of the following properties:

- `base-url` (optional): A http(s), S3 or Google Storage URL that points to the root path where Rally can obtain the corresponding source file. Rally can also download data from private S3 or Google Storage buckets if access is properly configured:
  - S3 according to docs.
  - Google Storage: Either using client library authentication or by presenting an oauth2 token via the `GOOGLE_AUTH_TOKEN` environment variable, typically done using: `export GOOGLE_AUTH_TOKEN=$(gcloud auth print-access-token)`.
- `source-format` (optional, default: `bulk`): Defines in which format Rally should interpret the data file specified by `source-file`. Currently, only `bulk` is supported.
- `source-file` (mandatory): File name of the corresponding documents. For local use, this file can be a `.json` file. If you provide a `base-url` we recommend that you provide a compressed file here. The following extensions are supported: `.zip`, `.bz2`, `.gz`, `.tar`, `.tar.gz`, `.tgz` or `.tar.bz2`. It must contain exactly one JSON file with the same name. The preferred file extension for our official tracks is `.bz2`.
- `includes-action-and-meta-data` (optional, defaults to `false`): Defines whether the documents file contains already an action and meta-data line (`true`) or only documents (`false`).
- `document-count` (mandatory): Number of documents in the source file. This number is used by Rally to determine which client indexes which part of the document corpus (each of the N clients gets one N-th of the document corpus). If you are using parent-child, specify the number of parent documents.
- `compressed-bytes` (optional but recommended): The size in bytes of the compressed source file. This number is used to show users how much data will be downloaded by Rally and also to check whether the download is complete.
- `uncompressed-bytes` (optional but recommended): The size in bytes of the source file after decompression. This number is used by Rally to show users how much disk space the decompressed file will need and to check that the whole file could be decompressed successfully.
- `target-index`: Defines the name of the index which should be targeted for bulk operations. Rally will automatically derive this value if you have defined exactly one index in the `indices` section. Ignored if `includes-action-and-meta-data` is `true`.
- `target-type` (optional): Defines the name of the document type which should be targeted for bulk operations. Rally will automatically derive this value if you have defined exactly one index in the `indices` section and this index has exactly one type. Ignored if `includes-action-and-meta-data` is `true`. Types have been removed in Elasticsearch 7.0.0 so you must not specify this property if you want to benchmark Elasticsearch 7.0.0 or later.

To avoid repetition, you can specify default values on document corpus level for the following properties:

- `base-url`
- `source-format`
- `includes-action-and-meta-data`
- `target-index`
- `target-type`

Examples

Here we define a single document corpus with one set of documents:

```
"corpora": [
  {
    "name": "geonames",
    "documents": [
      {
        "base-url": "http://benchmarks.elasticsearch.org.s3.amazonaws.com/corpora/
↪geonames",
        "source-file": "documents.json.bz2",
        "document-count": 11396505,
        "compressed-bytes": 264698741,
        "uncompressed-bytes": 3547614383,
        "target-index": "geonames",
        "target-type": "docs"
      }
    ]
  }
]
```

We can also define default values on document corpus level but override some of them (`base-url` for the last entry):

```
"corpora": [
  {
    "name": "http_logs",
    "base-url": "http://benchmarks.elasticsearch.org.s3.amazonaws.com/corpora/http_
↪logs",
    "target-type": "docs",
    "documents": [
      {
        "source-file": "documents-181998.json.bz2",
        "document-count": 2708746,
        "target-index": "logs-181998"
      },
      {
        "source-file": "documents-191998.json.bz2",
        "document-count": 9697882,
        "target-index": "logs-191998"
      },
      {
        "base-url": "http://example.org/corpora/http_logs",
        "source-file": "documents-201998.json.bz2",
        "document-count": 13053463,
        "target-index": "logs-201998"
      }
    ]
  }
]
```

challenge

If your track defines only one benchmarking scenario specify the `schedule` on top-level. Use the `challenge` element if you want to specify additional properties like a name or a description. You can think of a challenge as a benchmarking scenario. If you have multiple challenges, you can define an array of `challenges`.

This section contains one or more challenges which describe the benchmark scenarios for this data set. A challenge can reference all operations that are defined in the `operations` section.

Each challenge consists of the following properties:

- `name` (mandatory): A descriptive name of the challenge. Should not contain spaces in order to simplify handling on the command line for users.
- `description` (optional): A human readable description of the challenge.
- `default` (optional): If true, Rally selects this challenge by default if the user did not specify a challenge on the command line. If your track only defines one challenge, it is implicitly selected as default, otherwise you need to define `"default": true` on exactly one challenge.
- `schedule` (mandatory): Defines the workload. It is described in more detail below.

Note: You should strive to minimize the number of challenges. If you just want to run a subset of the tasks in a challenge, use *task filtering*.

schedule

The `schedule` element contains a list of tasks that are executed by Rally, i.e. it describes the workload. Each task consists of the following properties:

- `name` (optional): This property defines an explicit name for the given task. By default the operation's name is implicitly used as the task name but if the same operation is run multiple times, a unique task name must be specified using this property.
- `operation` (mandatory): This property refers either to the name of an operation that has been defined in the `operations` section or directly defines an operation inline.
- `clients` (optional, defaults to 1): The number of clients that should execute a task concurrently.
- `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).
- `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. Rally will just bulk index all documents and consider every sample after the warmup time period as measurement sample.
- `schedule` (optional, defaults to `deterministic`): Defines the schedule for this task, i.e. it defines at which point in time during the benchmark an operation should be executed. For example, if you specify a `deterministic` schedule and a target-interval of 5 (seconds), Rally will attempt to execute the corresponding operation at second 0, 5, 10, 15 … . Out of the box, Rally supports `deterministic` and `poisson` but you can define your own *custom schedules*.
- `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.

- `target-interval` (optional): This is just `1 / target-throughput` (in seconds) and may be more convenient for cases where the throughput is less than one operation per second. Define either `target-throughput` or `target-interval` but not both (otherwise Rally will raise an error).

Defining operations

In the following snippet we define two operations `force-merge` and a `match-all` query separately in an operations block:

```
{
  "operations": [
    {
      "name": "force-merge",
      "operation-type": "force-merge"
    },
    {
      "name": "match-all-query",
      "operation-type": "search",
      "body": {
        "query": {
          "match_all": {}
        }
      }
    }
  ],
  "schedule": [
    {
      "operation": "force-merge",
      "clients": 1
    },
    {
      "operation": "match-all-query",
      "clients": 4,
      "warmup-iterations": 1000,
      "iterations": 1000,
      "target-throughput": 100
    }
  ]
}
```

If we do not want to reuse these operations, we can also define them inline. Note that the `operations` section is gone:

```
{
  "schedule": [
    {
      "operation": {
        "name": "force-merge",
        "operation-type": "force-merge"
      },
      "clients": 1
    },
    {
      "operation": {
        "name": "match-all-query",
        "operation-type": "search",
```

(continues on next page)

(continued from previous page)

```
        "body": {
          "query": {
            "match_all": {}
          }
        }
      },
      "clients": 4,
      "warmup-iterations": 1000,
      "iterations": 1000,
      "target-throughput": 100
    }
  ]
}
```

Contrary to the `query`, the `force-merge` operation does not take any parameters, so Rally allows us to just specify the `operation-type` for this operation. It's name will be the same as the operation's type:

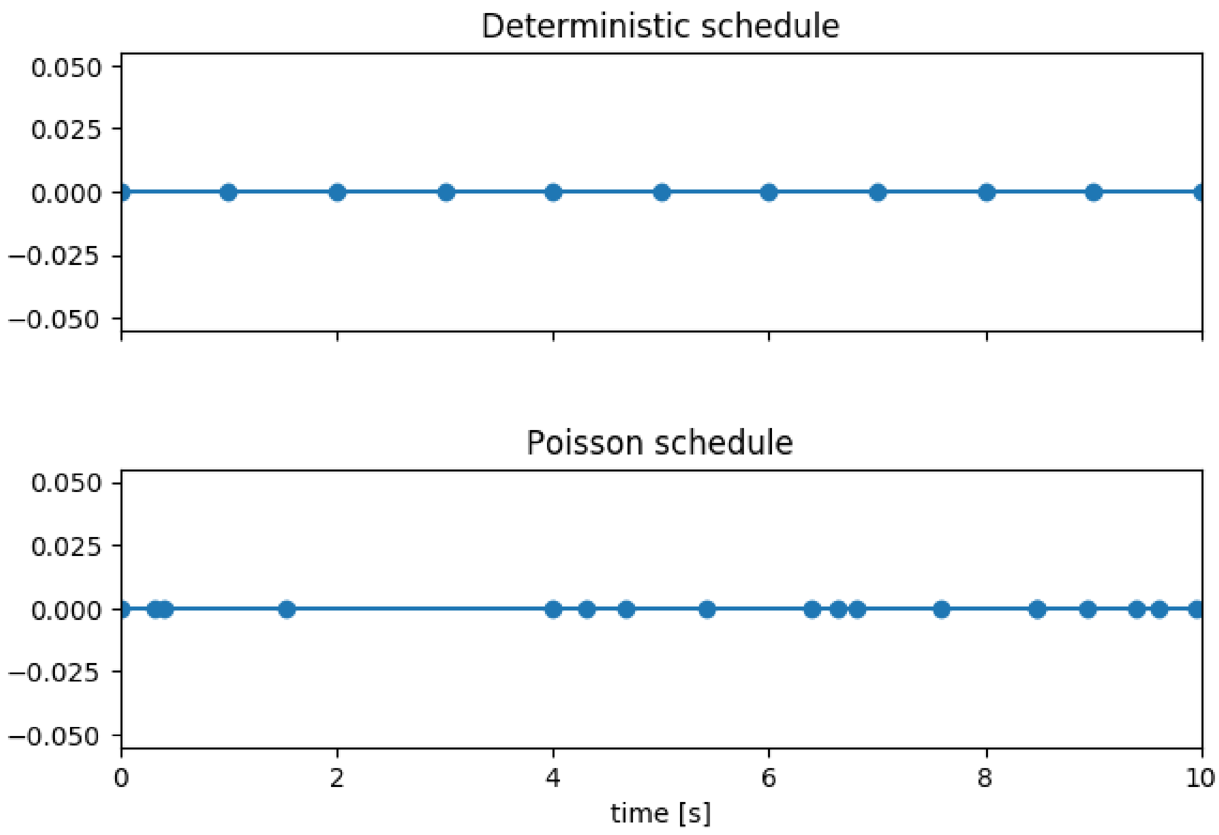
```
{
  "schedule": [
    {
      "operation": "force-merge",
      "clients": 1
    },
    {
      "operation": {
        "name": "match-all-query",
        "operation-type": "search",
        "body": {
          "query": {
            "match_all": {}
          }
        }
      },
      "clients": 4,
      "warmup-iterations": 1000,
      "iterations": 1000,
      "target-throughput": 100
    }
  ]
}
```

Choosing a schedule

Rally allows you to choose between the following schedules to simulate traffic:

- deterministically distributed
- Poisson distributed

The diagram below shows how different schedules in Rally behave during the first ten seconds of a benchmark. Each schedule is configured for a (mean) target throughput of one operation per second.



If you want as much reproducibility as possible you can choose the *deterministic* schedule. A Poisson distribution models random independent arrivals of clients which on average match the expected arrival rate which makes it suitable for modelling the behaviour of multiple clients that decide independently when to issue a request. For this reason, Poisson processes play an important role in queueing theory.

If you have more complex needs on how to model traffic, you can also implement a *custom schedule*.

Time-based vs. iteration-based

You should usually use time periods for batch style operations and iterations for the rest. However, you can also choose to run a query for a certain time period.

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. The `parallel` element defines of the following properties:

- `clients` (optional): The number of clients that should execute the provided tasks. If you specify this property, Rally will only use as many clients as you have defined on the `parallel` element (see examples)!
- `warmup-time-period` (optional, defaults to 0): Allows to define a default value for all tasks of the `parallel` element.
- `time-period` (optional, no default value if not specified): Allows to define a default value for all tasks of the `parallel` element.
- `warmup-iterations` (optional, defaults to 0): Allows to define a default value for all tasks of the `parallel` element.
- `iterations` (optional, defaults to 1): Allows to define a default value for all tasks of the `parallel` element.

- `completed-by` (optional): Allows to define the name of one task in the `tasks` list. As soon as this task has completed, the whole `parallel` task structure is considered completed. If this property is not explicitly defined, the `parallel` task structure is considered completed as soon as all its subtasks have completed. A task is completed if and only if all associated clients have completed execution.
- `tasks` (mandatory): Defines a list of tasks that should be executed concurrently. Each task in the list can define the following properties that have been defined above: `clients`, `warmup-time-period`, `time-period`, `warmup-iterations` and `iterations`.

Note: `parallel` elements cannot be nested.

Warning: Specify the number of clients on each task separately. If you specify this number on the `parallel` element instead, Rally will only use that many clients in total and you will only want to use this behavior in very rare cases (see examples)!

operations

The `operations` section contains a list of all operations that are available when specifying a schedule. Operations define the static properties of a request against Elasticsearch whereas the `schedule` element defines the dynamic properties (such as the target throughput).

Each operation consists of the following properties:

- `name` (mandatory): The name of this operation. You can choose this name freely. It is only needed to reference the operation when defining schedules.
- `operation-type` (mandatory): Type of this operation. See below for the operation types that are supported out of the box in Rally. You can also add arbitrary operations by defining *custom runners*.
- `include-in-reporting` (optional, defaults to `true` for normal operations and to `false` for administrative operations): Whether or not this operation should be included in the command line report. For example you might want Rally to create an index for you but you are not interested in detailed metrics about it. Note that Rally will still record all metrics in the metrics store.

Some of the operations below are also retryable (marked accordingly below). Retryable operations expose the following properties:

- `retries` (optional, defaults to 0): The number of times the operation is retried.
- `retry-until-success` (optional, defaults to `false`): Retries until the operation returns a success. This will also forcibly set `retry-on-error` to `true`.
- `retry-wait-period` (optional, defaults to 0.5): The time in seconds to wait between retry attempts.
- `retry-on-timeout` (optional, defaults to `true`): Whether to retry on connection timeout.
- `retry-on-error` (optional, defaults to `false`): Whether to retry on errors (e.g. when an index could not be deleted).

Depending on the operation type a couple of further parameters can be specified.

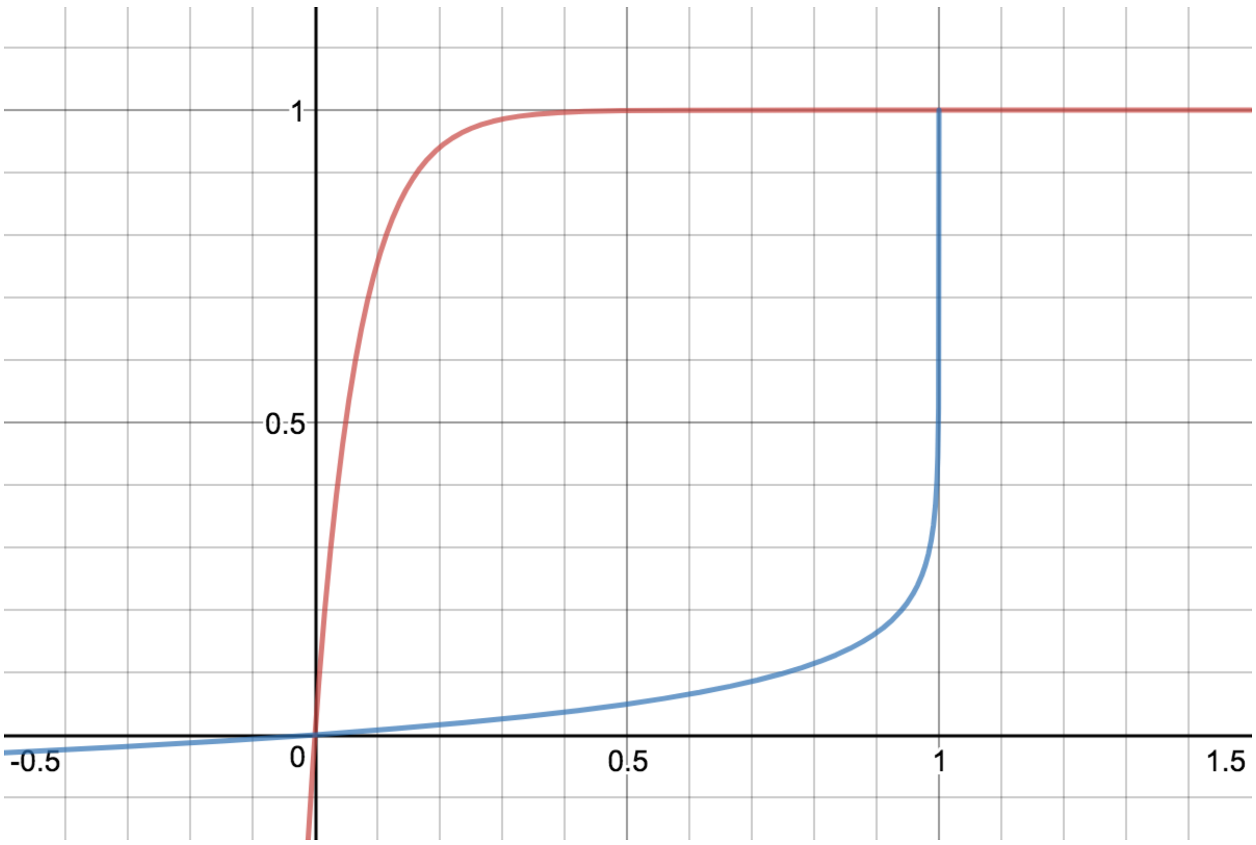
bulk

With the operation type `bulk` you can execute bulk requests. It supports the following properties:

- `bulk-size` (mandatory): Defines the bulk size in number of documents.

- `ingest-percentage` (optional, defaults to 100): A number between (0, 100] that defines how much of the document corpus will be bulk-indexed.
- `corpora` (optional): A list of document corpus names that should be targeted by this bulk-index operation. Only needed if the `corpora` section contains more than one document corpus and you don't want to index all of them with this operation.
- `indices` (optional): A list of index names that defines which indices should be used by this bulk-index operation. Rally will then only select the documents files that have a matching `target-index` specified.
- `batch-size` (optional): Defines how many documents Rally will read at once. This is an expert setting and only meant to avoid accidental bottlenecks for very small bulk sizes (e.g. if you want to benchmark with a bulk-size of 1, you should set `batch-size` higher).
- `pipeline` (optional): Defines the name of an (existing) ingest pipeline that should be used (only supported from Elasticsearch 5.0).
- `conflicts` (optional): Type of index conflicts to simulate. If not specified, no conflicts will be simulated (also read below on how to use external index ids with no conflicts). Valid values are: 'sequential' (A document id is replaced with a document id with a sequentially increasing id), 'random' (A document id is replaced with a document id with a random other id).
- `conflict-probability` (optional, defaults to 25 percent): A number between [0, 100] that defines how many of the documents will get replaced. Combining `conflicts=sequential` and `conflict-probability=0` makes Rally generate index ids by itself, instead of relying on Elasticsearch's automatic id generation.
- `on-conflict` (optional, defaults to `index`): Determines whether Rally should use the action `index` or `update` on id conflicts.
- `recency` (optional, defaults to 0): A number between [0,1] indicating whether to bias conflicting ids towards more recent ids (`recency` towards 1) or whether to consider all ids for id conflicts (`recency` towards 0). See the diagram below for details.
- `detailed-results` (optional, defaults to `false`): Records more detailed meta-data for bulk requests. As it analyzes the corresponding bulk response in more detail, this might incur additional overhead which can skew measurement results.

The image below shows how Rally behaves with a `recency` set to 0.5. Internally, Rally uses the blue function for its calculations but to understand the behavior we will focus on red function (which is just the inverse). Suppose we have already generated ids from 1 to 100 and we are about to simulate an id conflict. Rally will randomly choose a value on the y-axis, e.g. 0.8 which is mapped to 0.1 on the x-axis. This means that in 80% of all cases, Rally will choose an id within the most recent 10%, i.e. between 90 and 100. With 20% probability the id will be between 1 and 89. The closer `recency` gets to zero, the "flatter" the red curve gets and the more likely Rally will choose less recent ids.



You can also explore the recency calculation interactively.

Example:

```
{
  "name": "index-append",
  "operation-type": "bulk",
  "bulk-size": 5000
}
```

Throughput will be reported as number of indexed documents per second.

force-merge

With the operation type `force-merge` you can call the force merge API. On older versions of Elasticsearch (prior to 2.1), Rally will use the `optimize API` instead. It supports the following parameters:

- `index` (optional, defaults to the indices defined in the `indices` section or `_all` if no indices are defined there): The name of the index that should be force-merged.
- `mode` (optional, default to `blocking`): In the default `blocking` mode the Elasticsearch client blocks until the operation returns or times out as dictated by the *client-options*. In mode *polling* the client timeout is ignored. Instead, the api call is given 1s to complete. If the operation has not finished, the operator will poll every `poll-period` until all force merges are complete.
- `poll-period` (defaults to 10s): Only applicable if `mode` is set to `polling`. Determines the internal at which a check is performed that all force merge operations are complete.

- `max-num-segments` (optional) The number of segments the index should be merged into. Defaults to simply checking if a merge needs to execute, and if so, executes it.

This is an administrative operation. Metrics are not reported by default. If reporting is forced by setting `include-in-reporting` to `true`, then throughput is reported as the number of completed force-merge operations per second.

index-stats

With the operation type `index-stats` you can call the indices stats API. It supports the following properties:

- `index` (optional, defaults to *_all*): An index pattern that defines which indices should be targeted by this operation.
- `condition` (optional, defaults to no condition): A structured object with the properties `path` and `expected-value`. If the actual value returned by indices stats API is equal to the expected value at the provided path, this operation will return successfully. See below for an example how this can be used.

In the following example the `index-stats` operation will wait until all segments have been merged:

```
{
    "operation-type": "index-stats",
    "index": "_all",
    "condition": {
        "path": "_all.total.merges.current",
        "expected-value": 0
    },
    "retry-until-success": true
}
```

Throughput will be reported as number of completed *index-stats* operations per second.

This operation is *retryable*.

node-stats

With the operation type `nodes-stats` you can execute nodes stats API. It does not support any parameters.

Throughput will be reported as number of completed *node-stats* operations per second.

search

With the operation type `search` you can execute request body searches. It supports the following properties:

- `index` (optional): An index pattern that defines which indices should be targeted by this query. Only needed if the `index` section contains more than one index. Otherwise, Rally will automatically derive the index to use. If you have defined multiple indices and want to query all of them, just specify `"index": "_all"`.
- `type` (optional): Defines the type within the specified index for this query. By default, no `type` will be used and the query will be performed across all types in the provided index. Also, types have been removed in Elasticsearch 7.0.0 so you must not specify this property if you want to benchmark Elasticsearch 7.0.0 or later.
- `cache` (optional): Whether to use the query request cache. By default, Rally will define no value thus the default depends on the benchmark candidate settings and Elasticsearch version.
- `request-params` (optional): A structure containing arbitrary request parameters. The supported parameters names are documented in the Search URI Request docs.

Note:

1. Parameters that are implicitly set by Rally (e.g. *body* or *request_cache*) are not supported (i.e. you should not try to set them and if so expect unspecified behavior).
2. Rally will not attempt to serialize the parameters and pass them as is. Always use "true" / "false" strings for boolean parameters (see example below).

- `body` (mandatory): The query body.
- `response-compression-enabled` (optional, defaults to `true`): Allows to disable HTTP compression of responses. As these responses are sometimes large and decompression may be a bottleneck on the client, it is possible to turn off response compression.
- `detailed-results` (optional, defaults to `false`): Records more detailed meta-data about queries. As it analyzes the corresponding response in more detail, this might incur additional overhead which can skew measurement results. This flag is ineffective for scroll queries.
- `pages` (optional): Number of pages to retrieve. If this parameter is present, a scroll query will be executed. If you want to retrieve all result pages, use the value "all".
- `results-per-page` (optional): Number of documents to retrieve per page for scroll queries.

If `detailed-results` is set to `true`, the following meta-data properties will be determined and stored:

- `hits`
- `hits_relation`
- `timed_out`
- `took`

Example:

```
{
  "name": "default",
  "operation-type": "search",
  "body": {
    "query": {
      "match_all": {}
    }
  },
  "request-params": {
    "_source_include": "some_field",
    "analyze_wildcard": "false"
  }
}
```

For scroll queries, throughput will be reported as number of retrieved scroll pages per second. The unit is ops/s, where one op(eration) is one page that has been retrieved. The rationale is that each HTTP request corresponds to one operation and we need to issue one HTTP request per result page. Note that if you use a dedicated Elasticsearch metrics store, you can also use other request-level meta-data such as the number of hits for your own analyses.

For other queries, throughput will be reported as number of search requests per second, also measured as ops/s.

put-pipeline

With the operation-type `put-pipeline` you can execute the put pipeline API. Note that this API is only available from Elasticsearch 5.0 onwards. It supports the following properties:

- *id* (mandatory): Pipeline id
- *body* (mandatory): Pipeline definition

In this example we setup a pipeline that adds location information to a ingested document as well as a pipeline failure block to change the index in which the document was supposed to be written. Note that we need to use the `raw` and `endraw` blocks to ensure that Rally does not attempt to resolve the Mustache template. See *template language* for more information.

Example:

```
{
  "name": "define-ip-geocoder",
  "operation-type": "put-pipeline",
  "id": "ip-geocoder",
  "body": {
    "description": "Extracts location information from the client IP.",
    "processors": [
      {
        "geoip": {
          "field": "clientip",
          "properties": [
            "city_name",
            "country_iso_code",
            "country_name",
            "location"
          ]
        }
      }
    ],
    "on_failure": [
      {
        "set": {
          "field": "_index",
          {% raw %}
          "value": "failed-{{ _index }}"
          {% endraw %}
        }
      }
    ]
  }
}
```

Please see the pipeline documentation for details on handling failures in pipelines.

This example requires that the `ingest-geoip` Elasticsearch plugin is installed.

This is an administrative operation. Metrics are not reported by default. Reporting can be forced by setting `include-in-reporting` to `true`.

This operation is *retryable*.

put-settings

With the operation-type `put-settings` you can execute the cluster update settings API. It supports the following properties:

- *body* (mandatory): The cluster settings to apply.

Example:

```
{
  "name": "increase-watermarks",
  "operation-type": "put-settings",
  "body": {
    "transient" : {
        "cluster.routing.allocation.disk.watermark.low" : "95%",
        "cluster.routing.allocation.disk.watermark.high" : "97%",
        "cluster.routing.allocation.disk.watermark.flood_stage" : "99%"
    }
  }
}
```

This is an administrative operation. Metrics are not reported by default. Reporting can be forced by setting `include-in-reporting` to `true`.

This operation is *retryable*.

cluster-health

With the operation `cluster-health` you can execute the cluster health API. It supports the following properties:

- `request-params` (optional): A structure containing any request parameters that are allowed by the cluster health API. Rally will not attempt to serialize the parameters and pass them as is. Always use “true” / “false” strings for boolean parameters (see example below).
- `index` (optional): The name of the index that should be used to check.

The `cluster-health` operation will check whether the expected cluster health and will report a failure if this is not the case. Use `--on-error` on the command line to control Rally’s behavior in case of such failures.

Example:

```
{
  "name": "check-cluster-green",
  "operation-type": "cluster-health",
  "index": "logs-*",
  "request-params": {
    "wait_for_status": "green",
    "wait_for_no_relocating_shards": "true"
  }
}
```

This is an administrative operation. Metrics are not reported by default. Reporting can be forced by setting `include-in-reporting` to `true`.

This operation is *retryable*.

refresh

With the operation `refresh` you can execute the refresh API. It supports the following properties:

- `index` (optional, defaults to `_all`): The name of the index that should be refreshed.

This is an administrative operation. Metrics are not reported by default. Reporting can be forced by setting `include-in-reporting` to `true`.

This operation is *retryable*.

create-index

With the operation `create-index` you can execute the create index API. It supports two modes: it creates either all indices that are specified in the track's `indices` section or it creates one specific index defined by this operation.

If you want it to create all indices that have been declared in the `indices` section you can specify the following properties:

- `settings` (optional): Allows to specify additional index settings that will be merged with the index settings specified in the body of the index in the `indices` section.
- `request-params` (optional): A structure containing any request parameters that are allowed by the create index API. Rally will not attempt to serialize the parameters and pass them as is. Always use "true" / "false" strings for boolean parameters (see example below).

If you want it to create one specific index instead, you can specify the following properties:

- `index` (mandatory): One or more names of the indices that should be created. If only one index should be created, you can use a string otherwise this needs to be a list of strings.
- `body` (optional): The body for the create index API call.
- `request-params` (optional): A structure containing any request parameters that are allowed by the create index API. Rally will not attempt to serialize the parameters and pass them as is. Always use "true" / "false" strings for boolean parameters (see example below).

Examples

The following snippet will create all indices that have been defined in the `indices` section. It will reuse all settings defined but override the number of shards:

```
{
  "name": "create-all-indices",
  "operation-type": "create-index",
  "settings": {
    "index.number_of_shards": 1
  },
  "request-params": {
    "wait_for_active_shards": "true"
  }
}
```

With the following snippet we will create a new index that is not defined in the `indices` section. Note that we specify the index settings directly in the body:

```
{
  "name": "create-an-index",
  "operation-type": "create-index",
```

(continues on next page)

(continued from previous page)

```
  "index": "people",
  "body": {
    "settings": {
      "index.number_of_shards": 0
    },
    "mappings": {
      "docs": {
        "properties": {
          "name": {
            "type": "text"
          }
        }
      }
    }
  }
}
```

Note: Types have been removed in Elasticsearch 7.0.0. If you want to benchmark Elasticsearch 7.0.0 or later you need to remove the mapping type above.

This is an administrative operation. Metrics are not reported by default. Reporting can be forced by setting `include-in-reporting` to `true`.

This operation is *retryable*.

delete-index

With the operation `delete-index` you can execute the delete index API. It supports two modes: it deletes either all indices that are specified in the track's `indices` section or it deletes one specific index (pattern) defined by this operation.

If you want it to delete all indices that have been declared in the `indices` section, you can specify the following properties:

- `only-if-exists` (optional, defaults to `true`): Defines whether an index should only be deleted if it exists.
- `request-params` (optional): A structure containing any request parameters that are allowed by the delete index API. Rally will not attempt to serialize the parameters and pass them as is. Always use "true" / "false" strings for boolean parameters (see example below).

If you want it to delete one specific index (pattern) instead, you can specify the following properties:

- `index` (mandatory): One or more names of the indices that should be deleted. If only one index should be deleted, you can use a string otherwise this needs to be a list of strings.
- `only-if-exists` (optional, defaults to `true`): Defines whether an index should only be deleted if it exists.
- `request-params` (optional): A structure containing any request parameters that are allowed by the delete index API. Rally will not attempt to serialize the parameters and pass them as is. Always use "true" / "false" strings for boolean parameters (see example below).

Examples

With the following snippet we will delete all indices that are declared in the `indices` section but only if they existed previously (implicit default):

```
{
  "name": "delete-all-indices",
  "operation-type": "delete-index"
}
```

With the following snippet we will delete all `logs-*` indices:

```
{
  "name": "delete-logs",
  "operation-type": "delete-index",
  "index": "logs-*",
  "only-if-exists": false,
  "request-params": {
    "expand_wildcards": "all",
    "allow_no_indices": "true",
    "ignore_unavailable": "true"
  }
}
```

This is an administrative operation. Metrics are not reported by default. Reporting can be forced by setting `include-in-reporting` to `true`.

This operation is *retryable*.

create-index-template

With the operation `create-index-template` you can execute the create index template API. It supports two modes: it creates either all index templates that are specified in the track's `templates` section or it creates one specific index template defined by this operation.

If you want it to create index templates that have been declared in the `templates` section you can specify the following properties:

- `template` (optional): If you specify a template name, only the template with this name will be created.
- `settings` (optional): Allows to specify additional settings that will be merged with the settings specified in the body of the index template in the `templates` section.
- `request-params` (optional): A structure containing any request parameters that are allowed by the create index template API. Rally will not attempt to serialize the parameters and pass them as is. Always use "true" / "false" strings for boolean parameters (see example below).

If you want it to create one specific index instead, you can specify the following properties:

- `template` (mandatory): The name of the index template that should be created.
- `body` (mandatory): The body for the create index template API call.
- `request-params` (optional): A structure containing any request parameters that are allowed by the create index template API. Rally will not attempt to serialize the parameters and pass them as is. Always use "true" / "false" strings for boolean parameters (see example below).

Examples

The following snippet will create all index templates that have been defined in the `templates` section:

```
{
  "name": "create-all-templates",
  "operation-type": "create-index-template",
```

(continues on next page)

(continued from previous page)

```
  "request-params": {
    "create": "true"
  }
}
```

With the following snippet we will create a new index template that is not defined in the `templates` section. Note that we specify the index template settings directly in the body:

```
{
  "name": "create-a-template",
  "operation-type": "create-index-template",
  "template": "defaults",
  "body": {
    "index_patterns": ["*"],
    "settings": {
      "number_of_shards": 3
    },
    "mappings": {
      "docs": {
        "_source": {
          "enabled": false
        }
      }
    }
  }
}
```

Note: Types have been removed in Elasticsearch 7.0.0. If you want to benchmark Elasticsearch 7.0.0 or later you need to remove the mapping type above.

This is an administrative operation. Metrics are not reported by default. Reporting can be forced by setting `include-in-reporting` to `true`.

This operation is *retryable*.

delete-index-template

With the operation `delete-index-template` you can execute the delete index template API. It supports two modes: it deletes either all index templates that are specified in the track's `templates` section or it deletes one specific index template defined by this operation.

If you want it to delete all index templates that have been declared in the `templates` section, you can specify the following properties:

- `only-if-exists` (optional, defaults to `true`): Defines whether an index template should only be deleted if it exists.
- `request-params` (optional): A structure containing any request parameters that are allowed by the delete index template API. Rally will not attempt to serialize the parameters and pass them as is. Always use "true" / "false" strings for boolean parameters.

If you want it to delete one specific index template instead, you can specify the following properties:

- `template` (mandatory): The name of the index that should be deleted.

- `only-if-exists` (optional, defaults to `true`): Defines whether the index template should only be deleted if it exists.
- `delete-matching-indices` (optional, defaults to `false`): Whether to delete indices that match the index template's index pattern.
- `index-pattern` (mandatory iff `delete-matching-indices` is `true`): Specifies the index pattern to delete.
- `request-params` (optional): A structure containing any request parameters that are allowed by the delete index template API. Rally will not attempt to serialize the parameters and pass them as is. Always use "true" / "false" strings for boolean parameters.

Examples

With the following snippet we will delete all index templates that are declared in the `templates` section but only if they existed previously (implicit default):

```
{
  "name": "delete-all-index-templates",
  "operation-type": "delete-index-template"
}
```

With the following snippet we will delete the *default*' index template:

```
{
  "name": "delete-default-template",
  "operation-type": "delete-index-template",
  "template": "default",
  "only-if-exists": false,
  "delete-matching-indices": true,
  "index-pattern": "*"
}
```

Note: If `delete-matching-indices` is set to `true`, indices with the provided `index-pattern` are deleted regardless whether the index template has previously existed.

This is an administrative operation. Metrics are not reported by default. Reporting can be forced by setting `include-in-reporting` to `true`.

This operation is *retryable*.

shrink-index

With the operation `shrink-index` you can execute the shrink index API. Note that this does not correspond directly to the shrink index API call in Elasticsearch but it is a high-level operation that executes all the necessary low-level operations under the hood to shrink an index. It supports the following parameters:

- `source-index` (mandatory): The name of the index that should be shrinked.
- `target-index` (mandatory): The name of the index that contains the shrinked shards.
- `target-body` (mandatory): The body containing settings and aliases for `target-index`.
- `shrink-node` (optional, defaults to a random data node): As a first step, the source index needs to be fully relocated to a single node. Rally will automatically choose a random data node in the cluster but you can choose one explicitly if needed.

Example:

```
{
  "operation-type": "shrink-index",
  "shrink-node": "rally-node-0",
  "source-index": "src",
  "target-index": "target",
  "target-body": {
    "settings": {
      "index.number_of_replicas": 1,
      "index.number_of_shards": 1,
      "index.codec": "best_compression"
    }
  }
}
```

This will shrink the index `src` to `target`. The target index will consist of one shard and have one replica. With `shrink-node` we also explicitly specify the name of the node where we want the source index to be relocated to.

This operation is *retryable*.

delete-ml-datafeed

With the operation `delete-ml-datafeed` you can execute the delete datafeeds API. The `delete-ml-datafeed` operation supports the following parameters:

- `datafeed-id` (mandatory): The name of the machine learning datafeed to delete.
- `force` (optional, defaults to `false`): Whether to force deletion of a datafeed that has already been started.

This runner will intentionally ignore 404s from Elasticsearch so it is safe to execute this runner regardless whether a corresponding machine learning datafeed exists.

This operation works only if machine-learning is properly installed and enabled. This is an administrative operation. Metrics are not reported by default. Reporting can be forced by setting `include-in-reporting` to `true`.

create-ml-datafeed

With the operation `create-ml-datafeed` you can execute the create datafeeds API. The `create-ml-datafeed` operation supports the following parameters:

- `datafeed-id` (mandatory): The name of the machine learning datafeed to create.
- `body` (mandatory): Request body containing the definition of the datafeed. Please see the create datafeed API documentation for more details.

This operation works only if machine-learning is properly installed and enabled. This is an administrative operation. Metrics are not reported by default. Reporting can be forced by setting `include-in-reporting` to `true`.

This operation is *retryable*.

start-ml-datafeed

With the operation `start-ml-datafeed` you can execute the start datafeeds API. The `start-ml-datafeed` operation supports the following parameters which are documented in the start datafeed API documentation:

- `datafeed-id` (mandatory): The name of the machine learning datafeed to start.

- `body` (optional, defaults to empty): Request body with start parameters.
- `start` (optional, defaults to empty): Start timestamp of the datafeed.
- `end` (optional, defaults to empty): End timestamp of the datafeed.
- `timeout` (optional, defaults to empty): Amount of time to wait until a datafeed starts.

This operation works only if machine-learning is properly installed and enabled. This is an administrative operation. Metrics are not reported by default. Reporting can be forced by setting `include-in-reporting` to `true`.

This operation is *retryable*.

stop-ml-datafeed

With the operation `stop-ml-datafeed` you can execute the stop datafeed API. The `stop-ml-datafeed` operation supports the following parameters:

- `datafeed-id` (mandatory): The name of the machine learning datafeed to start.
- `force` (optional, defaults to `false`): Whether to forcefully stop an already started datafeed.
- `timeout` (optional, defaults to empty): Amount of time to wait until a datafeed stops.

This operation works only if machine-learning is properly installed and enabled. This is an administrative operation. Metrics are not reported by default. Reporting can be forced by setting `include-in-reporting` to `true`.

This operation is *retryable*.

delete-ml-job

With the operation `delete-ml-job` you can execute the delete jobs API. The `delete-ml-job` operation supports the following parameters:

- `job-id` (mandatory): The name of the machine learning job to delete.
- `force` (optional, defaults to `false`): Whether to force deletion of a job that has already been opened.

This runner will intentionally ignore 404s from Elasticsearch so it is safe to execute this runner regardless whether a corresponding machine learning job exists.

This operation works only if machine-learning is properly installed and enabled. This is an administrative operation. Metrics are not reported by default. Reporting can be forced by setting `include-in-reporting` to `true`.

This operation is *retryable*.

create-ml-job

With the operation `create-ml-job` you can execute the create jobs API. The `create-ml-job` operation supports the following parameters:

- `job-id` (mandatory): The name of the machine learning job to create.
- `body` (mandatory): Request body containing the definition of the job. Please see the create job API documentation for more details.

This operation works only if machine-learning is properly installed and enabled. This is an administrative operation. Metrics are not reported by default. Reporting can be forced by setting `include-in-reporting` to `true`.

This operation is *retryable*.

open-ml-job

With the operation `open-ml-job` you can execute the open jobs API. The `open-ml-job` operation supports the following parameters:

- `job-id` (mandatory): The name of the machine learning job to open.

This operation works only if machine-learning is properly installed and enabled. This is an administrative operation. Metrics are not reported by default. Reporting can be forced by setting `include-in-reporting` to `true`.

This operation is *retryable*.

close-ml-job

With the operation `close-ml-job` you can execute the *close jobs API. The ‘‘close-ml-job‘* operation supports the following parameters:

- `job-id` (mandatory): The name of the machine learning job to start.
- `force` (optional, defaults to `false`): Whether to forcefully stop an already opened job.
- `timeout` (optional, defaults to empty): Amount of time to wait until a job stops.

This operation works only if machine-learning is properly installed and enabled. This is an administrative operation. Metrics are not reported by default. Reporting can be forced by setting `include-in-reporting` to `true`.

This operation is *retryable*.

raw-request

With the operation `raw-request` you can execute arbitrary HTTP requests against Elasticsearch. This is a low-level operation that should only be used if no high-level operation is available. Note that it is always possible to write a *custom runner*. The `raw-request` operation supports the following parameters:

- `method` (optional, defaults to `GET`): The HTTP request method to use
- `path` (mandatory): Path for the API call (excluding host and port). The path must begin with a `/`. Example: `/myindex/_flush`.
- `header` (optional): A structure containing any request headers as key-value pairs.
- `body` (optional): The document body.
- `request-params` (optional): A structure containing HTTP request parameters.
- `ignore` (optional): An array of HTTP response status codes to ignore (i.e. consider as successful).

sleep

With the operation `sleep` you can sleep for a certain duration to ensure no requests are executed by the corresponding clients. The `sleep` operation supports the following parameter:

- `duration` (mandatory): A non-negative number that defines the sleep duration in seconds.

Note: The `sleep` operation is only useful in very limited circumstances. To throttle throughput, specify a `target-throughput` on the corresponding task instead.

This is an administrative operation. Metrics are not reported by default. Reporting can be forced by setting `include-in-reporting` to `true`.

delete-snapshot-repository

With the operation `delete-snapshot-repository` you can delete an existing snapshot repository. The `delete-snapshot-repository` operation supports the following parameter:

- `repository` (mandatory): The name of the snapshot repository to delete.

This is an administrative operation. Metrics are not reported by default. Reporting can be forced by setting `include-in-reporting` to `true`.

This operation is *retryable*.

create-snapshot-repository

With the operation `create-snapshot-repository` you can create a new snapshot repository. The `create-snapshot-repository` operation supports the following parameters:

- `repository` (mandatory): The name of the snapshot repository to create.
- `body` (mandatory): The body of the create snapshot repository request.
- `request-params` (optional): A structure containing HTTP request parameters.

This is an administrative operation. Metrics are not reported by default. Reporting can be forced by setting `include-in-reporting` to `true`.

This operation is *retryable*.

create-snapshot

With the operation `create-snapshot` you can create a snapshot. The `create-snapshot` operation supports the following parameters:

- `repository` (mandatory): The name of the snapshot repository to use.
- `snapshot` (mandatory): The name of the snapshot to create.
- `body` (mandatory): The body of the create snapshot request.
- `wait-for-completion` (optional, defaults to `False`): Whether this call should return immediately or block until the snapshot is created.
- `request-params` (optional): A structure containing HTTP request parameters.

Note: It's not recommended to rely on `wait-for-completion=true`. Instead you should keep the default value (`False`) and use an additional `wait-for-snapshot-create` operation in the next step. This is mandatory on Elastic Cloud or environments where Elasticsearch is connected via intermediate network components, such as proxies, that may terminate the blocking connection after a timeout.

wait-for-snapshot-create

With the operation `wait-for-snapshot-create` you can wait until a snapshot has finished successfully. Typically you'll use this operation directly after a `create-snapshot` operation.

It supports the following parameters:

- `repository` (mandatory): The name of the snapshot repository to use.
- `snapshot` (mandatory): The name of the snapshot that this operation will wait until it succeeds.
- `completion-recheck-wait-period` (optional, defaults to 1 second): Time in seconds to wait in between consecutive attempts.

Rally will report the achieved throughput in byte/s.

This operation is *retryable*.

restore-snapshot

With the operation `restore-snapshot` you can restore a snapshot from an already created snapshot repository. The `restore-snapshot` operation supports the following parameters:

- `repository` (mandatory): The name of the snapshot repository to use. This snapshot repository must exist prior to calling `restore-snapshot`.
- `snapshot` (mandatory): The name of the snapshot to restore.
- `body` (optional): The body of the snapshot restore request.
- `wait-for-completion` (optional, defaults to `False`): Whether this call should return immediately or block until the snapshot is restored.
- `request-params` (optional): A structure containing HTTP request parameters.

Note: In order to ensure that the track execution only continues after a snapshot has been restored, set `wait-for-completion` to `true` **and** increase the request timeout. In the example below we set it to 7200 seconds (or 2 hours):

```
"request-params": {
    "request_timeout": 7200
}
```

However, this might not work if a proxy is in between the client and Elasticsearch and the proxy has a shorter request timeout configured than the client. In this case, keep the default value for `wait-for-completion` and instead add a `wait-for-recovery` runner in the next step.

wait-for-recovery

With the operation `wait-for-recovery` you can wait until an ongoing shard recovery finishes. The `wait-for-recovery` operation supports the following parameters:

- `index` (mandatory): The name of the index or an index pattern which is being recovered.
- `completion-recheck-wait-period` (optional, defaults to 1 seconds): Time in seconds to wait in between consecutive attempts.

This operation is *retryable*.

create-transform

With the operation `create-transform` you can execute the create transform API. It supports the following parameters:

- `transform-id` (mandatory): The id of the transform to create.
- `body` (mandatory): Request body containing the configuration of the transform. Please see the create transform API documentation for more details.
- `defer-validation` (optional, defaults to false): When true, deferrable validations are not run. This behavior may be desired if the source index does not exist until after the transform is created.

This operation requires at least Elasticsearch 7.5.0 (non-OSS). This is an administrative operation. Metrics are not reported by default. Reporting can be forced by setting `include-in-reporting` to `true`.

This operation is *retryable*.

start-transform

With the operation `start-transform` you can execute the start transform API. It supports the following parameters:

- `transform-id` (mandatory): The id of the transform to start.
- `timeout` (optional, defaults to empty): Amount of time to wait until a transform starts.

This operation requires at least Elasticsearch 7.5.0 (non-OSS). This is an administrative operation. Metrics are not reported by default. Reporting can be forced by setting `include-in-reporting` to `true`.

This operation is *retryable*.

wait-for-transform

With the operation `wait-for-transform` you can stop a transform after a certain amount of work is done. Use this operation for measuring performance. It supports the following parameters:

- `transform-id` (mandatory): The id of the transform to stop.
- `force` (optional, defaults to false): Whether to forcefully stop the transform.
- `timeout` (optional, defaults to empty): Amount of time to wait until a transform stops.
- `wait-for-completion` (optional, defaults to true) If set to true, causes the API to block until the indexer state completely stops.
- `wait-for-checkpoint` (optional, defaults to true) If set to true, the transform will not completely stop until the current checkpoint is completed.
- `transform-timeout` (optional, defaults to *3600* (*1h*)) Overall runtime timeout of the batch transform in seconds.
- `poll-interval` (optional, defaults to *0.5*) How often transform stats are polled, used to set progress and check the state. You should not set this too low, because polling can skew the result.

This operation requires at least Elasticsearch 7.5.0 (non-OSS). This is an administrative operation. Metrics are not reported by default. Reporting can be forced by setting `include-in-reporting` to `true`.

This operation is *retryable*.

delete-transform

With the operation `delete-transform` you can execute the delete transform API. It supports the following parameters:

- `transform-id` (mandatory): The id of the transform to delete.
- `force` (optional, defaults to false): Whether to delete the transform regardless of its current state.

This operation requires at least Elasticsearch 7.5.0 (non-OSS). This is an administrative operation. Metrics are not reported by default. Reporting can be forced by setting `include-in-reporting` to `true`.

This operation is *retryable*.

2.14.5 Examples

A track with a single task

To get started with custom tracks, you can benchmark a single task, e.g. a match_all query:

```
{
  "schedule": [
    {
      "operation": {
        "operation-type": "search",
        "index": "_all",
        "body": {
          "query": {
            "match_all": {}
          }
        }
      },
      "warmup-iterations": 100,
      "iterations": 100,
      "target-throughput": 10
    }
  ]
}
```

This track assumes that you have an existing cluster with pre-populated data. It will run the provided `match_all` query at 10 operations per second with one client and use 100 iterations as warmup and the next 100 iterations to measure.

For the examples below, note that we do not show the operation definition but you should be able to infer from the operation name what it is doing.

Running unthrottled

In this example Rally will run a bulk index operation unthrottled for one hour:

```
"schedule": [
  {
    "operation": "bulk",
    "warmup-time-period": 120,
    "time-period": 3600,
    "clients": 8
```

(continues on next page)

(continued from previous page)

```
  }
]
```

Running tasks in parallel

Note: You cannot nest parallel tasks.

If we want to run tasks in parallel, we can use the `parallel` element. In the simplest case, you let Rally decide the number of clients needed to run the parallel tasks (note how we can define default values on the `parallel` element):

```
    {
      "parallel": {
        "warmup-iterations": 50,
        "iterations": 100,
        "tasks": [
          {
            "operation": "default",
            "target-throughput": 50
          },
          {
            "operation": "term",
            "target-throughput": 200
          },
          {
            "operation": "phrase",
            "target-throughput": 200
          }
        ]
      }
    }
  ]
}
```

Rally will determine that three clients are needed to run each task in a dedicated client. You can also see that each task can have different settings.

However, you can also explicitly define the number of clients:

```
"schedule": [
  {
    "parallel": {
      "warmup-iterations": 50,
      "iterations": 100,
      "tasks": [
        {
          "operation": "match-all",
          "clients": 4,
          "target-throughput": 50
        },
        {
          "operation": "term",
          "clients": 2,
          "target-throughput": 200
```

(continues on next page)

(continued from previous page)

```
        },
        {
          "operation": "phrase",
          "clients": 2,
          "target-throughput": 200
        }
      ]
    }
  }
]
```

This schedule will run a match all query, a term query and a phrase query concurrently. It will run with eight clients in total (four for the match all query and two each for the term and phrase query).

In this scenario, we run indexing and a few queries in parallel with a total of 14 clients:

```
"schedule": [
  {
    "parallel": {
      "tasks": [
        {
          "operation": "bulk",
          "warmup-time-period": 120,
          "time-period": 3600,
          "clients": 8,
          "target-throughput": 50
        },
        {
          "operation": "default",
          "clients": 2,
          "warmup-iterations": 50,
          "iterations": 100,
          "target-throughput": 50
        },
        {
          "operation": "term",
          "clients": 2,
          "warmup-iterations": 50,
          "iterations": 100,
          "target-throughput": 200
        },
        {
          "operation": "phrase",
          "clients": 2,
          "warmup-iterations": 50,
          "iterations": 100,
          "target-throughput": 200
        }
      ]
    }
  }
]
```

We can use `completed-by` to stop querying as soon as bulk-indexing has completed:

```
"schedule": [
  {
```

(continues on next page)

(continued from previous page)

```
    "parallel": {
      "completed-by": "bulk",
      "tasks": [
        {
          "operation": "bulk",
          "warmup-time-period": 120,
          "time-period": 3600,
          "clients": 8,
          "target-throughput": 50
        },
        {
          "operation": "default",
          "clients": 2,
          "warmup-time-period": 480,
          "time-period": 7200,
          "target-throughput": 50
        }
      ]
    }
  }
]
```

We can also mix sequential tasks with the `parallel` element. In this scenario we are indexing with 8 clients and continue querying with 6 clients after indexing has finished:

```
"schedule": [
  {
    "operation": "bulk",
    "warmup-time-period": 120,
    "time-period": 3600,
    "clients": 8,
    "target-throughput": 50
  },
  {
    "parallel": {
      "warmup-iterations": 50,
      "iterations": 100,
      "tasks": [
        {
          "operation": "default",
          "clients": 2,
          "target-throughput": 50
        },
        {
          "operation": "term",
          "clients": 2,
          "target-throughput": 200
        },
        {
          "operation": "phrase",
          "clients": 2,
          "target-throughput": 200
        }
      ]
    }
  }
]
```

Be aware of the following case where we explicitly define that we want to run only with two clients *in total*:

```
"schedule": [
  {
    "parallel": {
      "warmup-iterations": 50,
      "iterations": 100,
      "clients": 2,
      "tasks": [
        {
          "operation": "match-all",
          "target-throughput": 50
        },
        {
          "operation": "term",
          "target-throughput": 200
        },
        {
          "operation": "phrase",
          "target-throughput": 200
        }
      ]
    }
  }
]
```

Rally will *not* run all three tasks in parallel because you specified that you want only two clients in total. Hence, Rally will first run "match-all" and "term" concurrently (with one client each). After they have finished, Rally will run "phrase" with one client. You could also specify more clients than there are tasks but these will then just idle.

You can also specify a number of clients on sub tasks explicitly (by default, one client is assumed per subtask). This allows to define a weight for each client operation. Note that you need to define the number of clients also on the `parallel` parent element, otherwise Rally would determine the number of total needed clients again on its own:

```
{
  "parallel": {
    "clients": 3,
    "warmup-iterations": 50,
    "iterations": 100,
    "tasks": [
      {
        "operation": "default",
        "target-throughput": 50
      },
      {
        "operation": "term",
        "target-throughput": 200
      },
      {
        "operation": "phrase",
        "target-throughput": 200,
        "clients": 2
      }
    ]
  }
}
```

This will ensure that the phrase query will be executed by two clients. All other ones are executed by one client.

2.15 Configure Elasticsearch: Cars

Note: You can skip this section if you use Rally only as a load generator.

2.15.1 Definition

A Rally "car" is a specific configuration of Elasticsearch. You can list the available cars with `esrally list cars`:

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

Name                       Type    Description
-----------------------    ------  ----------------------------------
16gheap                    car     Sets the Java heap to 16GB
1gheap                     car     Sets the Java heap to 1GB
2gheap                     car     Sets the Java heap to 2GB
4gheap                     car     Sets the Java heap to 4GB
8gheap                     car     Sets the Java heap to 8GB
defaults                   car     Sets the Java heap to 1GB
ea                         mixin   Enables Java assertions
fp                         mixin   Preserves frame pointers
x-pack-ml                  mixin   X-Pack Machine Learning
x-pack-monitoring-http     mixin   X-Pack Monitoring (HTTP exporter)
x-pack-monitoring-local    mixin   X-Pack Monitoring (local exporter)
x-pack-security            mixin   X-Pack Security
```

You can specify the car that Rally should use with e.g. `--car="4gheap"`. It is also possible to specify one or more "mixins" to further customize the configuration. For example, you can specify `--car="4gheap,ea"` to run with a 4GB heap and enable Java assertions (they are disabled by default) or `--car="4gheap,x-pack-security"` to benchmark Elasticsearch with X-Pack Security enabled (requires Elasticsearch 6.3.0 or better).

Note: To benchmark `x-pack-security` you need to add the following command line options: `--client-options="use_ssl:true,verify_certs:false,basic_auth_user:'rally', basic_auth_password:'rally-password'"`

Similar to *custom tracks*, you can also define your own cars.

2.15.2 The Anatomy of a car

The default car definitions of Rally are stored in `~/.rally/benchmarks/teams/default/cars`. There we find the following structure:

```
.
└── v1
    ├── 1gheap.ini
    ├── 2gheap.ini
```

(continues on next page)

(continued from previous page)

```
    ├── defaults.ini
    ├── ea
    │   └── templates
    │       └── config
    │           └── jvm.options
    ├── ea.ini
    └── vanilla
        ├── config.ini
        └── templates
            └── config
                ├── elasticsearch.yml
                ├── jvm.options
                └── log4j2.properties
```

The top-level directory "v1" denotes the configuration format in version 1. Below that directory, each `.ini` file defines a car. Each directory (`ea` or `vanilla`) contains templates for the config files. Rally will only copy the files in the `templates` subdirectory. The top-level directory is reserved for a special file, `config.ini` which you can use to define default variables that apply to all cars that are based on this configuration. Below is an example `config.ini` file:

```
[variables]
clean_command=./gradlew clean
```

This defines the variable `clean_command` for all cars that reference this configuration. Rally will treat the following variable names specially:

- *clean_command*: The command to clean the Elasticsearch project directory.
- *build_command*: The command to build an Elasticsearch source distribution.
- *artifact_path_pattern*: A glob pattern to find a previously built source distribution within the project directory.
- *release_url*: A download URL for Elasticsearch distributions. The placeholder `{{VERSION}}` is replaced by Rally with the actual Elasticsearch version.

Let's have a look at the `1gheap` car by inspecting `1gheap.ini`:

```
[meta]
description=Sets the Java heap to 1GB
type=car

[config]
base=vanilla

[variables]
heap_size=1g
```

The name of the car is derived from the `.ini` file name. In the `meta` section we can provide a `description` and the `type`. Use `car` if a configuration can be used standalone and `mixin` if it needs to be combined with other configurations. In the `config` section we define that this definition is based on the `vanilla` configuration. We also define a variable `heap_size` and set it to `1g`. Note that variables defined here take precedence over variables defined in the `config.ini` file of any of the referenced configurations.

Let's open `vanilla/config/templates/jvm.options` to see how this variable is used (we'll only show the relevant part here):

```
# Xms represents the initial size of total heap space
# Xmx represents the maximum size of total heap space
```

(continues on next page)

(continued from previous page)

```
-Xms{{heap_size}}
-Xmx{{heap_size}}
```

So Rally reads all variables and the template files and replaces the variables in the final configuration. Note that Rally does not know anything about `jvm.options` or `elasticsearch.yml`. For Rally, these are just plain text templates that need to be copied to the Elasticsearch directory before running a benchmark. Under the hood, Rally uses Jinja2 as template language. This allows you to use Jinja2 expressions in your car configuration files.

If you open `vanilla/templates/config/elasticsearch.yml` you will see a few variables that are not defined in the `.ini` file:

- `network_host`
- `http_port`

These values are derived by Rally internally based on command line flags and you cannot override them in your car definition. You also cannot use these names as names for variables because Rally would simply override them.

If you specify multiple configurations, e.g. `--car="4gheap,ea"`, Rally will apply them in order. It will first read all variables in `4gheap.ini`, then in `ea.ini`. Afterwards, it will copy all configuration files from the corresponding config base of `4gheap` and *append* all configuration files from `ea`. This also shows when to define a separate "car" and when to define a "mixin": If you need to amend configuration files, use a mixin, if you need to have a specific configuration, define a car.

Simple customizations

For simple customizations you can create the directory hierarchy as outlined above and use the `--team-path` command line parameter to refer to this configuration. For more complex use cases and distributed multi-node benchmarks, we recommend to use custom team repositories.

Custom Team Repositories

Rally provides a default team repository that is hosted on Github. You can also add your own team repositories although this requires a bit of additional work. First of all, team repositories need to be managed by git. The reason is that Rally can benchmark multiple versions of Elasticsearch and we use git branches in the track repository to determine the best match. The versioning scheme is as follows:

- The *master* branch needs to work with the latest *master* branch of Elasticsearch.
- All other branches need to match the version scheme of Elasticsearch, i.e. `MAJOR.MINOR.PATCH-SUFFIX` where all parts except `MAJOR` are optional.

Rally implements a fallback logic so you don't need to define a branch for each patch release of Elasticsearch. For example:

- The branch *6.0.0-alpha1* will be chosen for the version `6.0.0-alpha1` of Elasticsearch.
- The branch *5* will be chosen for all versions for Elasticsearch with the major version 5, e.g. `5.0.0`, `5.1.3` (provided there is no specific branch).

Rally tries to use the branch with the best match to the benchmarked version of Elasticsearch.

Creating a new team repository

All team repositories are located in `~/.rally/benchmarks/teams`. If you want to add a dedicated team repository, called `private` follow these steps:

```
cd ~/.rally/benchmarks/teams
mkdir private
cd private
git init
# add your team now (don't forget to add the subdirectory "cars").
git add .
git commit -m "Initial commit"
```

If you want to share your teams with others (or you want to run remote benchmarks) you need to add a remote and push it:

```
git remote add origin git@git-repos.acme.com:acme/rally-teams.git
git push -u origin master
```

If you have added a remote you should also add it in `~/.rally/rally.ini`, otherwise you can skip this step. Open the file in your editor of choice and add the following line in the section `teams`:

```
private.url = <<URL_TO_YOUR_ORIGIN>>
```

Rally will then automatically update the local tracking branches before the benchmark starts.

Warning: If you run benchmarks against a remote machine that is under the control of Rally then you need to add the custom team configuration on every node!

You can now verify that everything works by listing all teams in this team repository:

```
esrally list cars --team-repository=private
```

This shows all teams that are available on the `master` branch of this repository. Suppose you only created tracks on the branch `2` because you're interested in the performance of Elasticsearch 2.x, then you can specify also the distribution version:

```
esrally list teams --team-repository=private --distribution-version=2.0.0
```

Rally will follow the same branch fallback logic as described above.

Adding an already existing team repository

If you want to add a team repository that already exists, just open `~/.rally/rally.ini` in your editor of choice and add the following line in the section `teams`:

```
your_repo_name.url = <<URL_TO_YOUR_ORIGIN>>
```

After you have added this line, have Rally list the tracks in this repository:

```
esrally list cars --team-repository=your_repo_name
```

2.16 Using Elasticsearch Plugins

You can have Rally setup an Elasticsearch cluster with plugins for you. However, there are a couple of restrictions:

- This feature is only supported from Elasticsearch 5.0.0 onwards
- Whereas Rally caches downloaded Elasticsearch distributions, plugins will always be installed via the Internet and thus each machine where an Elasticsearch node will be installed, requires an active Internet connection.

2.16.1 Listing plugins

To see which plugins are available, run `esrally list elasticsearch-plugins`:

```
Available Elasticsearch plugins:

Name                     Configuration
-----------------------  ----------------
analysis-icu
analysis-kuromoji
analysis-phonetic
analysis-smartcn
analysis-stempel
analysis-ukrainian
discovery-azure-classic
discovery-ec2
discovery-file
discovery-gce
ingest-attachment
ingest-geoip
ingest-user-agent
lang-javascript
lang-python
mapper-attachments
mapper-murmur3
mapper-size
repository-azure
repository-gcs
repository-hdfs
repository-s3
store-smb
```

Rally supports plugins only for Elasticsearch 5.0 or better. As the availability of plugins may change from release to release we recommend that you include the `--distribution-version` parameter when listing plugins. By default Rally assumes that you want to benchmark the latest master version of Elasticsearch.

Let's see what happens if we run `esrally list elasticsearch-plugins --distribution-version=2.4.0`:

```
No Elasticsearch plugins are available.
```

As mentioned before, this is expected as only Elasticsearch 5.0 or better is supported.

2.16.2 Running a benchmark with plugins

In order to tell Rally to install a plugin, use the `--elasticsearch-plugins` parameter when starting a race. You can provide multiple plugins (comma-separated) and they will be installed in the order to that you define on the command line.

Example:

```
esrally --distribution-version=5.5.0 --elasticsearch-plugins="analysis-icu,analysis-
↪phonetic"
```

This will install the plugins `analysis-icu` and `analysis-phonetic` (in that order). In order to use the features that these plugins provide, you need to write a *custom track*.

Rally will use several techniques to install and configure plugins:

- First, Rally checks whether directory `plugins/PLUGIN_NAME` in the currently configured team repository exists. If this is the case, then plugin installation and configuration details will be read from this directory.
- Next, Rally will use the provided plugin name when running the Elasticsearch plugin installer. With this approach we can avoid to create a plugin configuration directory in the team repository for very simple plugins that do not need any configuration.

As mentioned above, Rally also allows you to specify a plugin configuration and you can even combine them. Here are some examples (requires Elasticsearch < 6.3.0 because with 6.3.0 x-pack has turned into a module of Elasticsearch which is treated as a "car" in Rally):

- Run a benchmark with the `x-pack` plugin in the `security` configuration: `--elasticsearch-plugins=x-pack:security`
- Run a benchmark with the `x-pack` plugin in the `security` and the `graph` configuration: `--elasticsearch-plugins=x-pack:security+graph`

Note: To benchmark the `security` configuration of `x-pack` you need to add the following command line options: `--client-options="use_ssl:true,verify_certs:false,basic_auth_user:'rally', basic_auth_password:'rally-password'"`

You can also override plugin variables with `--plugin-params` which is needed for example if you want to use the `monitoring-http` configuration in order to export monitoring data. You can export monitoring data e.g. with the following configuration:

```
--elasticsearch-plugins="x-pack:monitoring-http" --plugin-params="monitoring_type:
↪'https',monitoring_host:'some_remote_host',monitoring_port:10200,monitoring_user:
↪'rally',monitoring_password:'m0n1t0r1ng'"
```

The `monitoring_user` and `monitoring_password` parameters are optional, the other parameters are mandatory. For more details on the configuration options check the Monitoring plugin documentation.

If you are behind a proxy, set the environment variable `ES_JAVA_OPTS` accordingly on each target machine as described in the Elasticsearch plugin documentation.

2.16.3 Building plugins from sources

Plugin authors may want to benchmark source builds of their plugins. Your plugin is either:

- built alongside Elasticsearch
- built against a released version of Elasticsearch

Plugins built alongside Elasticsearch

To make this work, you need to manually edit Rally's configuration file in `~/.rally/rally.ini`. Suppose, we want to benchmark the plugin "my-plugin". Then you need to add the following entries in the `source` section:

```
plugin.my-plugin.remote.repo.url = git@github.com:example-org/my-plugin.git
plugin.my-plugin.src.subdir = elasticsearch-extra/my-plugin
plugin.my-plugin.build.command = ./gradlew :my-plugin:plugin:assemble
plugin.my-plugin.build.artifact.subdir = plugin/build/distributions
```

Let's discuss these properties one by one:

- `plugin.my-plugin.remote.repo.url` (optional): This is needed to let Rally checkout the source code of the plugin. If this is a private repo, credentials need to be setup properly. If the source code is already locally available you may not need to define this property. The remote's name is assumed to be "origin" and this is not configurable. Also, only git is supported as revision control system.
- `plugin.my-plugin.src.subdir` (mandatory): This is the directory to which the plugin will be checked out relative to `src.root.dir`. In order to allow to build the plugin alongside Elasticsearch, the plugin needs to reside in a subdirectory of `elasticsearch-extra` (see also the Elasticsearch testing documentation.
- `plugin.my-plugin.build.command` (mandatory): The full build command to run in order to build the plugin artifact. Note that this command is run from the Elasticsearch source directory as Rally assumes that you want to build your plugin alongside Elasticsearch (otherwise, see the next section).
- `plugin.my-plugin.build.artifact.subdir` (mandatory): This is the subdirectory relative to `plugin.my-plugin.src.subdir` in which the final plugin artifact is located.

Warning: `plugin.my-plugin.build.command` has replaced `plugin.my-plugin.build.task` in earlier Rally versions. It now requires the **full** build command.

In order to run a benchmark with `my-plugin`, you'd invoke Rally as follows: `esrally --revision="elasticsearch:some-elasticsearch-revision, my-plugin:some-plugin-revision" --elasticsearch-plugins="my-plugin"` where you need to replace `some-elasticsearch-revision` and `some-plugin-revision` with the appropriate *git revisions*. Adjust other command line parameters (like track or car) accordingly. In order for this to work, you need to ensure that:

- All prerequisites for source builds are installed.
- The Elasticsearch source revision is compatible with the chosen plugin revision. Note that you do not need to know the revision hash to build against an already released version and can use git tags instead. E.g. if you want to benchmark against Elasticsearch 5.6.1, you can specify `--revision="elasticsearch:v5.6.1,my-plugin:some-plugin-revision"` (see e.g. the Elasticsearch tags on Github or use `git tag` in the Elasticsearch source directory on the console).
- If your plugin needs to be configured, create a proper plugin specification (see below).

Note: Rally can build all Elasticsearch core plugins out of the box without any further configuration.

Plugins based on a released Elasticsearch version

To make this work, you need to manually edit Rally's configuration file in `~/.rally/rally.ini`. Suppose, we want to benchmark the plugin "my-plugin". Then you need to add the following entries in the `source` section:

```
plugin.my-plugin.remote.repo.url = git@github.com:example-org/my-plugin.git
plugin.my-plugin.src.dir = /path/to/your/plugin/sources
plugin.my-plugin.build.command = /usr/local/bin/gradle :my-plugin:plugin:assemble
plugin.my-plugin.build.artifact.subdir = build/distributions
```

Let's discuss these properties one by one:

- `plugin.my-plugin.remote.repo.url` (optional): This is needed to let Rally checkout the source code of the plugin. If this is a private repo, credentials need to be setup properly. If the source code is already locally available you may not need to define this property. The remote's name is assumed to be "origin" and this is not configurable. Also, only git is supported as revision control system.
- `plugin.my-plugin.src.dir` (mandatory): This is the absolute directory to which the source code will be checked out.
- `plugin.my-plugin.build.command` (mandatory): The full build command to run in order to build the plugin artifact. This command is run from the plugin project's root directory.
- `plugin.my-plugin.build.artifact.subdir` (mandatory): This is the subdirectory relative to `plugin.my-plugin.src.dir` in which the final plugin artifact is located.

Warning: `plugin.my-plugin.build.command` has replaced `plugin.my-plugin.build.task` in earlier Rally versions. It now requires the **full** build command.

In order to run a benchmark with `my-plugin`, you'd invoke Rally as follows: `esrally --distribution-version="elasticsearch-version" --revision="my-plugin:some-plugin-revision" --elasticsearch-plugins="my-plugin"` where you need to replace `elasticsearch-version` with the correct release (e.g. 6.0.0) and `some-plugin-revision` with the appropriate *git revisions*. Adjust other command line parameters (like track or car) accordingly. In order for this to work, you need to ensure that:

- All prerequisites for source builds are installed.
- The Elasticsearch release is compatible with the chosen plugin revision.
- If your plugin needs to be configured, create a proper plugin specification (see below).

2.16.4 Anatomy of a plugin specification

Simple plugins

You can use Rally to benchmark community-contributed or even your own plugins. In the simplest case, the plugin does not need any custom configuration. Then you just need to add the download URL to your Rally configuration file. Consider we want to benchmark the plugin "my-plugin":

```
[distributions]
plugin.my-plugin.release.url=https://example.org/my-plugin/releases/{{VERSION}}/my-
↪plugin-{{VERSION}}.zip
```

Then you can use `--elasticsearch-plugins=my-plugin` to run a benchmark with your plugin. Rally will also replace `{{VERSION}}` with the distribution version that you have specified on the command line.

Plugins which require configuration

If the plugin needs a custom configuration we recommend to fork the official Rally teams repository and add your plugin configuration there. Suppose, you want to benchmark "my-plugin" which has the following settings that can be configured in `elasticsearch.yml`:

- `myplugin.active`: a boolean which activates the plugin
- `myplugin.mode`: Either `simple` or `advanced`

We want to support two configurations for this plugin: `simple` which will set `myplugin.mode` to `simple` and `advanced` which will set `myplugin.mode` to `advanced`.

First, we need a template configuration. We will call this a "config base" in Rally. We will just need one config base for this example and will call it "default".

In `$TEAM_REPO_ROOT` create the directory structure for the plugin and its config base with *mkdir -p myplugin/default/templates/config* and add the following `elasticsearch.yml` in the new directory:

```
myplugin.active: true
myplugin.mode={{my_plugin_mode}}
```

That's it. Later, Rally will just copy all files in `myplugin/default/templates` to the home directory of the Elasticsearch node that it configures. First, Rally will always apply the car's configuration and then plugins can add their configuration on top. This also explains why we have created a `config/elasticsearch.yml`. Rally will just copy this file and replace template variables on the way.

Note: If you create a new customization for a plugin, ensure that the plugin name in the team repository matches the core plugin name. Note that hyphens need to be replaced by underscores (e.g. "x-pack" becomes "x_pack"). The reason is that Rally allows to write custom install hooks and the plugin name will become the root package name of the install hook. However, hyphens are not supported in Python which is why we use underscores instead.

The next step is now to create our two plugin configurations where we will set the variables for our config base "default". Create a file `simple.ini` in the `myplugin` directory:

```
[config]
# reference our one and only config base here
base=default

[variables]
my_plugin_mode=simple
```

Similarly, create `advanced.ini` in the `myplugin` directory:

```
[config]
# reference our one and only config base here
base=default

[variables]
my_plugin_mode=advanced
```

Rally will now know about `myplugin` and its two configurations. Let's check that with `esrally list elasticsearch-plugins`:

```
Available Elasticsearch plugins:

Name                         Configuration
```

(continues on next page)

(continued from previous page)

```
-----------------------  ----------------
analysis-icu
analysis-kuromoji
analysis-phonetic
analysis-smartcn
analysis-stempel
analysis-ukrainian
discovery-azure-classic
discovery-ec2
discovery-file
discovery-gce
ingest-attachment
ingest-geoip
ingest-user-agent
lang-javascript
lang-python
mapper-attachments
mapper-murmur3
mapper-size
myplugin                 simple
myplugin                 advanced
repository-azure
repository-gcs
repository-hdfs
repository-s3
store-smb
```

As `myplugin` is not a core plugin, the Elasticsearch plugin manager does not know from where to install it, so we need to add the download URL to `~/.rally/rally.ini` as before:

```
[distributions]
plugin.myplugin.release.url=https://example.org/myplugin/releases/{{VERSION}}/
↪myplugin-{{VERSION}}.zip
```

Now you can run benchmarks with the custom Elasticsearch plugin, e.g. with `esrally --distribution-version=5.5.0 --elasticsearch-plugins="myplugin:simple"`.

For this to work you need ensure two things:

1. The plugin needs to be available for the version that you want to benchmark (5.5.0 in the example above).
2. Rally will choose the most appropriate branch in the team repository before starting the benchmark. In practice, this will most likely be branch "5" for this example. Therefore you need to ensure that your plugin configuration is also available on that branch. See the README in the team repository to learn how the versioning scheme works.

2.17 Telemetry Devices

You probably want to gain additional insights from a race. Therefore, we have added telemetry devices to Rally. If you invoke `esrally list telemetry`, it will show which telemetry devices are available:

```
dm@io:Projects/rally ‹master*›$ esrally list telemetry


    ____        ____
   / __ \____ _/ / /_  __
```

(continues on next page)

(continued from previous page)

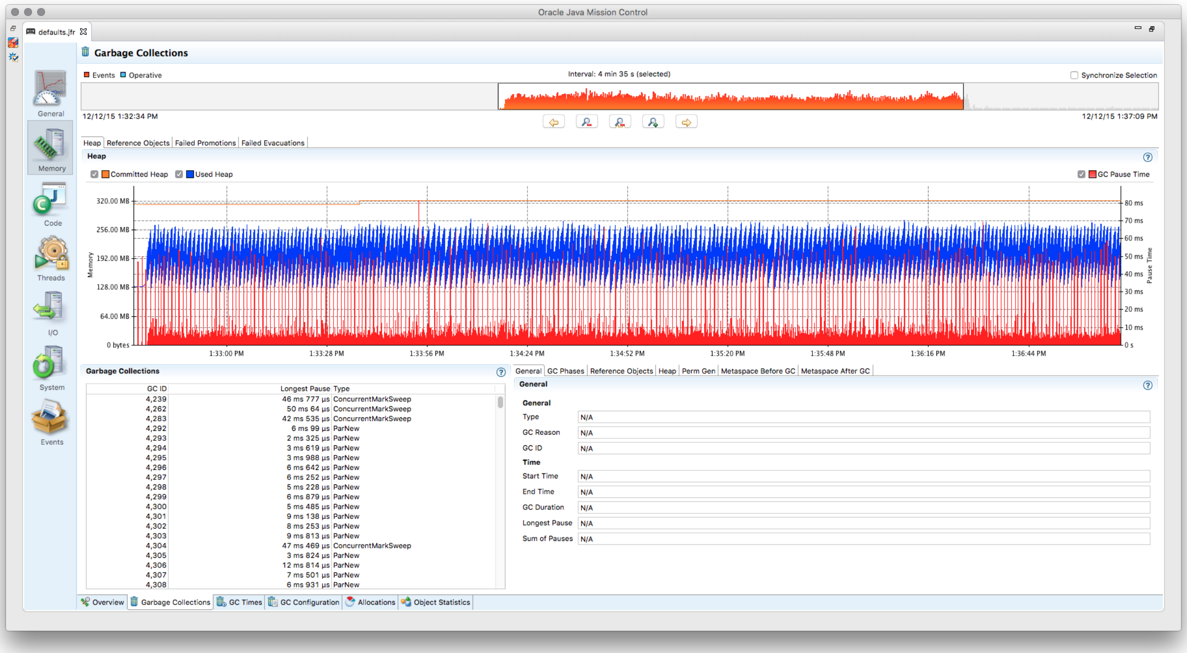
```
  / /_/ / __ `/ / / / / /
 / _, _/ /_/ / / / /_/ /
/_/ |_|\__,_/_/_/\__, /
                /____/


Available telemetry devices:

Command         Name                    Description
--------------  ----------------------  ----------------------------------------------
↪---------------------
jit              JIT Compiler Profiler  Enables JIT compiler logs.
gc               GC log                 Enables GC logs.
jfr              Flight Recorder        Enables Java Flight Recorder (requires an␣
↪Oracle JDK or OpenJDK 11+)
heapdump         Heap Dump              Captures a heap dump.
node-stats       Node Stats             Regularly samples node stats
recovery-stats   Recovery Stats         Regularly samples shard recovery stats
ccr-stats        CCR Stats              Regularly samples Cross Cluster Replication␣
↪(CCR) related stats
transform-stats  Transform Stats        Regularly samples transform stats

Keep in mind that each telemetry device may incur a runtime overhead which can skew␣
↪results.
```

You can attach one or more of these telemetry devices to the benchmarked cluster. Except for `node-stats`, this only works if Rally provisions the cluster (i.e. it does not work with `--pipeline=benchmark-only`).

2.17.1 jfr

The `jfr` telemetry device enables the Java Flight Recorder on the benchmark candidate. Up to JDK 11, Java flight recorder ships only with Oracle JDK, so Rally assumes that Oracle JDK is used for benchmarking. If you run benchmarks on JDK 11 or later, Java flight recorder is also available on OpenJDK.

To enable `jfr`, invoke Rally with `esrally --telemetry jfr`. `jfr` will then write a flight recording file which can be opened in Java Mission Control. Rally prints the location of the flight recording file on the command line.



Garbage Collections

GC ID	Longest Pause	Type
4,239	46 ms 777 µs	ConcurrentMarkSweep
4,262	50 ms 64 µs	ConcurrentMarkSweep
4,283	42 ms 535 µs	ConcurrentMarkSweep
4,292	6 ms 99 µs	ParNew
4,293	2 ms 325 µs	ParNew
4,294	3 ms 619 µs	ParNew
4,295	3 ms 988 µs	ParNew
4,296	6 ms 642 µs	ParNew
4,297	6 ms 252 µs	ParNew
4,298	5 ms 228 µs	ParNew
4,299	6 ms 879 µs	ParNew
4,300	5 ms 485 µs	ParNew
4,301	9 ms 138 µs	ParNew
4,302	8 ms 253 µs	ParNew
4,303	9 ms 813 µs	ParNew
4,304	47 ms 469 µs	ConcurrentMarkSweep
4,305	3 ms 824 µs	ParNew
4,306	12 ms 814 µs	ParNew
4,307	7 ms 501 µs	ParNew
4,308	6 ms 931 µs	ParNew

General

General

Type	N/A
GC Reason	N/A
GC ID	N/A

Time

Start Time	N/A
End Time	N/A
GC Duration	N/A
Longest Pause	N/A
Sum of Pauses	N/A

Supported telemetry parameters:

- `recording-template`: The name of a custom flight recording template. It is up to you to correctly install these recording templates on each target machine. If none is specified, the default recording template of Java flight recorder is used.

Note: Up to JDK 11 Java flight recorder ship only with Oracle JDK and the licensing terms do not allow you to run it in production environments without a valid license (for details, refer to the Oracle Java SE Advanced & Suite Products page). However, running in a QA environment is fine.

2.17.2 jit

The `jit` telemetry device enables JIT compiler logs for the benchmark candidate. If the HotSpot disassembler library is available, the logs will also contain the disassembled JIT compiler output which can be used for low-level analysis. We recommend to use JITWatch for analysis.

`hsdis` can be built for JDK 8 on Linux with (based on a description by Alex Blewitt):

```
curl -O -O -O -O https://raw.githubusercontent.com/dmlloyd/openjdk/jdk8u/jdk8u/
↪hotspot/src/share/tools/hsdis/{hsdis.c,hsdis.h,Makefile,README}
mkdir -p build/binutils
curl http://ftp.gnu.org/gnu/binutils/binutils-2.27.tar.gz | tar --strip-components=1 -
↪C build/binutils -z -x -f -
make BINUTILS=build/binutils ARCH=amd64
```

After it has been built, the binary needs to be copied to the JDK directory (see `README` of hsdis for details).

2.17.3 gc

The `gc` telemetry device enables GC logs for the benchmark candidate. You can use tools like GCViewer to analyze the GC logs.

If the runtime JDK is Java 9 or higher, the following telemetry parameters can be specified:

- `gc-log-config` (default: `gc*=info,safepoint=info,age*=trace`): The GC logging configuration consisting of a list of tags and levels. Run `java -Xlog:help` to see the list of available levels and tags.

Note: Use a JSON file for `telemetry-params` as the simple parameter format is not supported for the GC log configuration string. See the *command line reference* for details.

2.17.4 heapdump

The `heapdump` telemetry device will capture a heap dump after a benchmark has finished and right before the node is shutdown.

2.17.5 node-stats

Warning: With `Elasticsearch < 7.2.0`, using this telemetry device will skew your results because the node-stats API triggers additional refreshes. Additionally a lot of metrics get recorded impacting the measurement results even further.

The node-stats telemetry device regularly calls the cluster node-stats API and records metrics from the following sections:

- Indices stats (key `indices` in the node-stats API)
- Thread pool stats (key `thread_pool` in the node-stats API)
- JVM buffer pool stats (key `jvm.buffer_pools` in the node-stats API)
- JVM gc stats (key `jvm.gc` in the node-stats API)
- JVM mem stats (key `jvm.mem` in the node-stats API)
- Circuit breaker stats (key `breakers` in the node-stats API)
- Network-related stats (key `transport` in the node-stats API)
- Process cpu stats (key `process.cpu` in the node-stats API)

Supported telemetry parameters:

- `node-stats-sample-interval` (default: 1): A positive number greater than zero denoting the sampling interval in seconds.
- `node-stats-include-indices` (default: `false`): A boolean indicating whether indices stats should be included.
- `node-stats-include-indices-metrics` (default: `docs,store,indexing,search,merges,query_cache,fielddata,segments,translog,request_cache`): A comma-separated string specifying the Indices stats metrics to include. This is useful, for example, to restrict the

collected Indices stats metrics. Specifying this parameter implicitly enables collection of Indices stats, so you don't also need to specify `node-stats-include-indices: true`.

Example: `--telemetry-params="node-stats-include-indices-metrics:'docs'"` will **only** collect the `docs` metrics from Indices stats. If you want to use multiple fields, pass a JSON file to `telemetry-params` (see the *command line reference* for details).

- `node-stats-include-thread-pools` (default: `true`): A boolean indicating whether thread pool stats should be included.
- `node-stats-include-buffer-pools` (default: `true`): A boolean indicating whether buffer pool stats should be included.
- `node-stats-include-breakers` (default: `true`): A boolean indicating whether circuit breaker stats should be included.
- `node-stats-include-gc` (default: `true`): A boolean indicating whether JVM gc stats should be included.
- `node-stats-include-mem` (default: `true`): A boolean indicating whether JVM heap stats should be included.
- `node-stats-include-network` (default: `true`): A boolean indicating whether network-related stats should be included.
- `node-stats-include-process` (default: `true`): A boolean indicating whether process cpu stats should be included.
- `node-stats-include-indexing-pressure` (default: `true`): A boolean indicating whether indexing pressuer stats should be included.

2.17.6 recovery-stats

The recovery-stats telemetry device regularly calls the indices recovery API and records one metrics document per shard.

Supported telemetry parameters:

- `recovery-stats-indices` (default: all indices): An index pattern for which recovery stats should be checked.
- `recovery-stats-sample-interval` (default 1): A positive number greater than zero denoting the sampling interval in seconds.

2.17.7 ccr-stats

The ccr-stats telemetry device regularly calls the cross-cluster replication stats API and records one metrics document per shard.

Supported telemetry parameters:

- `ccr-stats-indices` (default: all indices): An index pattern for which ccr stats should be checked.
- `ccr-stats-sample-interval` (default 1): A positive number greater than zero denoting the sampling interval in seconds.

2.17.8 transform-stats

The transform-stats telemetry device regularly calls the *transform stats API <https://www.elastic.co/guide/en/elasticsearch/reference/current/get-transform-stats.html>* and records one metrics document per transform.

Supported telemetry parameters:

- `transform-stats-transforms` (default: all transforms): A list of transforms per cluster for which transform stats should be checked.
- `transform-stats-sample-interval` (default 1): A positive number greater than zero denoting the sampling interval in seconds.

2.18 Rally Daemon

At its heart, Rally is a distributed system, just like Elasticsearch. However, in its simplest form you will not notice, because all components of Rally can run on a single node too. If you want Rally to *configure and start Elasticsearch nodes remotely* or *distribute the load test driver* to apply load from multiple machines, you need to use Rally daemon.

Rally daemon needs to run on every machine that should be under Rally's control. We can consider three different roles:

- Benchmark coordinator: This is the machine where you invoke `esrally`. It is responsible for user interaction, coordinates the whole benchmark and shows the results. Only one node can be the benchmark coordinator.
- Load driver: Nodes of this type will interpret and run *tracks*.
- Provisioner: Nodes of this type will configure an Elasticsearch cluster according to the provided *car* and *Elasticsearch plugin* configurations.

The two latter roles are not statically preassigned but rather determined by Rally based on the command line parameter `--load-driver-hosts` (for the load driver) and `--target-hosts` (for the provisioner).

2.18.1 Preparation

First, *install* and *configure* Rally on all machines that are involved in the benchmark. If you want to automate this, there is no need to use the interactive configuration routine of Rally. You can copy *~/.rally/rally.ini* to the target machines adapting the paths in the file as necessary. We also recommend that you copy `~/.rally/benchmarks/data` to all load driver machines before-hand. Otherwise, each load driver machine will need to download a complete copy of the benchmark data.

Note: Rally Daemon will listen on port 1900 and the actor system that Rally uses internally require access to arbitrary (unprivileged) ports. Be sure to open up these ports between the Rally nodes.

2.18.2 Starting

For all this to work, Rally needs to form a cluster. This is achieved with the binary `esrallyd` (note the "d" - for daemon - at the end). You need to start the Rally daemon on all nodes: First on the coordinator node, then on all others. The order does matter, because nodes attempt to connect to the coordinator on startup.

On the benchmark coordinator, issue:

```
esrallyd start --node-ip=IP_OF_COORDINATOR_NODE --coordinator-ip=IP_OF_COORDINATOR_
↪NODE
```

On all other nodes, issue:

```
esrallyd start --node-ip=IP_OF_THIS_NODE --coordinator-ip=IP_OF_COORDINATOR_NODE
```

After that, all Rally nodes, know about each other and you can use Rally as usual. See the *tips and tricks* for more examples.

2.18.3 Stopping

You can leave the Rally daemon processes running in case you want to run multiple benchmarks. When you are done, you can stop the Rally daemon on each node with:

```
esrallyd stop
```

Contrary to startup, order does not matter here.

2.18.4 Status

You can query the status of the local Rally daemon with:

```
esrallyd status
```

2.18.5 Troubleshooting

Rally uses the actor system Thespian under the hood.

At startup, Thespian attempts to detect an appropriate IP address. If Rally fails to startup the actor system indicated by the following message:

```
thespian.actors.InvalidActorAddress: ActorAddr-(T|:1900) is not a valid ActorSystem␣
↪admin
```

then set a routable IP address yourself by setting the environment variable `THESPIAN_BASE_IPADDR` before starting Rally.

Note: This issue often occurs when Rally is started on a machine that is connected via a VPN to the Internet. We advise against such a setup for benchmarking and suggest to setup the load generator and the target machines close to each other, ideally in the same subnet.

To inspect Thespian's status in more detail you can use the Thespian shell. Below is an example invocation that demonstrates how to retrieve the actor system status:

```
python3 -m thespian.shell
Thespian Actor shell.  Type help or '?' to list commands.'

thespian> start multiprocTCPBase
Starting multiprocTCPBase ActorSystem
Capabilities: {}
```

(continues on next page)

(continued from previous page)

```
Started multiprocTCPBase ActorSystem
thespian> address localhost 1900
Args is: {'port': '1900', 'ipaddr': 'localhost'}
Actor Address 0:  ActorAddr-(T|:1900)
thespian> status
Requesting status from Actor (or Admin) @ ActorAddr-(T|:1900) (#0)
Status of ActorSystem @ ActorAddr-(T|192.168.14.2:1900) [#1]:
  |Capabilities[9]:
                              ip: 192.168.14.2
          Convention Address.IPv4: 192.168.14.2:1900
              Thespian Generation: (3, 9)
         Thespian Watch Supported: True
                   Python Version: (3, 5, 2, 'final', 0)
        Thespian ActorSystem Name: multiprocTCPBase
     Thespian ActorSystem Version: 2
                 Thespian Version: 1581669778176
                      coordinator: True
  |Convention Leader: ActorAddr-(T|192.168.14.2:1900) [#1]
  |Convention Attendees [3]:
    @ ActorAddr-(T|192.168.14.4:1900) [#2]: Expires_in_0:21:41.056599
    @ ActorAddr-(T|192.168.14.3:1900) [#3]: Expires_in_0:21:41.030934
    @ ActorAddr-(T|192.168.14.5:1900) [#4]: Expires_in_0:21:41.391251
  |Primary Actors [0]:
  |Rate Governer: Rate limit: 4480 messages/sec (currently low with 1077 ticks)
  |Pending Messages [0]:
  |Received Messages [0]:
  |Pending Wakeups [0]:
  |Pending Address Resolution [0]:
  |>        1077 - Actor.Message Send.Transmit Started
  |>          84 - Admin Handle Convention Registration
  |>        1079 - Admin Message Received.Total
  |>           6 - Admin Message Received.Type.QueryExists
  |>         988 - Admin Message Received.Type.StatusReq
  |> sock#0-fd10 - Idle-socket <socket.socket fd=10, family=AddressFamily.AF_INET,␣
↪type=2049, proto=6, laddr=('192.168.14.2', 1900), raddr=('192.168.14.4', 44024)>->
↪ActorAddr-(T|192.168.14.4:1900) (Expires_in_0:19:35.060480)
  |> sock#2-fd11 - Idle-socket <socket.socket fd=11, family=AddressFamily.AF_INET,␣
↪type=2049, proto=6, laddr=('192.168.14.2', 1900), raddr=('192.168.14.3', 40244)>->
↪ActorAddr-(T|192.168.14.3:1900) (Expires_in_0:19:35.034779)
  |> sock#3-fd12 - Idle-socket <socket.socket fd=12, family=AddressFamily.AF_INET,␣
↪type=2049, proto=6, laddr=('192.168.14.2', 1900), raddr=('192.168.14.5', 58358)>->
↪ActorAddr-(T|192.168.14.5:1900) (Expires_in_0:19:35.394918)
  |> sock#1-fd13 - Idle-socket <socket.socket fd=13, family=AddressFamily.AF_INET,␣
↪type=2049, proto=6, laddr=('127.0.0.1', 1900), raddr=('127.0.0.1', 34320)>->
↪ActorAddr-(T|:46419) (Expires_in_0:19:59.999337)
  |DeadLetter Addresses [0]:
  |Source Authority: None
  |Loaded Sources [0]:
  |Global Actors [0]:
```

2.19 Pipelines

A pipeline is a series of steps that are performed to get benchmark results. This is *not* intended to customize the actual benchmark but rather what happens before and after a benchmark.

An example will clarify the concept: If you want to benchmark a binary distribution of Elasticsearch, Rally has to download a distribution archive, decompress it, start Elasticsearch and then run the benchmark. However, if you want to benchmark a source build of Elasticsearch, it first has to build a distribution using the Gradle Wrapper. So, in both cases, different steps are involved and that's what pipelines are for.

You can get a list of all pipelines with `esrally list pipelines`:

```
Available pipelines:

Name                      Description
-----------------------  -------------------------------------------------------------
↪--------------------------------
from-sources              Builds and provisions Elasticsearch, runs a benchmark and␣
↪reports results.
from-sources-complete     Builds and provisions Elasticsearch, runs a benchmark and␣
↪reports results [deprecated].
from-sources-skip-build   Provisions Elasticsearch (skips the build), runs a benchmark␣
↪and reports results [deprecated].
from-distribution         Downloads an Elasticsearch distribution, provisions it, runs␣
↪a benchmark and reports results.
benchmark-only            Assumes an already running Elasticsearch instance, runs a␣
↪benchmark and reports results
```

2.19.1 benchmark-only

This is intended if you want to provision a cluster by yourself. Do not use this pipeline unless you are absolutely sure you need to. As Rally has not provisioned the cluster, results are not easily reproducable and it also cannot gather a lot of metrics (like CPU usage).

To benchmark a cluster, you also have to specify the hosts to connect to. An example invocation:

```
esrally --pipeline=benchmark-only --target-hosts=search-node-a.intranet.acme.com:9200,
↪search-node-b.intranet.acme.com:9200
```

2.19.2 from-distribution

This pipeline allows to benchmark an official Elasticsearch distribution which will be automatically downloaded by Rally. The earliest supported version is Elasticsearch 1.7.0. An example invocation:

```
esrally --pipeline=from-distribution --distribution-version=1.7.5
```

The version numbers have to match the name in the download URL path.

You can also benchmark Elasticsearch snapshot versions by specifying the snapshot repository:

```
esrally --pipeline=from-distribution --distribution-version=5.0.0-SNAPSHOT --
↪distribution-repository=snapshot
```

However, this feature is mainly intended for continuous integration environments and by default you should just benchmark official distributions.

Note: This pipeline is just mentioned for completeness but Rally will autoselect it for you. All you need to do is to define the `--distribution-version` flag.

2.19.3 from-sources

You should use this pipeline when you want to build and benchmark Elasticsearch from sources. This pipeline will only work from Elasticsearch 5.0 onwards because Elasticsearch switched from Maven to Gradle and Rally only supports one build tool in the interest of maintainability.

Remember that you also need git installed. If that's not the case you'll get an error and have to run `esrally configure` first. An example invocation:

```
esrally --pipeline=from-sources --revision=latest
```

You have to specify a *revision*.

Note: This pipeline is just mentioned for completeness but Rally will automatically select it for you. All you need to do is to define the `--revision` flag.

Artifacts are cached for seven days by default in `~/.rally/benchmarks/distributions/src`. Artifact caching can be configured with the following sections in the section `source` in the configuration file in `~/.rally/rally.ini`:

- `cache` (default: `True`): Set to `False` to disable artifact caching.
- `cache.days` (default: `7`): The maximum age in days of an artifact before it gets evicted from the artifact cache.

2.19.4 from-sources-complete

This deprecated pipeline is an alias for `from-sources` and is only provided for backwards-compatibility. Use `from-sources` instead.

2.19.5 from-sources-skip-build

This deprecated pipeline is similar to `from-sources-complete` except that it assumes you have built the binary once. Use `from-sources` instead which caches built artifacts automatically.

2.20 Metrics

2.20.1 Metrics Records

At the end of a race, Rally stores all metrics records in its metrics store, which is a dedicated Elasticsearch cluster. Rally stores the metrics in the indices `rally-metrics-*`. It will create a new index for each month.

Here is a typical metrics record:

```
{
      "environment": "nightly",
      "race-timestamp": "20160421T042749Z",
      "race-id": "6ebc6e53-ee20-4b0c-99b4-09697987e9f4",
      "@timestamp": 1461213093093,
      "relative-time": 10507328,
      "track": "geonames",
```

(continues on next page)

(continued from previous page)

```
    "track-params": {
      "shard-count": 3
    },
    "challenge": "append-no-conflicts",
    "car": "defaults",
    "sample-type": "normal",
    "name": "throughput",
    "value": 27385,
    "unit": "docs/s",
    "task": "index-append-no-conflicts",
    "operation": "index-append-no-conflicts",
    "operation-type": "Index",
    "meta": {
      "cpu_physical_cores": 36,
      "cpu_logical_cores": 72,
      "cpu_model": "Intel(R) Xeon(R) CPU E5-2699 v3 @ 2.30GHz",
      "os_name": "Linux",
      "os_version": "3.19.0-21-generic",
      "host_name": "beast2",
      "node_name": "rally-node0",
      "source_revision": "a6c0a81",
      "distribution_version": "5.0.0-SNAPSHOT",
      "tag_reference": "Github ticket 1234"
    }
  }
```

As you can see, we do not only store the metrics name and its value but lots of meta-information. This allows you to create different visualizations and reports in Kibana.

Below we describe each field in more detail.

environment

The environment describes the origin of a metric record. You define this value in the initial configuration of Rally. The intention is to clearly separate different benchmarking environments but still allow to store them in the same index.

track, track-params, challenge, car

This is the track, challenge and car for which the metrics record has been produced. If the user has provided track parameters with the command line parameter, `--track-params`, each of them is listed here too.

If you specify a car with mixins, it will be stored as one string separated with "+", e.g. `--car="4gheap,ea"` will be stored as `4gheap+ea` in the metrics store in order to simplify querying in Kibana. Check the *cars* documentation for more details.

sample-type

Rally can be configured to run for a certain period in warmup mode. In this mode samples will be collected with the `sample-type` "warmup" but only "normal" samples are considered for the results that reported.

race-timestamp

A constant timestamp (always in UTC) that is determined when Rally is invoked.

race-id

A UUID that changes on every invocation of Rally. It is intended to group all samples of a benchmarking run.

@timestamp

The timestamp in milliseconds since epoch determined when the sample was taken.

relative-time

The relative time in microseconds since the start of the benchmark. This is useful for comparing time-series graphs over multiple races, e.g. you might want to compare the indexing throughput over time across multiple races. Obviously, they should always start at the same (relative) point in time and absolute timestamps are useless for that.

name, value, unit

This is the actual metric name and value with an optional unit (counter metrics don't have a unit). Depending on the nature of a metric, it is either sampled periodically by Rally, e.g. the CPU utilization or query latency or just measured once like the final size of the index.

task, operation, operation-type

`task` is the name of the task (as specified in the track file) that ran when this metric has been gathered. Most of the time, this value will be identical to the operation's name but if the same operation is ran multiple times, the task name will be unique whereas the operation may occur multiple times. It will only be set for metrics with name `latency` and `throughput`.

`operation` is the name of the operation (as specified in the track file) that ran when this metric has been gathered. It will only be set for metrics with name `latency` and `throughput`.

`operation-type` is the more abstract type of an operation. During a race, multiple queries may be issued which are different `` operation``s but they all have the same ``operation-type `` (Search). For some metrics, only the operation type matters, e.g. it does not make any sense to attribute the CPU usage to an individual query but instead attribute it just to the operation type.

meta

Rally captures also some meta information for each metric record:

- CPU info: number of physical and logical cores and also the model name
- OS info: OS name and version
- Host name
- Node name: If Rally provisions the cluster, it will choose a unique name for each node.
- Source revision: We always record the git hash of the version of Elasticsearch that is benchmarked. This is even done if you benchmark an official binary release.
- Distribution version: We always record the distribution version of Elasticsearch that is benchmarked. This is even done if you benchmark a source release.
- Custom tag: You can define one custom tag with the command line flag `--user-tag`. The tag is prefixed by `tag_` in order to avoid accidental clashes with Rally internal tags.

- Operation-specific: The optional substructure `operation` contains additional information depending on the type of operation. For bulk requests, this may be the number of documents or for searches the number of hits.

Note that depending on the "level" of a metric record, certain meta information might be missing. It makes no sense to record host level meta info for a cluster wide metric record, like a query latency (as it cannot be attributed to a single node).

2.20.2 Metric Keys

Rally stores the following metrics:

- `latency`: Time period between submission of a request and receiving the complete response. It also includes wait time, i.e. the time the request spends waiting until it is ready to be serviced by Elasticsearch.
- `service_time` Time period between start of request processing and receiving the complete response. This metric can easily be mixed up with `latency` but does not include waiting time. This is what most load testing tools refer to as "latency" (although it is incorrect).
- `throughput`: Number of operations that Elasticsearch can perform within a certain time period, usually per second. See the *track reference* for a definition of what is meant by one "operation" for each operation type.
- `disk_io_write_bytes`: number of bytes that have been written to disk during the benchmark. On Linux this metric reports only the bytes that have been written by Elasticsearch, on Mac OS X it reports the number of bytes written by all processes.
- `disk_io_read_bytes`: number of bytes that have been read from disk during the benchmark. The same caveats apply on Mac OS X as for `disk_io_write_bytes`.
- `node_startup_time`: The time in seconds it took from process start until the node is up.
- `node_total_young_gen_gc_time`: The total runtime of the young generation garbage collector across the whole cluster as reported by the node stats API.
- `node_total_young_gen_gc_count`: The total number of young generation garbage collections across the whole cluster as reported by the node stats API.
- `node_total_old_gen_gc_time`: The total runtime of the old generation garbage collector across the whole cluster as reported by the node stats API.
- `node_total_old_gen_gc_count`: The total number of old generation garbage collections across the whole cluster as reported by the node stats API.
- `segments_count`: Total number of segments as reported by the indices stats API.
- `segments_memory_in_bytes`: Number of bytes used for segments as reported by the indices stats API.
- `segments_doc_values_memory_in_bytes`: Number of bytes used for doc values as reported by the indices stats API.
- `segments_stored_fields_memory_in_bytes`: Number of bytes used for stored fields as reported by the indices stats API.
- `segments_terms_memory_in_bytes`: Number of bytes used for terms as reported by the indices stats API.
- `segments_norms_memory_in_bytes`: Number of bytes used for norms as reported by the indices stats API.
- `segments_points_memory_in_bytes`: Number of bytes used for points as reported by the indices stats API.

- `merges_total_time`: Cumulative runtime of merges of primary shards, as reported by the indices stats API. Note that this is not Wall clock time (i.e. if M merge threads ran for N minutes, we will report M * N minutes, not N minutes). These metrics records also have a `per-shard` property that contains the times across primary shards in an array.
- `merges_total_count`: Cumulative number of merges of primary shards, as reported by indices stats API under `_all/primaries`.
- `merges_total_throttled_time`: Cumulative time within merges have been throttled as reported by the indices stats API. Note that this is not Wall clock time. These metrics records also have a `per-shard` property that contains the times across primary shards in an array.
- `indexing_total_time`: Cumulative time used for indexing of primary shards, as reported by the indices stats API. Note that this is not Wall clock time. These metrics records also have a `per-shard` property that contains the times across primary shards in an array.
- `indexing_throttle_time`: Cumulative time that indexing has been throttled, as reported by the indices stats API. Note that this is not Wall clock time. These metrics records also have a `per-shard` property that contains the times across primary shards in an array.
- `refresh_total_time`: Cumulative time used for index refresh of primary shards, as reported by the indices stats API. Note that this is not Wall clock time. These metrics records also have a `per-shard` property that contains the times across primary shards in an array.
- `refresh_total_count`: Cumulative number of refreshes of primary shards, as reported by indices stats API under `_all/primaries`.
- `flush_total_time`: Cumulative time used for index flush of primary shards, as reported by the indices stats API. Note that this is not Wall clock time. These metrics records also have a `per-shard` property that contains the times across primary shards in an array.
- `flush_total_count`: Cumulative number of flushes of primary shards, as reported by indices stats API under `_all/primaries`.
- `final_index_size_bytes`: Final resulting index size on the file system after all nodes have been shutdown at the end of the benchmark. It includes all files in the nodes' data directories (actual index files and translog).
- `store_size_in_bytes`: The size in bytes of the index (excluding the translog), as reported by the indices stats API.
- `translog_size_in_bytes`: The size in bytes of the translog, as reported by the indices stats API.
- `ml_processing_time`: A structure containing the minimum, mean, median and maximum bucket processing time in milliseconds per machine learning job. These metrics are only available if a machine learning job has been created in the respective benchmark.

2.21 Summary Report

At the end of each *race*, Rally shows a summary report. Below we'll explain the meaning of each line including a reference to its corresponding *metrics key* which can be helpful if you want to build your own reports in Kibana. Note that not every summary report will show all lines.

2.21.1 Cumulative indexing time of primary shards

- **Definition**: Cumulative time used for indexing as reported by the indices stats API. Note that this is not Wall clock time (i.e. if M indexing threads ran for N minutes, we will report M * N minutes, not N minutes).

- **Corresponding metrics key**: `indexing_total_time`

2.21.2 Cumulative indexing time across primary shards

- **Definition**: Minimum, median and maximum cumulative time used for indexing across primary shards as reported by the indices stats API.
- **Corresponding metrics key**: `indexing_total_time` (property: `per-shard`)

2.21.3 Cumulative indexing throttle time of primary shards

- **Definition**: Cumulative time that indexing has been throttled as reported by the indices stats API. Note that this is not Wall clock time (i.e. if M indexing threads ran for N minutes, we will report M * N minutes, not N minutes).
- **Corresponding metrics key**: `indexing_throttle_time`

2.21.4 Cumulative indexing throttle time across primary shards

- **Definition**: Minimum, median and maximum cumulative time used that indexing has been throttled across primary shards as reported by the indices stats API.
- **Corresponding metrics key**: `indexing_throttle_time` (property: `per-shard`)

2.21.5 Cumulative merge time of primary shards

- **Definition**: Cumulative runtime of merges of primary shards, as reported by the indices stats API. Note that this is not Wall clock time.
- **Corresponding metrics key**: `merges_total_time`

2.21.6 Cumulative merge count of primary shards

- **Definition**: Cumulative number of merges of primary shards, as reported by indices stats API under `_all/primaries`.
- **Corresponding metrics key**: `merges_total_count`

2.21.7 Cumulative merge time across primary shards

- **Definition**: Minimum, median and maximum cumulative time of merges across primary shards, as reported by the indices stats API.
- **Corresponding metrics key**: `merges_total_time` (property: `per-shard`)

2.21.8 Cumulative refresh time of primary shards

- **Definition**: Cumulative time used for index refresh of primary shards, as reported by the indices stats API. Note that this is not Wall clock time.
- **Corresponding metrics key**: `refresh_total_time`

2.21.9 Cumulative refresh count of primary shards

- **Definition**: Cumulative number of refreshes of primary shards, as reported by indices stats API under `_all/primaries`.
- **Corresponding metrics key**: `refresh_total_count`

2.21.10 Cumulative refresh time across primary shards

- **Definition**: Minimum, median and maximum cumulative time for index refresh across primary shards, as reported by the indices stats API.
- **Corresponding metrics key**: `refresh_total_time` (property: `per-shard`)

2.21.11 Cumulative flush time of primary shards

- **Definition**: Cumulative time used for index flush of primary shards, as reported by the indices stats API. Note that this is not Wall clock time.
- **Corresponding metrics key**: `flush_total_time`

2.21.12 Cumulative flush count of primary shards

- **Definition**: Cumulative number of flushes of primary shards, as reported by indices stats API under `_all/primaries`.
- **Corresponding metrics key**: `flush_total_count`

2.21.13 Cumulative flush time across primary shards

- **Definition**: Minimum, median and maximum time for index flush across primary shards as reported by the indices stats API.
- **Corresponding metrics key**: `flush_total_time` (property: `per-shard`)

2.21.14 Cumulative merge throttle time of primary shards

- **Definition**: Cumulative time within merges that have been throttled, as reported by the indices stats API. Note that this is not Wall clock time.
- **Corresponding metrics key**: `merges_total_throttled_time`

2.21.15 Cumulative merge throttle time across primary shards

- **Definition**: Minimum, median and maximum cumulative time that merges have been throttled across primary shards as reported by the indices stats API.
- **Corresponding metrics key**: `merges_total_throttled_time` (property: `per-shard`)

2.21.16 ML processing time

- **Definition**: Minimum, mean, median and maximum time in milliseconds that a machine learning job has spent processing a single bucket.
- **Corresponding metrics key**: `ml_processing_time`

2.21.17 Total Young Gen GC time

- **Definition**: The total runtime of the young generation garbage collector across the whole cluster as reported by the node stats API.
- **Corresponding metrics key**: `node_total_young_gen_gc_time`

2.21.18 Total Young Gen GC count

- **Definition**: The total number of young generation garbage collections across the whole cluster as reported by the node stats API.
- **Corresponding metrics key**: `node_total_young_gen_gc_count`

2.21.19 Total Old Gen GC time

- **Definition**: The total runtime of the old generation garbage collector across the whole cluster as reported by the node stats API.
- **Corresponding metrics key**: `node_total_old_gen_gc_time`

2.21.20 Total Old Gen GC count

- **Definition**: The total number of old generation garbage collections across the whole cluster as reported by the node stats API.
- **Corresponding metrics key**: `node_total_old_gen_gc_count`

2.21.21 Store size

- **Definition**: The size in bytes of the index (excluding the translog) as reported by the indices stats API.
- **Corresponding metrics key**: `store_size_in_bytes`

2.21.22 Translog size

- **Definition**: The size in bytes of the translog as reported by the indices stats API.
- **Corresponding metrics key**: `translog_size_in_bytes`

2.21.23 Heap used for `X`

Where `X` is one of:

- doc values
- terms
- norms
- points
- stored fields
- **Definition**: Number of bytes used for the corresponding item as reported by the indices stats API.
- **Corresponding metrics keys**: `segments_*_in_bytes`

2.21.24 Segment count

- **Definition**: Total number of segments as reported by the indices stats API.
- **Corresponding metrics key**: `segments_count`

2.21.25 Throughput

Rally reports the minimum, median and maximum throughput for each task.

- **Definition**: Number of operations that Elasticsearch can perform within a certain time period, usually per second.
- **Corresponding metrics key**: `throughput`

2.21.26 Latency

Rally reports several percentile numbers for each task. Which percentiles are shown depends on how many requests Rally could capture (i.e. Rally will not show a 99.99th percentile if it could only capture five samples because that would be a vanity metric).

- **Definition**: Time period between submission of a request and receiving the complete response. It also includes wait time, i.e. the time the request spends waiting until it is ready to be serviced by Elasticsearch.
- **Corresponding metrics key**: `latency`

2.21.27 Service time

Rally reports several percentile numbers for each task. Which percentiles are shown depends on how many requests Rally could capture (i.e. Rally will not show a 99.99th percentile if it could only capture five samples because that would be a vanity metric).

- **Definition**: Time period between start of request processing and receiving the complete response. This metric can easily be mixed up with `latency` but does not include waiting time. This is what most load testing tools refer to as "latency" (although it is incorrect).
- **Corresponding metrics key**: `service_time`

2.21.28 Error rate

- **Definition**: The ratio of erroneous responses relative to the total number of responses. Any exception thrown by the Python Elasticsearch client is considered erroneous (e.g. HTTP response codes 4xx, 5xx or network errors (network unreachable)). For specific details, check the reference documentation of the Elasticsearch client. Usually any error rate greater than zero is alerting. You should investigate the root cause by inspecting Rally and Elasticsearch logs and rerun the benchmark.
- **Corresponding metrics key**: `service_time`. Each `service_time` record has a `meta.success` flag. Rally simply counts how often this flag is `true` and `false` respectively.

2.22 Migration Guide

2.22.1 Migrating to Rally 2.0.1

Pipelines from-sources-complete and from-sources-skip-build are deprecated

Rally 2.0.1 caches source artifacts automatically in `~/.rally/benchmarks/distributions/src`. Therefore, it is not necessary anymore to explicitly skip the build with `--pipeline=from-sources-skip-build`. Specify `--pipeline=from-sources` instead. See the *pipeline reference documentation* for more details.

wait-for-recovery requires an `index` parameter

Previously, the `wait-for-recovery` operation checked all indices but with Rally 2.0.1 an `index` parameter is required and only that index (or index pattern) is checked.

2.22.2 Migrating to Rally 2.0.0

Minimum Python version is 3.8.0

Rally 2.0.0 requires Python 3.8.0. Check the *updated installation instructions* for more details.

JAVA_HOME and the bundled runtime JDK

Rally can optionally use the bundled runtime JDK by setting `--runtime-jdk="bundled"`. This setting will use the JDK that is bundled with Elasticsearch and not honor any `JAVA_HOME` settings you may have set.

Meta-Data for queries are omitted

Rally 2.0.0 does not determine query meta-data anymore by default to reduce the risk of client-side bottlenecks. The following meta-data fields are affected:

- `hits`
- `hits_relation`
- `timed_out`
- `took`

If you still want to retrieve them (risking skewed results due to additional overhead), set the new property `detailed-results` to `true` for any operation of type `search`.

Runner API uses asyncio

In order to support more concurrent clients in the future, Rally is moving from a synchronous model to an asynchronous model internally. With Rally 2.0.0 all custom runners need to be implemented using async APIs and a new bool argument `async_runner=True` needs to be provided upon registration. Below is an example how to migrate a custom runner function.

A custom runner prior to Rally 2.0.0:

```
def percolate(es, params):
    es.percolate(
       index="queries",
       doc_type="content",
       body=params["body"]
    )

def register(registry):
    registry.register_runner("percolate", percolate)
```

With Rally 2.0.0, the implementation changes as follows:

```
async def percolate(es, params):
    await es.percolate(
            index="queries",
            doc_type="content",
            body=params["body"]
          )

def register(registry):
    registry.register_runner("percolate", percolate, async_runner=True)
```

Apply to the following changes for each custom runner:

- Prefix the function signature with `async`.
- Add an `await` keyword before each Elasticsearch API call.
- Add `async_runner=True` as the last argument to the `register_runner` function.

For more details please refer to the updated documentation on *custom runners*.

`trial-id` and `trial-timestamp` are removed

Since Rally 1.4.0, Rally uses the properties `race-id` and `race-timestamp` when writing data to the Elasticsearch metrics store. The properties `trial-id` and `trial-timestamp` were populated but are removed in this release. Any visualizations that still rely on these properties need to be changed to the new ones.

2.22.3 Migrating to Rally 1.4.1

Document IDs are now padded with 0 instead of spaces

When Rally 1.4.1 generates document IDs, it will pad them with ‘0’ instead of ‘ ‘ - 0000000000 instead of ‘ 0’, etc. Elasticsearch has optimizations for numeric IDs, so observed performance in Elasticsearch should improve slightly.

2.22.4 Migrating to Rally 1.4.0

cluster-settings is deprecated in favor of the put-settings operation

Before Rally 1.4.0, cluster settings could be specified on the track with the `cluster-settings` property. This functionality is deprecated and you should set dynamic cluster settings via the new `put-settings` runner. Static settings should instead be set via `--car-params`.

Build logs are stored in Rally's log directory

If you benchmark source builds of Elasticsearch, Rally has previously stored the build output log in a race-specific directory. With this release, Rally will store the most recent build log in `/home/user/.rally/logs/build.log`.

Index size and Total Written are not included in the command line report

Elasticsearch nodes are now managed independently of benchmark execution and thus all system metrics ("index size" and "total written") may be determined after the command line report has been written. The corresponding metrics (`final_index_size_bytes` and `disk_io_write_bytes`) are still written to the Elasticsearch metrics store if one is configured.

Node details are omitted from race metadata

Before Rally 1.4.0, the file `race.json` contained node details (such as the number of cluster nodes or details about the nodes' operating system version) if Rally provisioned the cluster. With this release, this information is now omitted. This change also applies to the indices `rally-races*` in case you have setup an Elasticsearch metrics store. We recommend to use user tags in case such information is important, e.g. for visualising results.

`trial-id` and `trial-timestamp` are deprecated

With Rally 1.4.0, Rally will use the properties `race-id` and `race-timestamp` when writing data to the Elasticsearch metrics store. The properties `trial-id` and `trial-timestamp` are still populated but will be removed in a future release. Any visualizations that rely on these properties should be changed to the new ones.

Custom Parameter Sources

With Rally 1.4.0, we have changed the API for custom parameter sources. The `size()` method is now deprecated and is instead replaced with a new property called `infinite`. If you have previously returned `None` in `size()`, `infinite` should be set to `True`, otherwise `False`. Also, we recommend to implement the property `percent_completed` as Rally might not be able to determine progress in some cases. See below for some examples.

Old:

```
class CustomFiniteParamSource:
    # ...
    def size():
        return calculate_size()

    def params():
        return next_parameters()
```

(continues on next page)

(continued from previous page)

```
class CustomInfiniteParamSource:
    # ...
    def size():
        return None

    # ...
```

New:

```
class CustomFiniteParamSource:
    def __init__(self, track, params, **kwargs):
        self.infinite = False
        # to track progress
        self.current_invocation = 0

    # ...
    # Note that we have removed the size() method

    def params():
        self.current_invocation += 1
        return next_parameters()

    # Implementing this is optional but recommended for proper progress reports
    @property
    def percent_completed(self):
        # for demonstration purposes we use calculate_size() here
        # to determine the expected number of invocations. However, if
        # it is possible to determine this value upfront, it is best
        # to cache it in a field and just reuse the value
        return self.current_invocation / calculate_size()


class CustomInfiniteParamSource:
    def __init__(self, track, params, **kwargs):
        self.infinite = True
        # ...

    # ...
    # Note that we have removed the size() method
    # ...
```

2.22.5 Migrating to Rally 1.3.0

Races now stored by ID instead of timestamp

With Rally 1.3.0, Races will be stored by their Trial ID instead of their timestamp. This means that on disk, a given race will be found at `benchmarks/races/62d1e928-48b0-4d07-9899-07b45d031566/` instead of `benchmarks/races/2019-07-03-17-52-07`

Laps feature removed

The `--laps` parameter and corresponding multi-run trial functionality has been removed from execution and reporting. If you need lap functionality, the following shell script can be used instead:

```
RALLY_LAPS=3

for lap in $(seq 1 ${RALLY_LAPS})
do
  esrally --pipeline=benchmark-only --user-tag lap:$lap
done
```

2.22.6 Migrating to Rally 1.2.1

CPU usage is not measured anymore

With Rally 1.2.1, CPU usage will neither be measured nor reported. We suggest to use system monitoring tools like `mpstat`, `sar` or Metricbeat to measure CPU usage instead.

2.22.7 Migrating to Rally 1.1.0

`request-params` in operations are passed as is and not serialized

With Rally 1.1.0 any operations supporting the optional `request-params` property will pass the structure as is without attempting to serialize values. Until now, `request-params` relied on parameters being supported by the Elasticsearch Python client API calls. This means that for example boolean type parameters should be specified as strings i.e. `"true"` or `"false"` rather than `true/false`.

Example

Using `create-index` before `1.1.0`:

```
{
  "name": "create-all-indices",
  "operation-type": "create-index",
  "settings": {
    "index.number_of_shards": 1
  },
  "request-params": {
    "wait_for_active_shards": true
  }
}
```

Using `create-index` starting with `1.1.0`:

```
{
  "name": "create-all-indices",
  "operation-type": "create-index",
  "settings": {
    "index.number_of_shards": 1
  },
  "request-params": {
    "wait_for_active_shards": "true"
  }
}
```

2.22.8 Migrating to Rally 1.0.1

Logs are not rotated

With Rally 1.0.1 we have disabled automatic rotation of logs by default because it can lead to race conditions due to Rally's multi-process architecture. If you did not change the default out-of-the-box logging configuration, Rally will automatically fix your configuration. Otherwise, you need to replace all instances of `logging.handlers.TimedRotatingFileHandler` with `logging.handlers.WatchedFileHandler` to disable log rotation.

To rotate logs we recommend to use external tools like logrotate. See the following example as a starting point for your own `logrotate` configuration and ensure to replace the path `/home/user/.rally/logs/rally.log` with the proper one:

```
/home/user/.rally/logs/rally.log {
        # rotate daily
        daily
        # keep the last seven log files
        rotate 7
        # remove logs older than 14 days
        maxage 14
        # compress old logs ...
        compress
        # ... after moving them
        delaycompress
        # ignore missing log files
        missingok
        # don't attempt to rotate empty ones
        notifempty
}
```

2.22.9 Migrating to Rally 1.0.0

Handling of JDK versions

Previously the path to the JDK needed to be configured in Rally's configuration file (`~/.rally/rally.ini`) but this is too inflexible given the increased JDK release cadence. In order to keep up, we define now the allowed runtime JDKs in rally-teams per Elasticsearch version.

To resolve the path to the appropriate JDK you need to define the environment variable `JAVA_HOME` on each targeted machine.

You can also set version-specific environment variables, e.g. `JAVA7_HOME`, `JAVA8_HOME` or `JAVA10_HOME` which will take precedence over `JAVA_HOME`.

Note: Rally will choose the highest appropriate JDK per Elasticsearch version. You can use `--runtime-jdk` to force a specific JDK version but the path will still be resolved according to the logic above.

Custom Parameter Sources

In Rally 0.10.0 we have deprecated some parameter names in custom parameter sources. In Rally 1.0.0, these deprecated names have been removed. Therefore you need to replace the following parameter names if you use them in custom parameter sources:

Operation type	Old name	New name
search	use_request_cache	cache
search	request_params	request-params
search	items_per_page	results-per-page
bulk	action_metadata_present	action-metadata-present
force-merge	max_num_segments	max-num-segments

In Rally 0.9.0 the signature of custom parameter sources has also changed. In Rally 1.0.0 we have removed the backwards compatibility layer so you need to change the signatures.

Old:

```
# for parameter sources implemented as functions
def custom_param_source(indices, params):

# for parameter sources implemented as classes
class CustomParamSource:
    def __init__(self, indices, params):
```

New:

```
# for parameter sources implemented as functions
def custom_param_source(track, params, **kwargs):

# for parameter sources implemented as classes
class CustomParamSource:
    def __init__(self, track, params, **kwargs):
```

You can use the property `track.indices` to access indices.

2.22.10 Migrating to Rally 0.11.0

Versioned teams

Note: You can skip this section if you do not create custom Rally teams.

We have introduced versioned team specifications and consequently the directory structure changes. All cars and plugins need to reside in a version-specific subdirectory now. Up to now the structure of a team repository was as follows:

```
.
├── cars
│   ├── 1gheap.ini
│   ├── 2gheap.ini
│   ├── defaults.ini
│   ├── ea
│   │   └── config
│   │       └── jvm.options
│   ├── ea.ini
│   └── vanilla
│       └── config
│           ├── elasticsearch.yml
│           ├── jvm.options
```

(continues on next page)

(continued from previous page)

```
│           └── log4j2.properties
└── plugins
    ├── core-plugins.txt
    └── transport_nio
        ├── default
        │   └── config
        │       └── elasticsearch.yml
        └── transport.ini
```

Starting with Rally 0.11.0, Rally will look for a directory "v1" within `cars` and `plugins`. The files that should be copied to the Elasticsearch directory, need to be contained in a `templates` subdirectory. Therefore, the new structure is as follows:

```
.
├── cars
│   └── v1
│       ├── 1gheap.ini
│       ├── 2gheap.ini
│       ├── defaults.ini
│       ├── ea
│       │   └── templates
│       │       └── config
│       │           └── jvm.options
│       ├── ea.ini
│       └── vanilla
│           └── templates
│               └── config
│                   ├── elasticsearch.yml
│                   ├── jvm.options
│                   └── log4j2.properties
└── plugins
    └── v1
        ├── core-plugins.txt
        └── transport_nio
            ├── default
            │   └── templates
            │       └── config
            │           └── elasticsearch.yml
            └── transport.ini
```

It is also required that you create a file `variables.ini` for all your car config bases (optional for mixins). Therefore, the full directory structure is:

```
.
├── cars
│   └── v1
│       ├── 1gheap.ini
│       ├── 2gheap.ini
│       ├── defaults.ini
│       ├── ea
│       │   └── templates
│       │       └── config
│       │           └── jvm.options
│       ├── ea.ini
│       └── vanilla
│           ├── config.ini
```

(continues on next page)

(continued from previous page)

```
│           └── templates
│               └── config
│                   ├── elasticsearch.yml
│                   ├── jvm.options
│                   └── log4j2.properties
└── plugins
    └── v1
        ├── core-plugins.txt
        └── transport_nio
            ├── default
            │   └── templates
            │       └── config
            │           └── elasticsearch.yml
            └── transport.ini
```

For distribution-based builds, `config.ini` file needs to contain a section `variables` and a `release_url` property:

```
[variables]
release_url=https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-oss-{
↪{VERSION}}.tar.gz
```

2.22.11 Migrating to Rally 0.10.0

Removal of auto-detection and dependency on Gradle

We have removed the auto-detection and dependency on Gradle, required until now to build from source, in favor of the Gradle Wrapper which is present in the Elasticsearch repository for all branches >= 5.0.0.

Use full build command in plugin configuration

With Rally 0.10.0 we have removed the property `build.task` for plugin definitions, in the `source` section of the Rally configuration file. Instead, a new property `build.command` has been introduced where the **full build command** needs to be supplied.

The earlier syntax, to build a hypothetical plugin called `my-plugin` alongside Elasticsearch, required:

```
plugin.my-plugin.build.task = :my-plugin:plugin:assemble
```

This needs to be changed to the full command:

```
plugin.my-plugin.build.command = ./gradlew :my-plugin:plugin:assemble
```

Note that if you are configuring Plugins based on a released Elasticsearch version the command specified in `build.command` will be executed from the plugins root directory. It's likely this directory won't have the Gradle Wrapper so you'll need to specify the full path to a Gradle command e.g.:

```
plugin.my-plugin.build.command = /usr/local/bin/gradle :my-plugin:plugin:assemble
```

Check Building plugins from sources for more information.

Removal of operation type `index`

We have removed the operation type `index` which has been deprecated with Rally 0.8.0. Use `bulk` instead as operation type.

Removal of the command line parameter `--cluster-health`

We have removed the command line parameter `--cluster-health` which has been deprecated with Rally 0.8.0. When using Rally's standard tracks, specify the expected cluster health as a track parameter instead, e.g.: `--track-params="cluster_health:'yellow'"`.

Removal of index-automanagement

We have removed the possibility that Rally automatically deletes and creates indices. Therefore, you need to add the following definitions explicitly at the beginning of a schedule if you want Rally to create declared indices:

```
"schedule": [
  {
    "operation": "delete-index"
  },
  {
    "operation": {
      "operation-type": "create-index",
      "settings": {
        "index.number_of_replicas": 0
      }
    }
  },
  {
    "operation": {
      "operation-type": "cluster-health",
      "request-params": {
        "wait_for_status": "green"
      }
    }
  }
```

The example above also shows how to provide per-challenge index settings. If per-challenge index settings are not required, you can just specify them in the index definition file.

This behavior applies similarly to index templates as well.

Custom Parameter Sources

We have aligned the internal names between parameter sources and runners with the ones that are specified by the user in the track file. If you have implemented custom parameter sources or runners, adjust the parameter names as follows:

Operation type	Old name	New name
search	use_request_cache	cache
search	request_params	request-params
search	items_per_page	results-per-page
bulk	action_metadata_present	action-metadata-present
force-merge	max_num_segments	max-num-segments

2.22.12 Migrating to Rally 0.9.0

Track Syntax

With Rally 0.9.0, we have changed the track file format. While the previous format is still supported with deprecation warnings, we recommend that you adapt your tracks as we will remove the deprecated syntax with the next minor release.

Below is an example of a track with the previous syntax:

```
{
  "description": "Tutorial benchmark for Rally",
  "data-url": "http://benchmarks.elasticsearch.org.s3.amazonaws.com/corpora/geonames",
  "indices": [
    {
      "name": "geonames",
      "types": [
        {
          "name": "type",
          "mapping": "mappings.json",
          "documents": "documents.json",
          "document-count": 8647880,
          "uncompressed-bytes": 2790927196
        }
      ]
    }
  ],
  "challenge": {
    "name": "index-only",
    "index-settings": {
      "index.number_of_replicas": 0
    },
    "schedule": [
      {
        "operation": {
          "operation-type": "bulk",
          "bulk-size": 5000
        },
        "warmup-time-period": 120,
        "clients": 8
      }
    ]
  }
}
```

Before Rally 0.9.0, indices have been created implicitly. We will remove this ability and thus you need to tell Rally explicitly that you want to create indices. With Rally 0.9.0 your track should look as follows:

```
{
  "description": "Tutorial benchmark for Rally",
  "indices": [
    {
      "name": "geonames",
      "body": "index.json",
      "auto-managed": false,
      "types": [ "type" ]
    }
```

(continues on next page)

(continued from previous page)

```
  ],
  "corpora": [
    {
      "name": "geonames",
      "documents": [
        {
          "base-url": "http://benchmarks.elasticsearch.org.s3.amazonaws.com/corpora/
↪geonames",
          "source-file": "documents.json",
          "document-count": 8647880,
          "uncompressed-bytes": 2790927196
        }
      ]
    }
  ],
  "challenge": {
    "name": "index-only",
    "schedule": [
      {
        "operation": "delete-index"
      },
      {
        "operation": {
          "operation-type": "create-index",
          "settings": {
            "index.number_of_replicas": 0
          }
        }
      },
      {
        "operation": {
          "operation-type": "cluster-health",
          "request-params": {
            "wait_for_status": "green"
          }
        }
      },
      {
        "operation": {
          "operation-type": "bulk",
          "bulk-size": 5000
        },
        "warmup-time-period": 120,
        "clients": 8
      }
    ]
  }
}
```

Let's go through the necessary changes one by one.

Define the document corpus separately

Previously you had to define the document corpus together with the document type. In order to allow you to reuse existing document corpora across tracks, you now need to specify any document corpora separately:

```
"corpora": [
  {
    "name": "geonames",
    "documents": [
      {
        "base-url": "http://benchmarks.elasticsearch.org.s3.amazonaws.com/corpora/
↪geonames",
        "source-file": "documents.json",
        "document-count": 8647880,
        "uncompressed-bytes": 2790927196
      }
    ]
  }
]
```

Note that this is just a simple example that should cover the most basic case. Be sure to check the *track reference* for all details.

Change the index definition

The new index definition now looks as follows:

```
{
  "name": "geonames",
  "body": "index.json",
  "auto-managed": false,
  "types": [ "type" ]
}
```

We have added a `body` property to the index and removed the `mapping` property from the type. In fact, the only information that we need about the document type is its name, hence it is now a simple list of strings. Just put all type mappings now into the `mappings` property of the index definition; see also the create index API documentation.

Secondly, we have disabled index auto-management by setting `auto-managed` to `false`. This allows us to define explicit tasks below to manage our index. Note that index auto-management is still working in Rally 0.9.0 but it will be removed with the next minor release Rally 0.10.0.

Explicitly delete and recreate the index

We have also added three tasks at the beginning of the schedule:

```
{
  "operation": "delete-index"
},
{
  "operation": {
    "operation-type": "create-index",
    "settings": {
      "index.number_of_replicas": 0
    }
  }
},
{
  "operation": {
```

(continues on next page)

(continued from previous page)

```
    "operation-type": "cluster-health",
    "request-params": {
      "wait_for_status": "green"
    }
  }
}
```

These tasks represent what Rally previously did implicitly.

The first task will delete all indices that have been declared in the `indices` section if they existed previously. This ensures that we don't have any leftovers from previous benchmarks.

After that we will create all indices that have been declared in the `indices` section. Note that we have also removed the special property `index-settings` and moved it to the `settings` parameter of `create-index`. Rally will merge any settings from the index body definition with these settings. This means you should define settings that are always the same in the index body and settings that change from challenge to challenge in the `settings` property.

Finally, Rally will check that the cluster health is green. If you want to be able to override the cluster health check parameters from the command line, you can leverage Rally's track parameter feature:

```
{
  "operation": {
    "operation-type": "cluster-health",
    "request-params": {
      "wait_for_status": "{{ cluster_health|default('green') }}"
    }
  }
}
```

If you don't specify anything on the command line, Rally will use the default value but you can e.g. specify `--track-params="cluster_health:'yellow'"` so Rally will check for (at least) a yellow cluster health status.

Note that you can *customize these operations*.

Custom Parameter Sources

With Rally 0.9.0, the API for custom parameter sources has changed. Previously, the following syntax was valid:

```
# for parameter sources implemented as functions
def custom_param_source(indices, params):

# for parameter sources implemented as classes
class CustomParamSource:
    def __init__(self, indices, params):
```

With Rally 0.9.0, the signatures need to be changed to:

```
# for parameter sources implemented as functions
def custom_param_source(track, params, **kwargs):

# for parameter sources implemented as classes
class CustomParamSource:
    def __init__(self, track, params, **kwargs):
```

Rally will issue a warning along the lines of `Parameter source 'custom_param_source' is using deprecated method signature` if your track is affected. If you need access to the `indices` list, you can call `track.indices` to retrieve it from the track.

2.23 Frequently Asked Questions (FAQ)

2.23.1 A benchmark aborts with `Couldn't find a tar.gz distribution`. What's the problem?

This error occurs when Rally cannot build an Elasticsearch distribution from source code. The most likely cause is that there is some problem building the Elasticsearch distribution.

To see what's the problem, try building Elasticsearch yourself. First, find out where the source code is located (run `grep src ~/.rally/rally.ini`). Then change to the directory (`src.root.dir` + `elasticsearch.src.subdir` which is usually `~/.rally/benchmarks/src/elasticsearch`) and run the following commands:

```
./gradlew clean
./gradlew :distribution:tar:assemble
```

By that you are mimicking what Rally does. Fix any errors that show up here and then retry.

2.23.2 Where does Rally get the benchmark data from?

Rally comes with a set of tracks out of the box which we maintain in the rally-tracks repository on Github. This repository contains the track descriptions. The actual data are stored as compressed files in an S3 bucket.

2.23.3 Will Rally destroy my existing indices?

First of all: Please (please, please) do NOT run Rally against your production cluster if you are just getting started with it. You have been warned.

Depending on the track, Rally will delete and create one or more indices. For example, the geonames track specifies that Rally should create an index named "geonames" and Rally will assume it can do to this index whatever it wants. Specifically, Rally will check at the beginning of a race if the index "geonames" exists and delete it. After that it creates a new empty "geonames" index and runs the benchmark. So if you benchmark against your own cluster (by specifying the `benchmark-only` *pipeline*) and this cluster contains an index that is called "geonames" you will lose (all) data if you run Rally against it. Rally will neither read nor write (or delete) any other index. So if you apply the usual care nothing bad can happen.

2.23.4 What does *latency* and *service_time* mean and how do they related to the *took* field that Elasticsearch returns?

Let's start with the *took* field of Elasticsearch. *took* is the time needed by Elasticsearch to process a request. As it is determined on the server, it can neither include the time it took the client to send the data to Elasticsearch nor the time it took Elasticsearch to send it to the client. This time is captured by *service_time*, i.e. it is the time period from the start of a request (on the client) until it has received the response.

The explanation of *latency* is a bit more involved. First of all, Rally defines two benchmarking modes:

- Throughput benchmarking mode: In this mode, Rally will issue requests as fast as it can, i.e. as soon as it receives a response, it will issue the next request. This is ideal for benchmarking indexing. In this mode `latency == service_time`.
- Throughput-throttled mode: If you define a specific target throughput rate in a track, for example 100 requests per second (you should choose this number based on the traffic pattern that you experience in your production environment), then Rally will define a schedule internally and will issue requests according to this schedule regardless how fast Elasticsearch can respond. To put it differently: Imagine you want to grab a coffee on your way to work. You make this decision independently of all the other people going to the coffee shop so it is possible that you need to wait before you can tell the barista which coffee you want. The time it takes the barista to make your coffee is the service time. The service time is independent of the number of customers in the coffee shop. However, you as a customer also care about the length of the waiting line which depends on the number of customers in the coffee shop. The time it takes between you entering the coffee shop and taking your first sip of coffee is latency.

If you are interested in latency measurement, we recommend you watch the following talks:

"How NOT to Measure Latency" by Gil Tene:

Benchmarking Elasticsearch with Rally by Daniel Mitterdorfer:

2.23.5 Where and how long does Rally keep its data?

Rally stores a lot of data (this is just the nature of a benchmark) so you should keep an eye on disk usage. All data are kept in `~/.rally` and Rally does not implicitly delete them. These are the most important directories:

- `~/.rally/logs`: Contains all log files. Logs are rotated daily. If you don't need the logs anymore, you can safely wipe this directory.
- `~/.rally/benchmarks/races`: telemetry data, Elasticsearch logs and even complete Elasticsearch installations including the data directory if a benchmark failed. If you don't need the data anymore, you can safely wipe this directory.
- `~/.rally/benchmarks/src`: the Elasticsearch Github repository (only if you had Rally build Elasticsearch from sources at least once).
- `~/.rally/benchmarks/data`: the benchmark data sets. This directory can get very huge (way more than 100 GB if you want to try all default tracks). You can delete the files in this directory but keep in mind that Rally may needs to download them again.
- `~/.rally/benchmarks/distributions`: Contains all downloaded Elasticsearch distributions.

There are a few more directories but the ones above are the most disk-hogging ones.

2.23.6 Does Rally spy on me?

No. Rally does not collect or send any usage data and also the complete source code is open. We do value your feedback a lot though and if you got any ideas for improvements, found a bug or have any other feedback, head over to Rally's Discuss forum or raise an issue on Github.

2.23.7 Do I need an Internet connection?

You do NOT need Internet access on any node of your Elasticsearch cluster but the machine where you start Rally needs an Internet connection to download track data sets and Elasticsearch distributions. After it has downloaded all data, an Internet connection is not required anymore and you can specify `--offline`. If Rally detects no active Internet connection, it will automatically enable offline mode and warn you.

We have a dedicated documentation page for *running Rally offline* which should cover all necessary details.

2.24 Glossary

track A *track* is the description of one or more benchmarking scenarios with a specific document corpus. It defines for example the involved indices, data files and which operations are invoked. List the available tracks with `esrally list tracks`. Although Rally ships with some tracks out of the box, you should usually *create your own track* based on your own data.

challenge A challenge describes one benchmarking scenario, for example indexing documents at maximum throughput with 4 clients while issuing term and phrase queries from another two clients rate-limited at 10 queries per second each. It is always specified in the context of a track. See the available challenges by listing the corresponding tracks with `esrally list tracks`.

car A *car* is a specific configuration of an Elasticsearch cluster that is benchmarked, for example the out-of-the-box configuration, a configuration with a specific heap size or a custom logging configuration. List the available cars with `esrally list cars`.

telemetry *Telemetry* is used in Rally to gather metrics about the car, for example CPU usage or index size.

race A *race* is one invocation of the Rally binary. Another name for that is one "benchmarking trial". During a race, Rally runs one challenge on a track with the given car.

tournament A tournament is a comparison of two races. You can use Rally's *tournament mode* for that.

2.25 Community Resources

Below are a few community resources about Rally. If you find an interesting article, talk or custom tracks, raise an issue or open a pull request.

2.25.1 Talks

2.25.2 Articles

Using Rally to benchmark Elasticsearch queries by Darren Smith

CHAPTER 3

License

Index