ripe-atlas-monitor Documentation

Release 0.1.0

Pier Carlo Chiodi

December 27, 2016

Contents

A Python tool to monitor results collected by RIPE Atlas probes and verify they match against predefined expected values.

CHAPTER 1

How does it work?

On the basis of a RIPE Atlas measurement previously created, you define a *monitor* by declaring which results you expect that probes should produce: *rules* are used to map probes and their *expected results*. Depending on whether the collected results match the expectations, custom *actions* are performed: to log the result, to send an email, a syslog message or to run an external program.

```
descr: Check network reachability
matching_rules:
- descr: Probes from France via AS64496
  src_country: FR
  expected_results: ViaAS64496
  actions: EMailToNOC
- descr: RTT from AS64499 and AS64500 below 50ms
  src_as:
  - 64499
  - 64500
  expected_results: LowRTT
  actions: EMailToNOC
expected_results:
  ViaAS64496:
    upstream_as: 64496
  LowRTT:
    rtt: 50
actions:
  EMailToNOC:
    kind: email
    to_addr: noc@agreatcompany.org
    subject: "ripe-atlas-monitor: unexpected results"
measurement-id: 123456789
```

CHAPTER 2

Contents

2.1 QuickStart

2.1.1 Step 1: install dependencies

Some libraries **ripe-atlas-monitor** depends on need to be compiled and require a compiler and Python's dev libraries:

```
$ # on Debian/Ubuntu:
$ sudo apt-get install python-dev libffi-dev libssl-dev

$ # on CentOS:
$ sudo yum install gcc libffi-devel openssl-devel
```

Strongly suggested: install `pip` and setup a virtualenv:

```
$ # on Debian/Ubuntu:
$ sudo apt-get install python-virtualenv

$ # on CentOS:
$ sudo yum install epel-release
$ sudo yum install python-pip python-virtualenv

$ # setup a virtualenv
$ mkdir ripe-atlas-monitor
$ cd ripe-atlas-monitor
$ virtualenv venv
$ source venv/bin/activate
```

More: `virtualenv` installation and usage.

2.1.2 Step 2: install ripe-atlas-monitor

Install latest **ripe-atlas-monitor** version from PyPI:

```
$ pip install ripe-atlas-monitor

$ # to enable bash autocomplete:
$ eval "$(register-python-argcomplete ripe-atlas-monitor)"
```

More: installation options.

2.1.3 Step 3: global configuration

Create the `var` directory and let the config file to be inizialized; set (at least) the `var` parameter:

```
$ # directory where ripe-atlas-monitor can write a bunch of data
$ mkdir var
$ ripe-atlas-monitor init-config
```

```
var: /path/to/ripe-atlas-monitor/var
```

More: global configuration options.

2.1.4 Step 4: create a new monitor and customize its configuration

The `analyze` command can help you defining your rules by giving an overview of the results for a specific measurement, as elaborated by **ripe-atlas-monitor**:

```
$ ripe-atlas-monitor analyze --measurement-id 1234567890
```

More: Results analysis.

Once you have a clear idea how your rules should look like, create and edit a new monitor:

```
$ ripe-atlas-monitor init-monitor -m MonitorName
```

More: how monitors work and syntax.

Alternatively, you can take a look at the sample monitors provided within the examples directory.

2.1.5 Step 5: run the brand new monitor

```
$ ripe-atlas-monitor run -m MonitorName --latest -vvv
```

More: execution modes and options.

2.2 Requirements & Installation

2.2.1 Requirements

This is a tool written in Python for Linux environments; currently, Python 2.7 and 3.4 are supported. Windows is not supported at all.

It is based on two RIPE NCC packages: RIPE Atlas Sagan and RIPE Atlas Cousteau, both available on GitHub and PyPI. It also has some other dependencies, they are reported in the *setup.py* file and managed by the *pip* installer.

Some libraries need to be compiled and they require a compiler and development libraries for Python.

- On **Debian/Ubuntu** the following system packages need to be installed:

```
$ sudo apt-get install python-dev libffi-dev libssl-dev
```

  Since `pip` and `virtualenv` are also strongly suggested, you may need to install them too:

```
$ sudo apt-get install python-virtualenv python-pip
```

- On **CentOS**, the following packages are needed:

```
    $ sudo yum install gcc libffi-devel openssl-devel

    $ # for pip and virtualenv:
    $ sudo yum install epel-release
    $ sudo yum install python-pip python-virtualenv
```

2.2.2 Installation

Even if you can manually install it and run it as a system package, `pip` installation and `virtualenv` use are strongly recommended to ease installation and dependencies management and to have it running within an isolated environment.

More: pip installation, virtualenv installation.

Setup a virtualenv

Virtualenv usage is documented here, but the following should be enough in most cases:

```
$ mkdir ripe-atlas-monitor
$ cd ripe-atlas-monitor
$ virtualenv venv
$ source venv/bin/activate
```

Installation from PyPI

Python `pip` can install packages both globally (system wide) and on a per-user basis. To avoid conflicts with other packages, the second way is the preferred one. It can be achieved using the `virtualenv` tool (the preferred way) or passing the `--user` argument to `pip`, so that the package will be installed within the `$HOME/.local` directory.

```
$ # using virtualenv
$ pip install ripe-atlas-monitor

$ # in your user's local dir
$ pip install --user ripe-atlas-monitor
```

Installation from GitHub

If you just want to use the latest code on the `master` branch on GitHub, you can install it with

```
$ pip install git+https://github.com/pierky/ripe-atlas-monitor.git
```

“Editable” installation

If you want to contribute to the code, you can clone the repository and install it using the `-e` argument of `pip`; you’ll have it installed in a local directory where you can edit it and see the results without having to install it every time:

```
$ pip install -e git+https://github.com/YOUR_USERNAME/ripe-atlas-monitor.git#egg=ripe-atlas-monitor
```

See also: How to contribute.

Bash autocomplete

To enable bash autocomplete, register the **ripe-atlas-monitor** script and update your shell preferences:

```
eval "$(register-python-argcomplete ripe-atlas-monitor)"
```

If you want it to be enabled on every access, you can it to your `.bashrc` file.

2.3 Global configuration

The global configuration file contains some options that are used by the program to run: the working directory path, logging options, default values for some actions and so on.

By default, **ripe-atlas-monitor** looks for this file in `$HOME/.config/ripe-atlas-monitor` (if `$HOME` is not defined, it tries with `/etc/ripe-atlas-monitor/config.cfg`), but this path can be set with the `--cfg` command line argument.

Only one parameter is really needed, it is the `var` directory used by the program to store its monitors configuration files and a bunch of other data (IP addresses cache, running status).

You can initialize the global configuration file by executing `ripe-atlas-monitor init-config`: this command copies the template file to the default path. Add the `--cfg` argument to use a custom path.

Comments within the file itself should be enough to explain the various options. If you want to take a look at it, you can find it on GitHub.

2.4 Commands

2.4.1 Results analysis

The `analyze` command can be used to have an overview of results received from a measurement and how they are elaborated by **ripe-atlas-monitor**:

```
$ ripe-atlas-monitor analyze --measurement-id 1234567890

$ ripe-atlas-monitor analyze -m MonitorName
```

Some heuristics provide aggregated metrics for most of the measurement's types:

- RTTs distribution (*ping, traceroute*)
- target responded or not (*ping, traceroute*)
- destination IP addresses (*ping, traceroute, ssl*)
- SSL certificate fingerprints (*ssl*)
- destination AS numbers and upstream ASNs (*traceroute*)
- AS paths (*traceroute*)
- DNS RCODEs (*dns*)
- DNS responses' flags combinations (*dns*)
- EDNS status and NSID option (*dns*)
- DNS answers (*dns*)

Results of each metric are grouped on the basis of common patterns and sorted by the number of matching probes.

Example analysis of measurement ID 1674977, a traceroute from 50 probes all over the world toward www.ripe.net (see it on RIPE Atlas Tracepath):

```
$ ripe-atlas-monitor analyze --measurement-id 1674977

Downloading and processing results... please wait
Median RTTs:

      < 30 ms: 25 times, probe ID 10001 (AS3265, NL), probe ID 10012 (AS3265, NL), probe ID 10039 (AS

   30 - 60 ms: 14 times, probe ID 10068 (AS34594, HR), probe ID 10772 (AS12552, SE), probe ID 10816

    >= 180 ms: 5 times, probe ID 10509 (AS1273, HK), probe ID 10510 (AS1273, SG), probe ID 12468 (AS

 150 - 180 ms: 3 times, probe ID 10313 (US), probe ID 13631 (AS21502, FR), probe ID 14856 (AS7922, US

  (use the --show-all-rtts argument to show the full list)

Destination responded:

 yes: 38 times, probe ID 10001 (AS3265, NL), probe ID 10012 (AS3265, NL), probe ID 10068 (AS34594, HR

  no: 11 times, probe ID 10039 (AS701, US), probe ID 10460 (AS7155, GB), probe ID 10922 (RU), ...

Unique destination IP addresses:

 193.0.6.139: 49 times, probe ID 10001 (AS3265, NL), probe ID 10012 (AS3265, NL), probe ID 10039 (AS7

Destination AS:

  3333: 38 times, probe ID 10001 (AS3265, NL), probe ID 10012 (AS3265, NL), probe ID 10068 (AS34594,

 12513: 1 time, probe ID 12277 (AS12513, GB)

  7922: 1 time, probe ID 16134 (AS7922, US)

  7155: 1 time, probe ID 10460 (AS7155, GB)

  6830: 1 time, probe ID 12224 (AS6830, NL)

  5089: 1 time, probe ID 13335 (AS5089, GB)

  3320: 1 time, probe ID 11059 (AS3320, DE)

  3269: 1 time, probe ID 4228 (AS3269, IT)

   701: 1 time, probe ID 10039 (AS701, US)

Upstream AS:

  1200: 24 times, probe ID 10001 (AS3265, NL), probe ID 10012 (AS3265, NL), probe ID 10273 (AS9143, N

  1299: 3 times, probe ID 10068 (AS34594, HR), probe ID 11586 (AS29056, AT), probe ID 16063 (AS6830,

  3356: 2 times, probe ID 10313 (US), probe ID 14856 (AS7922, US)

 33765: 1 time, probe ID 15282 (AS33765, TZ)
 31213: 1 time, probe ID 11418 (AS39087, RU)

 21502: 1 time, probe ID 13631 (AS21502, FR)

 12513: 1 time, probe ID 10953 (AS12513, GB)

  8218: 1 time, probe ID 14175 (AS24651, LV)

  4755: 1 time, probe ID 14593 (AS4755, IN)

  2856: 1 time, probe ID 11610 (AS2856, GB)

  Only top 10 most common shown.
  (use the --show-all-upstream-asns argument to show the full list)

Most common ASs sequences:

       1200 3333: 24 times

     S 1200 3333: 14 times

          S 1200: 14 times

          S 3333: 5 times

       1299 3333: 3 times

     S 1299 3333: 2 times

  9002 1200 3333: 2 times

  3356 1200 3333: 2 times

 15589 1200 3333: 2 times

          S 6830: 2 times

  (use the --show-all-aspaths argument to show the full list)

Unique AS paths (with IXPs networks):

 S IX 2603 3333: 1 time, probe ID 11585 (AS29518, SE)
```

The `--probes` and `--countries` arguments can be used to restrict the analysis to results produced by a limited set of probes by specifying their IDs or the source countries.

```
$ ripe-atlas-monitor analyze --measurement-id 1234567890 --probes 1,23,456
```

The `--key` argument can be used to provide a RIPE Atlas key needed to fetch the results. Other arguments may be used to display statistics about probes distribution and to show sub-results, grouping them by country or by source AS: the `--help` will show all of these options.

Data that form the analysis report can be printed in JSON format using the `--use-json` argument.

2.4.2 Monitors' configuration management

Some commands can be used to manage monitors' configuration:

- `init-monitor`: initializes a new monitor configuration by cloning the template file;
- `edit-monitor`: opens the monitor's configuration file with the default text editor (`$EDITOR` or `misc.editor` global config option);
- `check-monitor`: verifies that the monitor's configuration syntax is valid and conforming to the measurement's type. The `-v` argument can be used to display an explanatory description of the given configuration as interpreted by the program.

```
$ ripe-atlas-monitor [init-monitor | edit-monitor | check-monitor] -m MonitorName
```

2.4.3 Execution modes

There are some ways this tool can be executed, depending on how many concurrent monitors you want to run and which measurement results you want to consider.

The `-v` argument is common to all the scenarios and allow to set the verbosity level:

- 0: only warnings and errors are produced;
- 1 (`-v`): messages from logging actions are produced;
- 2 (`-vv`): results from matching rules are produced too;
- 3 (`-vvv`): information messages are logged (internal decisions about rules and results processing);
- 4 (`-vvvv`): debug messages are logged too, useful to debug monitors' configurations.

Single monitor: `run` command

The `run` command allows to execute a single monitor. It is mostly useful to process one-off measurements, to schedule execution or to debug monitors' configurations.

```
$ ripe-atlas-monitor run -m MonitorName -vvv
```

In this mode, the `--start`, `--stop` and `--latest` arguments allow to set the time frame for the measurement's results to download, unless the monitor has the `stream` option set to use RIPE Atlas result streaming. The `--probes` and `--countries` arguments can be used to restrict the processing to results produced by a limited set of probes by specifying their IDs or the source countries.

Time frame options

By default, for measurements which are still running, results are fetched continously every *measurement's interval* seconds, starting from the time of the last received result.

- The `--start` and `--stop` arguments set the lower and upper bounds for results downloading and processing. They can be used togheter or separately.
- If the `--start` argument is not given, results are downloaded starting from the last processed result's timestamp, or from the last 7 days (configurable in the global config) if the measurement has not been processed yet.
- If the `--stop` argument is missing, results up to the last produced one are downloaded.
- The `--latest` argument can be used when the other two are not passed and it allows to download the latest results only.
- For running measurements, the `--dont-wait` argument allows to run a monitor against up to date results then exiting, without waiting for measurement's interval before running it again.

Scheduling monitors

Execution of **ripe-atlas-monitor** can be scheduled (using `crontab` for example) in order to periodically monitor measurements' results.

For continous measurements (those which are not stopped and keep producing results) the `--dont-wait` argument is particularly suggested, so that at each execution the program downloads and processes the results collected since the previous one.

Note: Since only one instance of **ripe-atlas-monitor** at a time can be executed, if you plan to run multiple monitors be careful to schedule them in order to avoid overlapping running; alternatively consider using the `daemonize` command (see below).

If you are using a virtualenv, you can point your cron's job at the full `python` executable that is in the virtualenv's `bin` directory...

```
1 * * * * /home/USERNAME/ripe-atlas-monitor/venv/bin/python /home/USERNAME/ripe-atlas-monitor/venv/bi
```

... or you can write a wrapper bash script that sets up the virtualenv and then runs your command...

```
#! /bin/bash
cd /home/USERNAME/ripe-atlas-monitor/venv/
source bin/activate
"$@"
```

```
1 * * * * /home/USERNAME/ripe-atlas-monitor/setup_venv_and_run ripe-atlas-monitor -m MonitorName --do
```

Multiple monitors: `daemonize` command

Note: This mode is highly experimental

The `daemonize` command allows to run multiple monitors within a single instance of **ripe-atlas-monitor** by forking the main process into many subprocesses, one for each monitor. This mode does not allow to use time frame arguments, results are downloaded starting from the last received one for each measurement. This mode is mostly suitable for streaming monitors or continous measurements.

```
$ ripe-atlas-monitor daemonize -m Monitor1Name -m Monitor2Name
```

2.5 Monitors: how they work

Monitors are the core of the program. You can initialize their configuration with `ripe-atlas-monitor init-monitor -m monitor_name`: a monitor template file will be created and opened for customization using the preferred text editor (which can also be set within the global configuration file or via the `$EDITOR` environment variable).

2.5.1 How they work

You have a RIPE Atlas measurement;

- probes involved in the measurement collect some results (ping, traceroute, ...);

- a **ripe-atlas-monitor**'s monitor is executed;
- results for the aforementioned measurement are downloaded and elaborated;
- for each result, the probe's information (ID, country, ASN) are matched against a set of rules;
- if a matching condition is found, the result collected by that probe is matched against a set of results you expected from that probe;
- actions (email, syslog, external programs) are performed on the basis of this process's output.

All this is written in YAML files, one for each monitor you want to configure:

```
descr: Check network reachability
matching_rules:
- descr: Probes from France via AS64496
  src_country: FR
  expected_results: ViaAS64496
  actions: EMailToNOC
- descr: RTT from AS64499 and AS64500 below 50ms
  src_as:
  - 64499
  - 64500
  expected_results: LowRTT
  actions: EMailToNOC
expected_results:
  ViaAS64496:
    upstream_as: 64496
  LowRTT:
    rtt: 50
actions:
  EMailToNOC:
    kind: email
    to_addr: noc@agreatcompany.org
    subject: "ripe-atlas-monitor: unexpected results"
measurement-id: 123456789
```

For the complete syntax of monitors' configuration file please refer to Monitor configuration syntax.

2.5.2 Kinds of monitors

Depending on the measurement they are configured to use and which command is used to run them, monitors can be grouped into 3 categories:

- *one-off* monitors are those used to process one-off measurements: they are executed using the `--latest` argument of **ripe-atlas-monitor** to download only the latest results, or they can be executed using both the `--start` and `--stop` command line argument in order to define a specific time frame within which results are downloaded;
- *continous* monitors are used to continously process results for those measurements which have not been stopped yet: results are downloaded and processed once every *x* seconds, where *x* is the `interval` value of the measurement itself; when the `--start` argument of **ripe-atlas-monitor** is used, results are downloaded starting at that time, otherwise results are downloaded starting from the timestamp of the last processed result;
- *streaming* monitors, which are those that use RIPE Atlas result streaming.

The type of monitor is not written anywhere, it's derived from the commands used to run **ripe-atlas-monitor**. For example, the same monitor can be run using `ripe-atlas-monitor run -m MonitorName --measurement-id 123456 --latest` to process the latest results from the measurement ID 123456, but also using `ripe-atlas-monitor daemonize -m MonitorName` to continously process results from the measurement reported in the `measurement-id` attribute of its configuration file. It can be also run in streaming mode, by using the `--stream` command line argument (provided that the measurement is still running).

2.5.3 Expected results criteria

Expected results can be of various kinds, depending on the measurement's type, and various criteria can be used to verify collected results.

Traceroute measurements can be used to monitor **AS path toward a destination**, ping measurements to test **network reachability** and performance, SSL measurements to be sure that the certificates received by a probe match the **expected fingerprints** and that TLS connections are not hijacked on their way, DNS measurements to verify **host name resolution**.

For the full list of implemented criteria please read Monitor configuration syntax.

2.5.4 Advanced use

Configuration syntax "tricks" and *internal labels* allow to describe complex scenarios.

Excluding probes from processing

A rule with no `expected_results` and the `process_next` attribute to its default value False (or missing) allows to stop further processing for those probes which match the rule's criteria:

```
matching_rules:
- descr: Do not process results for probe ID 123 and 456
  probe_id:
  - 123
  - 456
```

Match all probes except those...

The `reverse` attribute of a rule, when set to True, allows to match all the probes which do not meet the given criteria:

```
matching_rules:
- descr: All probes except those from AS64496
  src_as: 64496
  reverse: True
```

Actions execution

The `when` attribute of an action can be used to set when it has to be performed:

- `on_match`, the action is performed when the collected result matches one of the expected values, or when the rule has no expected results at all;
- `on_mismatch`, the action is performed when the collected result does not match the expected values;
- `always`, well, the action is always performed, independently of results.

Internal labels

Actions can be used to attach internal labels to probes on the basis of rules and results processing. These labels can be subsequently used to match probes against specific rules.

```
matching_rules:
- descr: Set 'VIP' (Very Important Probe) label to ID 123 and 456
  probe_id:
  - 123
  - 456
  process_next: True
  actions: SetVIPLabel
- descr: Set 'VIP' label to Italian probes too
  src_country: IT
  process_next: True
  actions: SetVIPLabel
- descr: VIPs must have low RTT
  internal_labels: VIP
  expected_results: LowRTT
actions:
  SetVIPLabel:
    when: always
    kind: label
    op: add
    label_name: VIP
```

Integration with ripe-atlas-tools (Magellan)

Magellan is the official command-line client for RIPE Atlas. It allows, moreover, to create new measurements from the command line. It can be used, for example, in an action to create one-off measurements from the probes which fail expectations.

```
actions:
  RunRIPEAtlasTraceroute:
    descr: Create new traceroute msm from the probe which missed expectations
    kind: run
    path: ripe-atlas
    args:
    - measure
    - traceroute
    - --target
    - www.example.com
    - --no-report
    - --from-probes
    - $ProbeID
```

2.6 Monitor configuration syntax

Contents

2.6.1 Monitor

A monitor allows to process results from a measurement.

Configuration fields:

- `descr` (optional): monitor's brief description.
- `measurement-id` (optional): measurement ID used to gather results. It can be given (and/or overwritten) via command line argument `--measurement-id`.
- `matching_rules`: list of rules to match probes against. When a probe matches one of these rules, its expected results are processed and its actions are performed.
- `expected_results` (optional): list of expected results. Probe's expected results contain references to this list.
- `actions` (optional): list of actions to be executed on the basis of probe's expected results.
- `stream` (optional): boolean indicating if results streaming must be used. It can be given (and/or overwritten) via command line argument `--stream`.
- `stream_timeout` (optional): how long to wait (in seconds) before stopping a streaming monitor if no results are received on the stream.

- `key` (optional): RIPE Atlas key to access the measurement. It can be given (and/or overwritten) via command line argument `--key`.
- `key_file` (optional): a file containing the RIPE Atlas key to access the measurement. The file must contain only the RIPE Atlas key, in plain text. If `key` is given, this field is ignored.

2.6.2 Rule

Probes which produced the results fetched from the measurement are matched against these rules to determine whether those results must be processed or not.

Configuration fields:

- `descr` (optional): a brief description of the rule.
- `process_next` (optional): determine whether the rule following the current one has to be elaborated or nor. More details on the description below.
- `src_country` (optional): list of two letters country ISO codes.
- `src_as` (optional): list of Autonomous System numbers.
- `probe_id` (optional): list of probes' IDs.
- `internal_labels` (optional): list of internal labels. More details on the description below.
- `reverse` (optional): boolean, indicating if the aforementioned criteria identify probes which have to be exluded from the matching.
- `expected_results` (optional): list of expected results' names which have to be processed on match. Must be one or more of the expected results defined in Monitor.``expected_results``. If empty or missing, the rule will be treated as if a match occurred and its actions are performed.
- `actions` (optional): list of actions' names which have to be perormed for matching probes. Must be one or more of the actions defined in Monitor.``actions``.

The `src_country` criterion matches when probe's source country is one of the country ISO codes given in the list.

The `src_as` criterion matches when probe's source AS is one of the ASN given in the list. Since RIPE Atlas defines two ASs for each probe (ASN_v4 and ASN_v6) the one corresponding to the measurement's address family is taken into account.

The `probe_id` criterion matches when probe's ID is one of the IDs given in the list.

The `internal_labels` criterion matches when a probe has been previously tagged with a label falling in the given list. See the `label` Action for more details.

A probe matches the rule when all the given criteria are satisfied or when no criteria are defined at all. If `reverse` is True, a probe matches when none of the criteria is satisfied.

When a probe matches the rule, the expected results given in `expected_results` are processed; actions given in the `actions` list are performed on the basis of expected results processing output. If no `expected_results` are given, actions will be performed too.

When a probe matches the current rule's criteria:

- if `process_next` is True, the rule which follows the current one is forcedly elaborated;
- if `process_next` if False or missing, the rules processing is stopped.

If a probe does not match the current rule's criteria:

- if `process_next` is False, the rule processing is forcedly stopped;
- if `process_next` is True or missing, the rule which follows the current one is regularly processed.

Examples:

```
matching_rules:
- descr: Do not process results for probe ID 123 and 456
  probe_id:
  - 123
  - 456
- descr: Check dst AS for any probe, errors to NOC; process next rule
  expected_results: DstAS
  actions: SendEMailToNOC
  process_next: True
- descr: Italian probes must reach target via AS64496
  src_country: IT
  expected_results: ViaAS64496
  actions: LogErrors
- descr: German and French probes must reach target with low RTT
  src_country:
  - DE
  - FR
  expected_results: LowRTT
  actions: LogErrors

matching_rules:
- descr: Set 'VIP' (Very Important Probe) label to ID 123 and 456
  probe_id:
  - 123
  - 456
  process_next: True
  actions: SetVIPLabel
- descr: Set 'VIP' label to Italian probes too
  src_country: IT
  process_next: True
  actions: SetVIPLabel
- descr: VIPs must have low RTT
  internal_labels: VIP
  expected_results: LowRTT
```

2.6.3 Expected result

A group of criteria used to match probes' results.

Configuration fields:

- `descr` (optional): a brief description of this group of criteria.

Matching rules reference this on their `expected_results` list.

When a probe matches a rule, the keys in the `expected_results` list of that rule are used to obtain the group of criteria to be used to process the result.

Example:

```
matching_rules:
- descr: Probes from France via AS64496
  src_country: FR
  expected_results: ViaAS64496
expected_results:
  ViaAS64496:
    upstream_as: 64496
```

Common criteria

Criterion: rtt

Test the median round trip time toward destination.

Available for:

- ping
- traceroute

Configuration fields:

- `rtt`: maximum RTT (in ms).
- `rtt_tolerance` (optional): tolerance (in %) on `rtt`.

If `rtt_tolerance` is not given, match when measured RTT is less than `rtt`, otherwise match when measured RTT is within `rtt` +/- `rtt_tolerance` %.

Examples:

```
expected_results:
  LowRTT:
    rtt: 50
  Near150:
    rtt: 150
    rtt_tolerance: 30
```

Criterion: dst_responded

Verify if destination responded.

Available for:

- traceroute
- ping
- sslcert

Configuration fields:

- `dst_responded`: boolean indicating if the destination is expected to be responding or not.

For ping, a destination is responding if a probe received at least one reply packet. For sslcert, a destination is responding if at least one certificate is received by the probe.

If `dst_responded` is True, match when a destination is responding. If `dst_responded` is False, match when a destination is not responding.

Example:

```
expected_results:
  DestinationReachable:
    dst_responded: True
```

Criterion: dst_ip

Verify that the destination IP used by the probe for the measurement is the expected one.

Available for:

- traceroute
- ping
- sslcert

Configuration fields:

- `dst_ip`: list of expected IP addresses (or prefixes).

Match when the probe destination IP is one of the expected ones (or falls within one of the expected prefixes).

Examples:

```
dst_ip: 192.168.0.1

dst_ip:
- 192.168.0.1
- 2001:DB8::1

dst_ip:
- 192.168.0.1
- 10.0.0.0/8
- 2001:DB8::/32
```

Traceroute criteria

Criterion: dst_as

Verify the traceroute destination's AS number.

Available for:

- traceroute

Configuration fields:

- `dst_as`: list of Autonomous System numbers.

It builds the path of ASs traversed by the traceroute. Match when the last AS in the path is one of the expected ones.

Examples:

```
dst_as:
- 64496

dst_as:
- 64496
- 65551
```

Criterion: as_path

Verify the path of ASs traversed by a traceroute.

Available for:

- traceroute

Configuration fields:

- `as_path`: list of Autonomous System path.

An AS path is made of AS numbers separated by white spaces. It can contain two special tokens:

- "S", that is expanded with the probe's source AS number;
- "IX", that represents an Internet Exchange Point peering network for those IXPs which don't announce their peering prefixes via BGP.

The "IX" token is meagniful only if the `ip_cache.use_ixps_info` global configuration parameter is True.

It builds the path of ASs traversed by the traceroute. Match when the AS path or a contiguous part of it is one of the expected ones.

Examples:

```
as_path: 64496 64497

as_path:
- 64496 64497
- 64498 64499 64500

as_path:
- S 64496 64497

as_path:
- S IX 64500
```

Criterion: upstream_as

Verify the traceroute destination upstream's AS number.

Available for:

- traceroute

Configuration fields:

- `upstream_as`: list of Autonomous System numbers.

It builds the path of ASs traversed by the traceroute. Match when the penultimate AS in the path is one of the expected ones.

Examples:

```
upstream_as:
- 64496

upstream_as:
- 64496
- 64497
```

SSL criteria

Criterion: cert_fp

Verify SSL certificates' fingerprints.

Available for:

- sslcert

Configuration fields:

- `cert_fp`: list of certificates' SHA256 fingerprints or SHA256 fingerprints of the chain.

A fingerprint must be in the format 12:34:AB:CD:EF:... 32 blocks of 2 characters hex values separated by colon (":").

The `cert_fp` parameter can contain stand-alone fingerprints or bundle of fingerprints in the format "fingerprint1,fingerprint2,fingerprintN".

A result matches if any of its certificates' fingerprint is in the list of stand-alone expected fingerprints or if the full chain fingerprints is in the list of bundle fingerprints.

Examples:

```
expected_results:
  MatchLeafCertificate:
    cert_fp: 01:02:[...]:31:32
  MatchLeacCertificates:
    cert_fp:
    - 01:02:[...]:31:32
    - 12:34:[...]:CD:EF
  MatchLeafOrChain:
    cert_fp:
    - 01:02:[...]:31:32
    - 12:34:[...]:CD:EF,56:78:[...]:AB:CD
```

DNS criteria

Criterion: dns_rcode

Verify if DNS responses received by a probe have the expected rcode.

Available for:

- dns

Configuration fields:

- `dns_rcode`: list of expected DNS rcodes ("NOERROR", "FORMERR", "SERVFAIL", "NXDOMAIN", "NOTIMP", "REFUSED", "YXDOMAIN", "YXRRSET", "NXRRSET", "NOTAUTH", "NOTZONE", "BADVERS").

Match when all the responses received by a probe have one of the expected rcodes listed in `dns_rcode`.

Example:

```
expected_results:
  DNS_NoError_or_NXDomain:
    dns_rcode:
    - "NOERROR"
    - "NXDOMAIN"
```

Criterion: dns_flags

Verify if DNS responses received by a probe have the expected headers flags on.

Available for:

- dns

Configuration fields:

- `dns_flags`: list of expected DNS flag (“aa”, “ad”, “cd”, “qr”, “ra”, “rd”).

Match when all the responses received by a probe have all the expected flags on.

Example:

```
expected_results:
  AA_and_AD:
    dns_flags:
    - aa
    - ad
```

Criterion: edns

Verify EDNS extension of DNS responses received by probes.

Available for:

- dns

Configuration fields:

- `edns`: boolean indicating whether EDNS support is expected or not.
- `edns_size` (optional): minimum expected size.
- `edns_do` (optional): boolean indicating the expected presence of DO flag.
- `edns_nsid` (optional): list of expected NSID values.

The optional parameters are taken into account only when `edns` is True.

If `edns` is True, match when all the responses contain EDNS extension, otherwise when all the responses do not contain it. If `edns_size` is given, the size reported must be >= than the expected one. If `edns_do` is given, all the responses must have (or have not) the DO flag on. If `edns_nsid` is given, all the responses must contain and EDNS NSID option which falls within the list of values herein specified.

Examples:

```
edns: true

edns: true
edns_do: true

edns: true
edns_nsid:
- "ods01.l.root-servers.org"
- "kbp01.l.root-servers.org"
```

Criterion: dns_answers

Verify if the responses received by a probe contain the expected records.

Available for:

- dns

Configuration fields:

- `dns_answers`: one or more sections where records are searched on. Must be one of "answers", "authorities", "additionals".

Each section must contain a list of records.

Match when all the responses received by a probe contain at least one record matching the expected ones in each of the given sections.

Example:

```
dns_answers:
    answers:
        - <record1>
        - <record2>
    authorities:
        - <record3>
        - <record4>
```

DNS record Test properties which are common to all DNS record types.

Configuration fields:

- `type`: record's type. Must be one of the DNS record types implemented and described below.
- `name` (optional): list of expected names.
- `ttl_min` (optional): minimum TTL that is expected for the record.
- `ttl_max` (optional): maximum TTL that is expected for the record.
- `class` (optional): expected class for the record.

Match when all the defined criteria are met:

- record name must be within the list of given names (`name`);
- record TTL must be >= `ttl_min` and <= `ttl_max`;
- record class must be equal to `class`.

On the basis of record's `type`, further parameters may be needed.

Example:

```
dns_answers:
    answers:
        - type: A
          name: www.ripe.net.
          address: 193.0.6.139
        - type: AAAA
          name:
          - www.ripe.net.
          - ripe.net.
          ttl_min: 604800
          address: 2001:67c:2e8:22::c100:0/64
```

A record Verify if record's type is A and if received address match the expectations.

Configuration fields:

- `address`: list of IPv4 addresses (or IPv4 prefixes).

Match when record's type is A and resolved address is one of the given addresses (or falls within one of the given prefixes).

AAAA record Verify if record's type is AAAA and if received address match the expectations.

Configuration fields:

- `address`: list of IPv6 addresses (or IPv6 prefixes).

Match when record's type is AAAA and resolved address is one of the given addresses (or falls within one of the given prefixes).

NS record Verify if record's type is NS and if target is one of the expected ones.

Configuration fields:

- `target`: list of expected targets.

Match when record's type is NS and received target is one of those given in `target`.

CNAME record Verify if record's type is CNAME and if target is one of the expected ones.

Configuration fields:

- `target`: list of expected targets.

Match when record's type is CNAME and received target is one of those given in `target`.

2.6.4 Action

Action performed on the basis of expected results processing for probes which match the `matching_rules` rules.

Configuration fields:

- `kind`: type of action.
- `descr` (optional): brief description of the action.
- `when` (optional): when the action must be performed (with regards of expected results processing output); one of "on_match", "on_mismatch", "always". Default: "on_mismatch".

When a probe matches a rule, it's expected results are processed; on the basis of the output, actions given in the rule's `actions` list are performed. For each expected result, if the probe's collected result matches the expectation actions whose `when` = "on_match" or "always" are performed. If the collected result does not match the expected result, actions whose `when` = "on_mismatch" or "always" are performed.

Action log

Log the match/mismatch along with the collected result.

No parameters required.

Action email

Send an email with the expected result processing output.

Configuration fields:

- `from_addr` (optional): email address used in the From field.
- `to_addr` (optional): email address used in the To field.
- `subject` (optional): subject of the email message.

- `smtp_host` (optional): SMTP server's host.
- `smtp_port` (optional): SMTP server's port.
- `use_ssl` (optional): boolean indicating whether the connection toward SMTP server must use encryption.
- `username` (optional): username for SMTP authentication.
- `password` (optional): password for SMTP authentication.
- `timeout` (optional): timeout, in seconds.

Parameters which are not given are read from the global configuration file `default_smtp` section.

Action run

Run an external program.

Configuration fields:

- `path`: path of the program to run.
- `env_prefix` (optional): prefix used to build environment variables.
- `args` (optional): list of arguments which have to be passed to the program. If the argument starts with "$" it is replaced with the value of the variable with the same name.

If `env_prefix` is not given, it's value is taken from the global configuration file `misc.env_prefix` parameter.

Variables are:

- `ResultMatches`: True, False or None
- `MsmID`: measurement's ID
- `MsmType`: measurement's type (ping, traceroute, sslcert, dns)
- `MsmAF`: measurement's address family (4, 6)
- `MsmStatus`: measurement's status (Running, Stopped) [https://atlas.ripe.net/docs/rest/]
- `MsmStatusID`: measurement's status ID [https://atlas.ripe.net/docs/rest/]
- `Stream`: True or False
- `ProbeID`: probe's ID
- `ProbeCC`: probe's ISO Country Code
- `ProbeASNv4`: probe's ASN (IPv4)
- `ProbeASNv6`: probe's ASN (IPv6)
- `ProbeASN`: probe's ASN related to measurement's address family
- `ResultCreated`: timestamp of result's creation date/time

Example:

```
actions:
  RunMyProgram:
    kind: run
    path: /path/to/my-program
    args:
    - command
    - -o
    - --msm
    - $MsmID
    - --probe
    - $ProbeID
```

Action syslog

Log the match/mismatch along with the collected result using syslog.

Configuration fields:

- `socket` (optional): where the syslog message has to be logged. One of "file", "udp", "tcp".
- `host` (optional): meaningful only when `socket` is "udp" or "tcp". Host where send the syslog message to.
- `port` (optional): meaningful only when `socket` is "udp" or "tcp". UDP/TCP port where send the syslog message to.
- `file` (optional): meaningful only when `socket` is "file". File where the syslog message has to be written to.
- `facility` (optional): syslog facility that must be used to log the message.
- `priority` (optional): syslog priority that must be used to log the message.

Parameters which are not given are read from the global configuration file `default_syslog` section.

Action label

Add or remove custom labels to/from probes.

Configuration fields:

- `op`: operation; one of "add" or "del".
- `label_name`: label to be added/removed.
- `scope` (optional): scope of the label; one of "result" or "probe". Default: "result".

Labels can be added to probes and subsequently used to match those probes in other rules (`internal_labels` criterion).

If scope is "result", the operation is significative only within the current result processing (that is, within the current `matching_rules` processing for the current result). Labels added to probe are removed when the current result processing is completed.

If scope is "probe", the operation is persistent across results processing.

2.7 How to contribute

Here a brief guide to contributing to this tool:

- fork it on GitHub and create your own repository;
- install it using the "editable" installation or clone it locally on your machine (virtualenv usage is strongly suggested);

```
$ # pip "editable" installation
$ pip install -e git+https://github.com/YOUR_USERNAME/ripe-atlas-monitor.git#egg=ripe-atlas-moni

$ # manual cloning from GitHub (you have to care about dependencies)
```

```
    $ git clone https://github.com/YOUR_USERNAME/ripe-atlas-monitor.git
    $ export PYTHONPATH="/path/to/your/ripe-atlas-monitor"
```

- run the tests in order to be sure that everything is fine;

```
    $ nosetests -vs
```

- finally, start making your changes and, possibly, add test units and docs to make the merging process easier.

Once you have done, please run tests again:

```
$ tox
```

If everything is fine, push to your fork and create a pull request.

2.7.1 Code

Adding a new check

- **ParsedResults.py**: a new property must be handled by a `ParsedResult` class in order to parse and prepare the new result to be checked. Create a new class (if needed) and add a new property (`PROPERTIES` and `@property`). The `prepare()` method must call `self.set_attr_to_cache()` to store the parsed values; the `@property` must read the value using `self.get_attr_from_cache()` and return it. More info on the `ParsedResult` class docstring.
- **ExpResCriteriaXXX.py**: an `ExpResCriterion`-derived class must implement the `__init__()` method to read the expected result attributes from the monitor's configuration and validate them. The `prepare()` method must parse the result received from probes and store its value in a local attribute (something like `self.response_xxx`); the `result_matches()` method must compare the parsed result received from the probe with the expected result. The new class must be added to the appropriate list in **ExpResCriteria.py**. More info on the `ExpResCriterion` class docstring (**ExpResCriteriaBase.py**). See also *ExpResCriterion-derived classes docstring*.
- **tests/monitor_cfg_test.py**:
  - Add the new criterion to `self.criteria` in `TestMonitorCfg.setUp()` (at least the `CRITERION_NAME` with a correct value).
  - Add a call to `self.add_criterion("<criterion_name>")` to the `test_expres_XXX()` methods, where *XXX* is the measurements' type this expected result applies to.
  - Add some tests for the new expected result configuration syntax.
- **tests/doc_test.py**: if the new expected result contains lists attributes, add the configuration field name to the appropriate `exp_list_fields` variables.
- **tests/results_XXX_test.py**: add some tests to validate how the new criterion processes results from real measurements.

Adding a new report to the analyzer

- **ParsedResults.py**: a new class must be created (or a new property added to an existing `ParsedResult`-derived class). See the previous section.
- **Analyzer.py**:
  - This step can be avoided if the new `ParsedResult` property to analyze is handled by an already existing `ParsedResult` class.

Create a new `BaseResultsAnalyzer`-derived class; its `get_parsed_results()` method must yield each `ParsedResult` element that need to be analyzed (for example, the result itself or each DNS response for DNS measurements' results). The `BaseResultsAnalyzer` and `ResultsAnalyzer_DNSBased` classes should already do most of the work. Add the new class to the `Analyzer.RESULTS_ANALYZERS` list.

- The `BaseResultsAnalyzer.PROPERTIES_ANALYZERS_CLASSES` and `BaseResultsAnalyzer.PROPERTIES_ORDER` lists must contain the new property defined in the `ParsedResult`-derived class.
- A `BasePropertyAnalyzer`-derived class must be created in order to implement the aggregation mechanism and the output formatting for the new property. More info on the `BasePropertyAnalyzer` and `BaseResultsAnalyzer` classes docstring.

• **docs/COMMANDS.rst**: add the new property to the list of those supported by the `analyze` command.

Data for unit testing

The **tests/data/download_msm.py** script can be used to download and store data used for tests. It downloads measurement's metadata, results and probes' information and stores them in a **measurement_id**.json file. The **tests/data.py** module loads the JSON files that can subsequently be used for unit testing purposes.

ExpResCriterion-derived classes docstring

These classes must use a special syntax in their docstrings which allows to automatically build documentation and config example (**doc.py** `build_doc` and `build_monitor_cfg_tpl` functions).

Example:

```
Criterion: rtt

    Test the median round trip time toward destination.

    Available for: ping, traceroute.

    `rtt`: maximum RTT (in ms).

    `rtt_tolerance` (optional): tolerance (in %) on `rtt`.

    If `rtt_tolerance` is not given, match when measured RTT is less
    than `rtt`, otherwise match when measured RTT is within `rtt`
    +/- `rtt_tolerance` %.

    Examples:

    expected_results:
      LowRTT:
        rtt: 50
      Near150:
        rtt: 150
        rtt_tolerance: 30
```

• The first line must include only the "Criterion: xxx" string, where *xxx* is the class `CRITERION_NAME` attribute.

  Example: `Criterion:  rtt`

• A brief description of the expected result must follow.

  Example: `Test the median round trip time toward destination.`

- The list of measurements' types for which this expected result can be used must follow, in the format `Available for: x[, y[, z]].`, where values are valid measurements' types (`ping`, `traceroute`, ...).

  Example: `Available for: ping, traceroute.`

- A list of configuration fields must follow. Every docstring line starting with a backquote is considered to be a field name.

  The format must be the following:

  ``` `field_name` ["(optional)"]: ["list"] "description..." ```

  The "(optional)" string is used to declare this field as optional, otherwise it's considered mandatory.

  The "list" string is used to declare that this field contains a list of values.

  Example: `` `rtt`: maximum RTT (in ms). ``, `` `rtt_tolerance` (optional): tolerance (in %) on `rtt`. ``, `` `dst_ip`: list of expected IP addresses (or prefixes). ``

- A (long) description of how this expected result's fields are used can follow. Here, be careful to avoid lines starting with the backquote, otherwise they will be interpreted as a field declaration.

- Finally, a line starting with the "Example" or "Examples" strings can be used to show some examples. They will be formatted using code blocks.

2.8 Changelog

2.8.1 0.1.10

- fix packaging issue

2.8.2 0.1.9

improvements

- `analyze` command, add the `--show-all-dns-answers` argument

2.8.3 0.1.8

improvements

- `dst_responded` criterion can be used for SSL measurements too (and is considered in analysis reports too)

fixes

- cosmetic
  ```

2.8.4 0.1.7

improvements

- `analyze` command:
  - JSON output
  - show unique probes count for DNS measurements
- new check and analysis of DNS RCODEs

fixes

- `analyze` command, DNS answers analysis for records with no name and no type
- bug in IP addresses cache

2.8.5 0.1.6

improvements

- new checks and analysis for EDNS NSID option
- DNS answers analysis
- probes filter for `run` and `analyze` commands now accepts probe IDs and country codes

2.8.6 0.1.5

improvements

- more options for the `analyze` command:
  - show probes (up to 3) beside results
  - destination AS and upstream AS results
  - show common sequences/patterns among results
- add `--probes` argument to `run` and `analyze` commands to filter results
- email logging of error messages

fixes

- fix empty resultset handling in `analyze` cmd

2.8.7 0.1.4

new features

- Python 3.4 support

improvements

- `-m` argument for `analyze` command, to gather msm id and auth key from the monitor itself
- `--dont-wait` argument for `run` command

fixes

- error handling for null RTT results in `analyze` command

2.8.8 0.1.3

improvements

- better RTT results formatting in `analyze` command
- no stdout logging when used in `daemonize` mode

fixes

- error handling for IXPs networks info unavailability

2.8.9 0.1.2

new features

- `analyze` command to show elaborated results from a measurement
- bash autocomplete

fixes

- continous monitors didn't run continously

2.8.10 0.1.1

improvements

- better results and actions logging

2.8.11 0.1.0

First release (beta)

CHAPTER 3

Status

This tool is currently in **beta**: some field tests have been done but it needs to be tested deeply and on more scenarios.

Moreover, contributions (fixes to code and to grammatical errors, typos, new features) are very much appreciated. More details on the contributing guide.

CHAPTER 4

Bug? Issues?

But also suggestions? New ideas?

Please create an issue on GitHub at https://github.com/pierky/ripe-atlas-monitor/issues

CHAPTER 5

Author

Pier Carlo Chiodi - https://pierky.com

Blog: https://blog.pierky.com Twitter: @pierky