ElastAlert Documentation

Release 0.0.1

Quentin Long

Aug 18, 2017

Contents

Contents:

CHAPTER 1

ElastAlert - Easy & Flexible Alerting With Elasticsearch

ElastAlert is a simple framework for alerting on anomalies, spikes, or other patterns of interest from data in Elasticsearch.

At Yelp, we use Elasticsearch, Logstash and Kibana for managing our ever increasing amount of data and logs. Kibana is great for visualizing and querying data, but we quickly realized that it needed a companion tool for alerting on inconsistencies in our data. Out of this need, ElastAlert was created.

If you have data being written into Elasticsearch in near real time and want to be alerted when that data matches certain patterns, ElastAlert is the tool for you.

Overview

We designed ElastAlert to be *reliable*, highly *modular*, and easy to *set up* and *configure*.

It works by combining Elasticsearch with two types of components, rule types and alerts. Elasticsearch is periodically queried and the data is passed to the rule type, which determines when a match is found. When a match occurs, it is given to one or more alerts, which take action based on the match.

This is configured by a set of rules, each of which defines a query, a rule type, and a set of alerts.

Several rule types with common monitoring paradigms are included with ElastAlert:

- "Match where there are X events in Y time" (`frequency` type)
- "Match when the rate of events increases or decreases" (`spike` type)
- "Match when there are less than X events in Y time" (`flatline` type)
- "Match when a certain field matches a blacklist/whitelist" (`blacklist` and `whitelist` type)
- "Match on any event matching a given filter" (`any` type)
- "Match when a field has two different values within some time" (`change` type)

Currently, we have support built in for these alert types:

- Command

- Email
- JIRA
- OpsGenie
- SNS
- HipChat
- Slack
- Telegram
- Debug
- Stomp

Additional rule types and alerts can be easily imported or written. (See *Writing rule types* and *Writing alerts*)

In addition to this basic usage, there are many other features that make alerts more useful:

- Alerts link to Kibana dashboards
- Aggregate counts for arbitrary fields
- Combine alerts into periodic reports
- Separate alerts by using a unique key field
- Intercept and enhance match data

To get started, check out *Running ElastAlert For The First Time*.

Reliability

ElastAlert has several features to make it more reliable in the event of restarts or Elasticsearch unavailability:

- ElastAlert *saves its state to Elasticsearch* and, when started, will resume where previously stopped
- If Elasticsearch is unresponsive, ElastAlert will wait until it recovers before continuing
- Alerts which throw errors may be automatically retried for a period of time

Modularity

ElastAlert has three main components that may be imported as a module or customized:

Rule types

The rule type is responsible for processing the data returned from Elasticsearch. It is initialized with the rule configuration, passed data that is returned from querying Elasticsearch with the rule's filters, and outputs matches based on this data. See *Writing rule types* for more information.

Alerts

Alerts are responsible for taking action based on a match. A match is generally a dictionary containing values from a document in Elasticsearch, but may contain arbitrary data added by the rule type. See *Writing alerts* for more information.

Enhancements

Enhancements are a way of intercepting an alert and modifying or enhancing it in some way. They are passed the match dictionary before it is given to the alerter. See *Enhancements* for more information.

Configuration

ElastAlert has a global configuration file, `config.yaml`, which defines several aspects of its operation:

`buffer_time`: ElastAlert will continuously query against a window from the present to `buffer_time` ago. This way, logs can be back filled up to a certain extent and ElastAlert will still process the events. This may be overridden by individual rules. This option is ignored for rules where `use_count_query` or `use_terms_query` is set to true. Note that back filled data may not always trigger count based alerts as if it was queried in real time.

`es_host`: The host name of the Elasticsearch cluster where ElastAlert records metadata about its searches. When ElastAlert is started, it will query for information about the time that it was last run. This way, even if ElastAlert is stopped and restarted, it will never miss data or look at the same events twice. It will also specify the default cluster for each rule to run on. The environment variable `ES_HOST` will override this field.

`es_port`: The port corresponding to `es_host`. The environment variable `ES_PORT` will override this field.

`use_ssl`: Optional; whether or not to connect to `es_host` using TLS; set to `True` or `False`. The environment variable `ES_USE_SSL` will override this field.

`verify_certs`: Optional; whether or not to verify TLS certificates; set to `True` or `False`. The default is `True`.

`client_cert`: Optional; path to a PEM certificate to use as the client certificate.

`client_key`: Optional; path to a private key file to use as the client key.

`ca_certs`: Optional; path to a CA cert bundle to use to verify SSL connections

`es_username`: Optional; basic-auth username for connecting to `es_host`. The environment variable `ES_USERNAME` will override this field.

`es_password`: Optional; basic-auth password for connecting to `es_host`. The environment variable `ES_PASSWORD` will override this field.

`es_url_prefix`: Optional; URL prefix for the Elasticsearch endpoint.

`es_send_get_body_as`: Optional; Method for querying Elasticsearch - `GET`, `POST` or `source`. The default is `GET`

`es_conn_timeout`: Optional; sets timeout for connecting to and reading from `es_host`; defaults to `10`.

`rules_folder`: The name of the folder which contains rule configuration files. ElastAlert will load all files in this folder, and all subdirectories, that end in .yaml. If the contents of this folder change, ElastAlert will load, reload or remove rules based on their respective config files.

`scan_subdirectories`: Optional; Sets whether or not ElastAlert should recursively descend the rules directory - `true` or `false`. The default is `true`

`run_every`: How often ElastAlert should query Elasticsearch. ElastAlert will remember the last time it ran the query for a given rule, and periodically query from that time until the present. The format of this field is a nested unit of time, such as `minutes: 5`. This is how time is defined in every ElastAlert configuration.

`writeback_index`: The index on `es_host` to use.

`max_query_size`: The maximum number of documents that will be downloaded from Elasticsearch in a single query. The default is 10,000, and if you expect to get near this number, consider using `use_count_query` for the

rule. If this limit is reached, ElastAlert will scroll through pages the size of `max_query_size` until processing all results.

`scroll_keepalive`: The maximum time (formatted in Time Units) the scrolling context should be kept alive. Avoid using high values as it abuses resources in Elasticsearch, but be mindful to allow sufficient time to finish processing all the results.

`max_aggregation`: The maximum number of alerts to aggregate together. If a rule has `aggregation` set, all alerts occuring within a timeframe will be sent together. The default is 10,000.

`old_query_limit`: The maximum time between queries for ElastAlert to start at the most recently run query. When ElastAlert starts, for each rule, it will search `elastalert_metadata` for the most recently run query and start from that time, unless it is older than `old_query_limit`, in which case it will start from the present time. The default is one week.

`disable_rules_on_error`: If true, ElastAlert will disable rules which throw uncaught (not EAException) exceptions. It will upload a traceback message to `elastalert_metadata` and if `notify_email` is set, send an email notification. The rule will no longer be run until either ElastAlert restarts or the rule file has been modified. This defaults to True.

`notify_email`: An email address, or list of email addresses, to which notification emails will be sent. Currently, only an uncaught exception will send a notification email. The from address, SMTP host, and reply-to header can be set using `from_addr`, `smtp_host`, and `email_reply_to` options, respectively. By default, no emails will be sent.

`from_addr`: The address to use as the from header in email notifications. This value will be used for email alerts as well, unless overwritten in the rule config. The default value is "ElastAlert".

`smtp_host`: The SMTP host used to send email notifications. This value will be used for email alerts as well, unless overwritten in the rule config. The default is "localhost".

`email_reply_to`: This sets the Reply-To header in emails. The default is the recipient address.

`aws_region`: This makes ElastAlert to sign HTTP requests when using Amazon Elasticsearch Service. It'll use instance role keys to sign the requests. The environment variable `AWS_DEFAULT_REGION` will override this field.

`boto_profile`: Deprecated! Boto profile to use when signing requests to Amazon Elasticsearch Service, if you don't want to use the instance role keys.

`profile`: AWS profile to use when signing requests to Amazon Elasticsearch Service, if you don't want to use the instance role keys. The environment variable `AWS_DEFAULT_PROFILE` will override this field.

`replace_dots_in_field_names`: If `True`, ElastAlert replaces any dots in field names with an underscore before writing documents to Elasticsearch. The default value is `False`. Elasticsearch 2.0 - 2.3 does not support dots in field names.

`string_multi_field_name`: If set, the suffix to use for the subfield for string multi-fields in Elasticsearch. The default value is `.raw` for Elasticsearch 2 and `.keyword` for Elasticsearch 5.

Running ElastAlert

```
$ python elastalert/elastalert.py
```

Several arguments are available when running ElastAlert:

`--config` will specify the configuration file to use. The default is `config.yaml`.

`--debug` will run ElastAlert in debug mode. This will increase the logging verboseness, change all alerts to `DebugAlerter`, which prints alerts and suppresses their normal action, and skips writing search and alert metadata back to Elasticsearch.

`--start <timestamp>` will force ElastAlert to begin querying from the given time, instead of the default, querying from the present. The timestamp should be ISO8601, e.g. `YYYY-MM-DDTHH:MM:SS` (UTC) or with timezone `YYYY-MM-DDTHH:MM:SS-08:00` (PST). Note that if querying over a large date range, no alerts will be sent until that rule has finished querying over the entire time period. To force querying from the current time, use "NOW".

`--end <timestamp>` will cause ElastAlert to stop querying at the specified timestamp. By default, ElastAlert will periodically query until the present indefinitely.

`--rule <rule.yaml>` will only run the given rule. The rule file may be a complete file path or a filename in `rules_folder` or its subdirectories.

`--silence <unit>=<number>` will silence the alerts for a given rule for a period of time. The rule must be specified using `--rule`. <unit> is one of days, weeks, hours, minutes or seconds. <number> is an integer. For example, `--rule noisy_rule.yaml --silence hours=4` will stop noisy_rule from generating any alerts for 4 hours.

`--verbose` will increase the logging verboseness, which allows you to see information about the state of queries.

`--es_debug` will enable logging for all queries made to Elasticsearch.

`--es_debug_trace` will enable logging curl commands for all queries made to Elasticsearch to a file.

`--end <timestamp>` will force ElastAlert to stop querying after the given time, instead of the default, querying to the present time. This really only makes sense when running standalone. The timestamp is formatted as `YYYY-MM-DDTHH:MM:SS` (UTC) or with timezone `YYYY-MM-DDTHH:MM:SS-XX:00` (UTC-XX).

`--pin_rules` will stop ElastAlert from loading, reloading or removing rules based on changes to their config files.

CHAPTER 2

Running ElastAlert for the First Time

Requirements

- Elasticsearch
- ISO8601 or Unix timestamped data
- Python 2.7
- pip, see requirements.txt
- Packages on Ubuntu 14.x: python-pip python-dev libffi-dev libssl-dev

Downloading and Configuring

You can either install the latest released version of ElastAlert using pip:

```
$ pip install elastalert
```

or you can clone the ElastAlert repository for the most recent changes:

```
$ git clone https://github.com/Yelp/elastalert.git
```

Install the module:

```
$ pip install "setuptools>=11.3"
$ python setup.py install
```

Depending on the version of Elasticsearch, you may need to manually install the correct version of elasticsearch-py.

Elasticsearch 5.0+:

```
$ pip install "elasticsearch>=5.0.0"
```

Elasticsearch 2.X:

```
$ pip install "elasticsearch<3.0.0"
```

Next, open up config.yaml.example. In it, you will find several configuration options. ElastAlert may be run without changing any of these settings.

`rules_folder` is where ElastAlert will load rule configuration files from. It will attempt to load every .yaml file in the folder. Without any valid rules, ElastAlert will not start. ElastAlert will also load new rules, stop running missing rules, and restart modified rules as the files in this folder change. For this tutorial, we will use the example_rules folder.

`run_every` is how often ElastAlert will query Elasticsearch.

`buffer_time` is the size of the query window, stretching backwards from the time each query is run. This value is ignored for rules where `use_count_query` or `use_terms_query` is set to true.

`es_host` is the address of an Elasticsearch cluster where ElastAlert will store data about its state, queries run, alerts, and errors. Each rule may also use a different Elasticsearch host to query against.

`es_port` is the port corresponding to `es_host`.

`use_ssl`: Optional; whether or not to connect to `es_host` using TLS; set to `True` or `False`.

`verify_certs`: Optional; whether or not to verify TLS certificates; set to `True` or `False`. The default is `True`

`client_cert`: Optional; path to a PEM certificate to use as the client certificate

`client_key`: Optional; path to a private key file to use as the client key

`ca_certs`: Optional; path to a CA cert bundle to use to verify SSL connections

`es_username`: Optional; basic-auth username for connecting to `es_host`.

`es_password`: Optional; basic-auth password for connecting to `es_host`.

`es_url_prefix`: Optional; URL prefix for the Elasticsearch endpoint.

`es_send_get_body_as`: Optional; Method for querying Elasticsearch - `GET`, `POST` or `source`. The default is `GET`

`writeback_index` is the name of the index in which ElastAlert will store data. We will create this index later.

`alert_time_limit` is the retry window for failed alerts.

Save the file as `config.yaml`

Setting Up Elasticsearch

ElastAlert saves information and metadata about its queries and its alerts back to Elasticsearch. This is useful for auditing, debugging, and it allows ElastAlert to restart and resume exactly where it left off. This is not required for ElastAlert to run, but highly recommended.

First, we need to create an index for ElastAlert to write to by running `elastalert-create-index` and following the instructions:

```
$ elastalert-create-index
New index name (Default elastalert_status)
Name of existing index to copy (Default None)
New index elastalert_status created
Done!
```

For information about what data will go here, see *ElastAlert Metadata Index*.

Creating a Rule

Each rule defines a query to perform, parameters on what triggers a match, and a list of alerts to fire for each match. We are going to use `example_rules/example_frequency.yaml` as a template:

```
# From example_rules/example_frequency.yaml
es_host: elasticsearch.example.com
es_port: 14900
name: Example rule
type: frequency
index: logstash-*
num_events: 50
timeframe:
    hours: 4
filter:
- term:
    some_field: "some_value"
alert:
- "email"
email:
- "elastalert@example.com"
```

`es_host` and `es_port` should point to the Elasticsearch cluster we want to query.

`name` is the unique name for this rule. ElastAlert will not start if two rules share the same name.

`type`: Each rule has a different type which may take different parameters. The `frequency` type means "Alert when more than `num_events` occur within `timeframe`." For information other types, see *Rule types*.

`index`: The name of the index(es) to query. If you are using Logstash, by default the indexes will match `"logstash-*"`.

`num_events`: This parameter is specific to `frequency` type and is the threshold for when an alert is triggered.

`timeframe` is the time period in which `num_events` must occur.

`filter` is a list of Elasticsearch filters that are used to filter results. Here we have a single term filter for documents with `some_field` matching `some_value`. See *Writing Filters For Rules* for more information. If no filters are desired, it should be specified as an empty list: `filter: []`

`alert` is a list of alerts to run on each match. For more information on alert types, see *Alerts*. The email alert requires an SMTP server for sending mail. By default, it will attempt to use localhost. This can be changed with the `smtp_host` option.

`email` is a list of addresses to which alerts will be sent.

There are many other optional configuration options, see *Common configuration options*.

All documents must have a timestamp field. ElastAlert will try to use `@timestamp` by default, but this can be changed with the `timestamp_field` option. By default, ElastAlert uses ISO8601 timestamps, though unix timestamps are supported by setting `timestamp_type`.

As is, this rule means "Send an email to elastalert@example.com when there are more than 50 documents with `some_field == some_value` within a 4 hour period."

Testing Your Rule

Running the `elastalert-test-rule` tool will test that your config file successfully loads and run it in debug mode over the last 24 hours:

```
$ elastalert-test-rule example_rules/example_frequency.yaml
```

If you want to specify a configuration file to use, you can run it with the config flag.

$ elastalert-test-rule –config <path-to-config-file> example_rules/example_frequency.yaml.

The configuration preferences will be loaded as follows:

1. Configurations specified in the yaml file.
2. Configurations specified in the config file, if specified.
3. Default configurations, for the tool to run.

See *the testing section for more details*

Running ElastAlert

There are two ways of invoking ElastAlert. As a daemon, through Supervisor (http://supervisord.org/), or directly with Python. For easier debugging purposes in this tutorial, we will invoke it directly:

```
$ python -m elastalert.elastalert --verbose --rule example_frequency.yaml  # or use␣
↪the entry point: elastalert --verbose --rule ...
No handlers could be found for logger "Elasticsearch"
INFO:root:Queried rule Example rule from 1-15 14:22 PST to 1-15 15:07 PST: 5 hits
INFO:Elasticsearch:POST http://elasticsearch.example.com:14900/elastalert_status/
↪elastalert_status?op_type=create [status:201 request:0.025s]
INFO:root:Ran Example rule from 1-15 14:22 PST to 1-15 15:07 PST: 5 query hits (0␣
↪already seen), 0 matches, 0 alerts sent
INFO:root:Sleeping for 297 seconds
```

ElastAlert uses the python logging system and `--verbose` sets it to display INFO level messages. `--rule example_frequency.yaml` specifies the rule to run, otherwise ElastAlert will attempt to load the other rules in the example_rules folder.

Let's break down the response to see what's happening.

```
Queried rule Example rule from 1-15 14:22 PST to 1-15 15:07 PST: 5 hits
```

ElastAlert periodically queries the most recent `buffer_time` (default 45 minutes) for data matching the filters. Here we see that it matched 5 hits.

```
POST http://elasticsearch.example.com:14900/elastalert_status/
elastalert_status?op_type=create [status:201 request:0.025s]
```

This line showing that ElastAlert uploaded a document to the elastalert_status index with information about the query it just made.

```
Ran Example rule from 1-15 14:22 PST to 1-15 15:07 PST: 5 query hits (0
already seen), 0 matches, 0 alerts sent
```

The line means ElastAlert has finished processing the rule. For large time periods, sometimes multiple queries may be run, but their data will be processed together. `query hits` is the number of documents that are downloaded from Elasticsearch, `already seen` refers to documents that were already counted in a previous overlapping query

and will be ignored, `matches` is the number of matches the rule type outputted, and `alerts sent` is the number of alerts actually sent. This may differ from `matches` because of options like `realert` and `aggregation` or because of an error.

```
Sleeping for 297 seconds
```

The default `run_every` is 5 minutes, meaning ElastAlert will sleep until 5 minutes have elapsed from the last cycle before running queries for each rule again with time ranges shifted forward 5 minutes.

Say, over the next 297 seconds, 46 more matching documents were added to Elasticsearch:

```
INFO:root:Queried rule Example rule from 1-15 14:27 PST to 1-15 15:12 PST: 51 hits
...
INFO:root:Sent email to ['elastalert@example.com']
...
INFO:root:Ran Example rule from 1-15 14:27 PST to 1-15 15:12 PST: 51 query hits, 1␣
↪matches, 1 alerts sent
```

The body of the email will contain something like:

```
Example rule

At least 50 events occurred between 1-15 11:12 PST and 1-15 15:12 PST

@timestamp: 2015-01-15T15:12:00-08:00
```

If an error occurred, such as an unreachable SMTP server, you may see:

```
ERROR:root:Error while running alert email:  Error connecting to SMTP host:
[Errno 61] Connection refused
```

Note that if you stop ElastAlert and then run it again later, it will look up `elastalert_status` and begin querying at the end time of the last query. This is to prevent duplication or skipping of alerts if ElastAlert is restarted.

By using the `--debug` flag instead of `--verbose`, the body of email will instead be logged and the email will not be sent. In addition, the queries will not be saved to `elastalert_status`.

CHAPTER 3

Rule Types and Configuration Options

Examples of several types of rule configuration can be found in the example_rules folder.

Note: All "time" formats are of the form `unit: X` where unit is one of weeks, days, hours, minutes or seconds. Such as `minutes: 15` or `hours: 1`.

Rule Configuration Cheat Sheet

FOR ALL RULES	
`es_host` (string)	Required
`es_port` (number)	
`index` (string)	
`type` (string)	
`alert` (string or list)	
`name` (string, defaults to the filename)	Optional
`use_strftime_index` (boolean, default False)	
`use_ssl` (boolean, default False)	
`verify_certs` (boolean, default True)	
`es_username` (string, no default)	
`es_password` (string, no default)	
`es_url_prefix` (string, no default)	
`es_send_get_body_as` (string, default "GET")	
`aggregation` (time, no default)	
`description` (string, default empty string)	
`generate_kibana_link` (boolean, default False)	
`use_kibana_dashboard` (string, no default)	
`kibana_url` (string, default from es_host)	
`use_kibana4_dashboard` (string, no default)	

Continued on next page

Table 3.1 – continued from previous page

FOR ALL RULES	
`kibana4_start_timedelta` (time, default: 10 min)	
`kibana4_end_timedelta` (time, default: 10 min)	
`use_local_time` (boolean, default True)	
`realert` (time, default: 1 min)	
`exponential_realert` (time, no default)	
`match_enhancements` (list of strs, no default)	
`top_count_number` (int, default 5)	
`top_count_keys` (list of strs)	
`raw_count_keys` (boolean, default True)	
`include` (list of strs, default ["*"])	
`filter` (ES filter DSL, no default)	
`max_query_size` (int, default global max_query_size)	
`query_delay` (time, default 0 min)	
`owner` (string, default empty string)	
`priority` (int, default 2)	
`import` (string) IGNORED IF `use_count_query` or `use_terms_query` is true	
`buffer_time` (time, default from config.yaml)	
`timestamp_type` (string, default iso)	
`timestamp_format` (string, default "%Y-%m-%dT%H:%M:%SZ")	
`timestamp_format_expr` (string, no default)	
`_source_enabled` (boolean, default True)	
`alert_text_args` (array of strs)	
`alert_text_kw` (object)	

RULE TYPE	Any	Blacklist	Whitelist	Change	Frequency	Spike	Flatline	New_term	Cardinality
`compare_key` (list of strs, no default)		Req	Req	Req					
`blacklist` (list of strs, no default)		Req							
`whitelist` (list of strs, no default)			Req						
`ignore_null` (boolean, no default)			Req	Req					
`query_key` (string, no default)	Opt			Req	Opt	Opt	Opt	Req	Opt
`aggregation_key` (string, no default)	Opt								
`summary_table_fields` (list, no default)	Opt								
`timeframe` (time, no default)				Opt	Req	Req	Req		Req
`num_events` (int, no default)					Req				
`attach_related` (boolean, no default)					Opt				
`use_count_query` (boolean, no default) `doc_type` (string, no default)					Opt	Opt	Opt		
`use_terms_query` (boolean, no default) `doc_type` (string, no default)					Opt	Opt		Opt	
`query_key` (string, no default)									

Common Configuration Options

Every file that ends in `.yaml` in the `rules_folder` will be run by default. The following configuration settings are common to all types of rules.

Required Settings

es_host

`es_host`: The hostname of the Elasticsearch cluster the rule will use to query. (Required, string, no default) The environment variable `ES_HOST` will override this field.

es_port

`es_port`: The port of the Elasticsearch cluster. (Required, number, no default) The environment variable `ES_PORT` will override this field.

index

`index`: The name of the index that will be searched. Wildcards can be used here, such as: `index: my-index-*` which will match `my-index-2014-10-05`. You can also use a format string containing `%Y` for year, `%m` for month, and `%d` for day. To use this, you must also set `use_strftime_index` to true. (Required, string, no default)

name

`name`: The name of the rule. This must be unique across all rules. The name will be used in alerts and used as a key when writing and reading search metadata back from Elasticsearch. (Required, string, no default)

type

`type`: The `RuleType` to use. This may either be one of the built in rule types, see *Rule Types* section below for more information, or loaded from a module. For loading from a module, the type should be specified as `module.file.RuleName`. (Required, string, no default)

alert

`alert`: The `Alerter` type to use. This may be one or more of the built in alerts, see *Alert Types* section below for more information, or loaded from a module. For loading from a module, the alert should be specified as `module.file.AlertName`. (Required, string or list, no default)

Optional Settings

import

`import`: If specified includes all the settings from this yaml file. This allows common config options to be shared. Note that imported files that aren't complete rules should not have a `.yml` or `.yaml` suffix so that ElastAlert doesn't treat them as rules. Filters in imported files are merged (ANDed) with any filters in the rule. You can only have one

import per rule, though the imported file can import another file, recursively. The filename can be an absolute path or relative to the rules directory. (Optional, string, no default)

use_ssl

`use_ssl`: Whether or not to connect to `es_host` using TLS. (Optional, boolean, default False) The environment variable `ES_USE_SSL` will override this field.

verify_certs

`verify_certs`: Whether or not to verify TLS certificates. (Optional, boolean, default True)

client_cert

`client_cert`: Path to a PEM certificate to use as the client certificate (Optional, string, no default)

client_key

`client_key`: Path to a private key file to use as the client key (Optional, string, no default)

ca_certs

`ca_certs`: Path to a CA cert bundle to use to verify SSL connections (Optional, string, no default)

es_username

`es_username`: basic-auth username for connecting to `es_host`. (Optional, string, no default) The environment variable `ES_USERNAME` will override this field.

es_password

`es_password`: basic-auth password for connecting to `es_host`. (Optional, string, no default) The environment variable `ES_PASSWORD` will override this field.

es_url_prefix

`es_url_prefix`: URL prefix for the Elasticsearch endpoint. (Optional, string, no default)

es_send_get_body_as

`es_send_get_body_as`: Method for querying Elasticsearch. (Optional, string, default "GET")

use_strftime_index

`use_strftime_index`: If this is true, ElastAlert will format the index using datetime.strftime for each query. See https://docs.python.org/2/library/datetime.html#strftime-strptime-behavior for more details. If a query spans multiple days, the formatted indexes will be concatenated with commas. This is useful as narrowing the number of indexes searched, compared to using a wildcard, may be significantly faster. For example, if `index` is `logstash-%Y.%m.%d`, the query url will be similar to `elasticsearch.example.com/logstash-2015.02.03/...` or `elasticsearch.example.com/logstash-2015.02.03,logstash-2015.02.04/...`.

aggregation

`aggregation`: This option allows you to aggregate multiple matches together into one alert. Every time a match is found, ElastAlert will wait for the `aggregation` period, and send all of the matches that have occurred in that time for a particular rule together.

For example:

```
aggregation:
  hours: 2
```

means that if one match occurred at 12:00, another at 1:00, and a third at 2:30, one alert would be sent at 2:00, containing the first two matches, and another at 4:30, containing the third match plus any additional matches occurring before 4:30. This can be very useful if you expect a large number of matches and only want a periodic report. (Optional, time, default none)

If you wish to aggregate all your alerts and send them on a recurring interval, you can do that using the `schedule` field.

For example, if you wish to receive alerts every Monday and Friday:

```
aggregation:
  schedule: '2 4 * * mon,fri'
```

This uses Cron syntax, which you can read more about here. Make sure to *only* include either a schedule field or standard datetime fields (such as `hours`, `minutes`, `days`), not both.

By default, all events that occur during an aggregation window are grouped together. However, if your rule has the `aggregation_key` field set, then each event sharing a common key value will be grouped together. A separate aggregation window will be made for each newly encountered key value.

For example, if you wish to receive alerts that are grouped by the user who triggered the event, you can set:

```
aggregation_key: 'my_data.username'
```

Then, assuming an aggregation window of 10 minutes, if you receive the following data points:

```
{'my_data': {'username': 'alice', 'event_type': 'login'}, '@timestamp': '2016-09-
↪20T00:00:00'}
{'my_data': {'username': 'bob', 'event_type': 'something'}, '@timestamp': '2016-09-
↪20T00:05:00'}
{'my_data': {'username': 'alice', 'event_type': 'something else'}, '@timestamp':
↪'2016-09-20T00:06:00'}
```

This should result in 2 alerts: One containing alice's two events, sent at `2016-09-20T00:10:00` and one containing bob's one event sent at `2016-09-20T00:16:00`

For aggregations, there can sometimes be a large number of documents present in the viewing medium (email, jira ticket, etc..). If you set the `summary_table_fields` field, Elastalert will provide a summary of the specified fields from all the results.

For example, if you wish to summarize the usernames and event_types that appear in the documents so that you can see the most relevant fields at a quick glance, you can set:

```
summary_table_fields:
    - my_data.username
    - my_data.event_type
```

Then, for the same sample data shown above listing alice and bob's events, Elastalert will provide the following summary table in the alert medium:

```
+------------------+--------------------+
| my_data.username | my_data.event_type |
+------------------+--------------------+
|      alice       |       login        |
|       bob        |     something      |
|      alice       |   something else   |
+------------------+--------------------+
```

Note: By default, aggregation time is relative to the current system time, not the time of the match. This means that running elastalert over past events will result in different alerts than if elastalert had been running while those events occured. This behavior can be changed by setting `aggregate_by_match_time`.

aggregate_by_match_time

Setting this to true will cause aggregations to be created relative to the timestamp of the first event, rather than the current time. This is useful for querying over historic data or if using a very large buffer_time and you want multiple aggregations to occur from a single query.

realert

`realert`: This option allows you to ignore repeating alerts for a period of time. If the rule uses a `query_key`, this option will be applied on a per key basis. All matches for a given rule, or for matches with the same `query_key`, will be ignored for the given time. All matches with a missing `query_key` will be grouped together using a value of `_missing`. This is applied to the time the alert is sent, not to the time of the event. It defaults to one minute, which means that if ElastAlert is run over a large time period which triggers many matches, only the first alert will be sent by default. If you want every alert, set realert to 0 minutes. (Optional, time, default 1 minute)

exponential_realert

`exponential_realert`: This option causes the value of `realert` to exponentially increase while alerts continue to fire. If set, the value of `exponential_realert` is the maximum `realert` will increase to. If the time between alerts is less than twice `realert`, `realert` will double. For example, if `realert: minutes: 10` and `exponential_realert: hours: 1`, an alerts fires at 1:00 and another at 1:15, the next alert will not be until at least 1:35. If another alert fires between 1:35 and 2:15, `realert` will increase to the 1 hour maximum. If more than 2 hours elapse before the next alert, `realert` will go back down. Note that alerts that are ignored (e.g. one that occurred at 1:05) would not change `realert`. (Optional, time, no default)

buffer_time

`buffer_time`: This options allows the rule to override the `buffer_time` global setting defined in config.yaml. This value is ignored if `use_count_query` or `use_terms_query` is true. (Optional, time)

query_delay

`query_delay`: This option will cause ElastAlert to subtract a time delta from every query, causing the rule to run with a delay. This is useful if the data is Elasticsearch doesn't get indexed immediately. (Optional, time)

owner

`owner`: This value will be used to identify the stakeholder of the alert. Optionally, this field can be included in any alert type. (Optional, string)

priority

`priority`: This value will be used to identify the relative priority of the alert. Optionally, this field can be included in any alert type (e.g. for use in email subject/body text). (Optional, int, default 2)

max_query_size

`max_query_size`: The maximum number of documents that will be downloaded from Elasticsearch in a single query. If you expect a large number of results, consider using `use_count_query` for the rule. If this limit is reached, a warning will be logged but ElastAlert will continue without downloading more results. This setting will override a global `max_query_size`. (Optional, int, default value of global `max_query_size`)

filter

`filter`: A list of Elasticsearch query DSL filters that is used to query Elasticsearch. ElastAlert will query Elasticsearch using the format `{'filter': {'bool': {'must': [config.filter]}}}` with an additional timestamp range filter. All of the results of querying with these filters are passed to the `RuleType` for analysis. For more information writing filters, see *Writing Filters*. (Required, Elasticsearch query DSL, no default)

include

`include`: A list of terms that should be included in query results and passed to rule types and alerts. When set, only those fields, along with '@timestamp', `query_key`, `compare_key`, and `top_count_keys` are included, if present. (Optional, list of strings, default all fields)

top_count_keys

`top_count_keys`: A list of fields. ElastAlert will perform a terms query for the top X most common values for each of the fields, where X is 5 by default, or `top_count_number` if it exists. For example, if `num_events` is 100, and `top_count_keys` is `- "username"`, the alert will say how many of the 100 events have each username, for the top 5 usernames. When this is computed, the time range used is from `timeframe` before the most recent event to 10 minutes past the most recent event. Because ElastAlert uses an aggregation query to compute this, it will attempt to use the field name plus ".raw" to count unanalyzed terms. To turn this off, set `raw_count_keys` to false.

top_count_number

`top_count_number`: The number of terms to list if `top_count_keys` is set. (Optional, integer, default 5)

raw_count_keys

`raw_count_keys`: If true, all fields in `top_count_keys` will have `.raw` appended to them. (Optional, boolean, default true)

description

`description`: text describing the purpose of rule. (Optional, string, default empty string) Can be referenced in custom alerters to provide context as to why a rule might trigger.

generate_kibana_link

`generate_kibana_link`: This option is for Kibana 3 only. If true, ElastAlert will generate a temporary Kibana dashboard and include a link to it in alerts. The dashboard consists of an events over time graph and a table with `include` fields selected in the table. If the rule uses `query_key`, the dashboard will also contain a filter for the `query_key` of the alert. The dashboard schema will be uploaded to the kibana-int index as a temporary dashboard. (Optional, boolean, default False)

kibana_url

`kibana_url`: The url to access Kibana. This will be used if `generate_kibana_link` or `use_kibana_dashboard` is true. If not specified, a URL will be constructed using `es_host` and `es_port`. (Optional, string, default `http://<es_host>:<es_port>/_plugin/kibana/`)

use_kibana_dashboard

`use_kibana_dashboard`: The name of a Kibana 3 dashboard to link to. Instead of generating a dashboard from a template, ElastAlert can use an existing dashboard. It will set the time range on the dashboard to around the match time, upload it as a temporary dashboard, add a filter to the `query_key` of the alert if applicable, and put the url to the dashboard in the alert. (Optional, string, no default)

use_kibana4_dashboard

`use_kibana4_dashboard`: A link to a Kibana 4 dashboard. For example, "https://kibana.example.com/#/dashboard/My-Dashboard". This will set the time setting on the dashboard from the match time minus the timeframe, to 10 minutes after the match time. Note that this does not support filtering by `query_key` like Kibana 3. This value can use *$VAR* and *${VAR}* references to expand environment variables.

kibana4_start_timedelta

`kibana4_start_timedelta`: Defaults to 10 minutes. This option allows you to specify the start time for the generated kibana4 dashboard. This value is added in front of the event. For example,

```
kibana4_start_timedelta: minutes: 2
```

kibana4_end_timedelta

`kibana4_end_timedelta`: Defaults to 10 minutes. This option allows you to specify the end time for the generated kibana4 dashboard. This value is added in back of the event. For example,

```
kibana4_end_timedelta:  minutes:  2
```

use_local_time

`use_local_time`: Whether to convert timestamps to the local time zone in alerts. If false, timestamps will be converted to UTC, which is what ElastAlert uses internally. (Optional, boolean, default true)

match_enhancements

`match_enhancements`: A list of enhancement modules to use with this rule. An enhancement module is a subclass of enhancements.BaseEnhancement that will be given the match dictionary and can modify it before it is passed to the alerter. The enhancements will be run after silence and realert is calculated and in the case of aggregated alerts, right before the alert is sent. This can be changed by setting `run_enhancements_first`. The enhancements should be specified as `module.file.EnhancementName`. See *Enhancements* for more information. (Optional, list of strings, no default)

run_enhancements_first

`run_enhancements_first`: If set to true, enhancements will be run as soon as a match is found. This means that they can be changed or dropped before affecting realert or being added to an aggregation. Silence stashes will still be created before the enhancement runs, meaning even if a `DropMatchException` is raised, the rule will still be silenced. (Optional, boolean, default false)

query_key

`query_key`: Having a query key means that realert time will be counted separately for each unique value of `query_key`. For rule types which count documents, such as spike, frequency and flatline, it also means that these counts will be independent for each unique value of `query_key`. For example, if `query_key` is set to `username` and `realert` is set, and an alert triggers on a document with `{'username':  'bob'}`, additional alerts for `{'username':  'bob'}` will be ignored while other usernames will trigger alerts. Documents which are missing the `query_key` will be grouped together. A list of fields may also be used, which will create a compound query key. This compound key is treated as if it were a single field whose value is the component values, or "None", joined by commas. A new field with the key "field1,field2,etc" will be created in each document and may conflict with existing fields of the same name.

aggregation_key

`aggregation_key`: Having an aggregation key in conjunction with an aggregation will make it so that each new value encountered for the aggregation_key field will result in a new, separate aggregation window.

summary_table_fields

`summary_table_fields`: Specifying the summmary_table_fields in conjunction with an aggregation will make it so that each aggregated alert will contain a table summarizing the values for the specified fields in all the matches that were aggregated together.

timestamp_type

`timestamp_type`: One of `iso`, `unix`, `unix_ms`, `custom`. This option will set the type of `@timestamp` (or `timestamp_field`) used to query Elasticsearch. `iso` will use ISO8601 timestamps, which will work with most Elasticsearch date type field. `unix` will query using an integer unix (seconds since 1/1/1970) timestamp. `unix_ms` will use milliseconds unix timestamp. `custom` allows you to define your own `timestamp_format`. The default is `iso`. (Optional, string enum, default iso).

timestamp_format

`timestamp_format`: In case Elasticsearch used custom date format for date type field, this option provides a way to define custom timestamp format to match the type used for Elastisearch date type field. This option is only valid if `timestamp_type` set to `custom`. (Optional, string, default '%Y-%m-%dT%H:%M:%SZ').

timestamp_format_expr

`timestamp_format_expr`: In case Elasticsearch used custom date format for date type field, this option provides a way to adapt the value obtained converting a datetime through `timestamp_format`, when the format cannot match perfectly what defined in Elastisearch. When set, this option is evaluated as a Python expression along with a *globals* dictionary containing the original datetime instance named `dt` and the timestamp to be refined, named `ts`. The returned value becomes the timestamp obtained from the datetime. For example, when the date type field in Elasticsearch uses milliseconds (`yyyy-MM-dd'T'HH:mm:ss.SSS'Z'`) and `timestamp_format` option is `'%Y-%m-%dT%H:%M:%S.%fZ'`, Elasticsearch would fail to parse query terms as they contain microsecond values - that is it gets 6 digits instead of 3 - since the `%f` placeholder stands for microseconds for Python *strftime* method calls. Setting `timestamp_format_expr: 'ts[:23] + ts[26:]'` will truncate the value to milliseconds granting Elasticsearch compatibility. This option is only valid if `timestamp_type` set to `custom`. (Optional, string, no default).

_source_enabled

`_source_enabled`: If true, ElastAlert will use _source to retrieve fields from documents in Elasticsearch. If false, ElastAlert will use `fields` to retrieve stored fields. Both of these are represented internally as if they came from `_source`. See https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping-fields.html for more details. The fields used come from `include`, see above for more details. (Optional, boolean, default True)

Some rules and alerts require additional options, which also go in the top level of the rule configuration file.

Testing Your Rule

Once you've written a rule configuration, you will want to validate it. To do so, you can either run ElastAlert in debug mode, or use `elastalert-test-rule`, which is a script that makes various aspects of testing easier.

It can:

- Check that the configuration file loaded successfully.
- Check that the Elasticsearch filter parses.
- Run against the last X day(s) and the show the number of hits that match your filter.
- Show the available terms in one of the results.
- Save documents returned to a JSON file.

- Run ElastAlert using either a JSON file or actual results from Elasticsearch.
- Print out debug alerts or trigger real alerts.
- Check that, if they exist, the primary_key, compare_key and include terms are in the results.
- Show what metadata documents would be written to `elastalert_status`.

Without any optional arguments, it will run ElastAlert over the last 24 hours and print out any alerts that would have occurred. Here is an example test run which triggered an alert:

```
$ elastalert-test-rule my_rules/rule1.yaml
Successfully Loaded Example rule1

Got 105 hits from the last 1 day

Available terms in first hit:
    @timestamp
    field1
    field2
    ...
Included term this_field_doesnt_exist may be missing or null

INFO:root:Queried rule Example rule1 from 6-16 15:21 PDT to 6-17 15:21 PDT: 105 hits
INFO:root:Alert for Example rule1 at 2015-06-16T23:53:12Z:
INFO:root:Example rule1

At least 50 events occurred between 6-16 18:30 PDT and 6-16 20:30 PDT

field1:
value1: 25
value2: 25

@timestamp: 2015-06-16T20:30:04-07:00
field1: value1
field2: something


Would have written the following documents to elastalert_status:

silence - {'rule_name': 'Example rule1', '@timestamp': datetime.datetime( ... ),
↪'exponent': 0, 'until':
datetime.datetime( ... )}

elastalert_status - {'hits': 105, 'matches': 1, '@timestamp': datetime.datetime( ...␣
↪), 'rule_name': 'Example rule1',
'starttime': datetime.datetime( ... ), 'endtime': datetime.datetime( ... ), 'time_
↪taken': 3.1415926}
```

Note that everything between "Alert for Example rule1 at ..." and "Would have written the following ..." is the exact text body that an alert would have. See the section below on alert content for more details. Also note that datetime objects are converted to ISO8601 timestamps when uploaded to Elasticsearch. See *the section on metadata* for more details.

Other options include:

`--schema-only`: Only perform schema validation on the file. It will not load modules or query Elasticsearch. This may catch invalid YAML and missing or misconfigured fields.

`--count-only`: Only find the number of matching documents and list available fields. ElastAlert will not be run and documents will not be downloaded.

`--days N`: Instead of the default 1 day, query N days. For selecting more specific time ranges, you must run ElastAlert itself and use `--start` and `--end`.

`--save-json FILE`: Save all documents downloaded to a file as JSON. This is useful if you wish to modify data while testing or do offline testing in conjunction with `--data FILE`. A maximum of 10,000 documents will be downloaded.

`--data FILE`: Use a JSON file as a data source instead of Elasticsearch. The file should be a single list containing objects, rather than objects on separate lines. Note than this uses mock functions which mimic some Elasticsearch query methods and is not guaranteed to have the exact same results as with Elasticsearch. For example, analyzed string fields may behave differently.

`--alert`: Trigger real alerts instead of the debug (logging text) alert.

Note: Results from running this script may not always be the same as if an actual ElastAlert instance was running. Some rule types, such as spike and flatline require a minimum elapsed time before they begin alerting, based on their timeframe. In addition, use_count_query and use_terms_query rely on run_every to determine their resolution. This script uses a fixed 5 minute window, which is the same as the default.

Rule Types

The various `RuleType` classes, defined in `elastalert/ruletypes.py`, form the main logic behind ElastAlert. An instance is held in memory for each rule, passed all of the data returned by querying Elasticsearch with a given filter, and generates matches based on that data.

To select a rule type, set the `type` option to the name of the rule type in the rule configuration file:

`type:  <rule type>`

Any

`any`: The any rule will match everything. Every hit that the query returns will generate an alert.

Blacklist

`blacklist`: The blacklist rule will check a certain field against a blacklist, and match if it is in the blacklist.

This rule requires two additional options:

`compare_key`: The name of the field to use to compare to the blacklist. If the field is null, those events will be ignored.

`blacklist`: A list of blacklisted values, and/or a list of paths to flat files which contain the blacklisted values using `- "!file /path/to/file"`; for example:

```
blacklist:
    - value1
    - value2
    - "!file /tmp/blacklist1.txt"
    - "!file /tmp/blacklist2.txt"
```

It is possible to mix between blacklist value definitions, or use either one. The `compare_key` term must be equal to one of these values for it to match.

Whitelist

`whitelist`: Similar to `blacklist`, this rule will compare a certain field to a whitelist, and match if the list does not contain the term.

This rule requires three additional options:

`compare_key`: The name of the field to use to compare to the whitelist.

`ignore_null`: If true, events without a `compare_key` field will not match.

`whitelist`: A list of whitelisted values, and/or a list of paths to flat files which contain the whitelisted values using `- "!file /path/to/file"`; for example:

```
whitelist:
    - value1
    - value2
    - "!file /tmp/whitelist1.txt"
    - "!file /tmp/whitelist2.txt"
```

It is possible to mix between whitelisted value definitions, or use either one. The `compare_key` term must be in this list or else it will match.

Change

For an example configuration file using this rule type, look at `example_rules/example_change.yaml`.

`change`: This rule will monitor a certain field and match if that field changes. The field must change with respect to the last event with the same `query_key`.

This rule requires three additional options:

`compare_key`: The names of the field to monitor for changes. Since this is list of strings, we can have multiple keys. An alert will trigger if any of the fields change.

`ignore_null`: If true, events without a `compare_key` field will not count as changed. Currently this check for all the fields in `compare_key`

`query_key`: This rule is applied on a per-`query_key` basis. This field must be present in all of the events that are checked.

There is also an optional field:

`timeframe`: The maximum time between changes. After this time period, ElastAlert will forget the old value of the `compare_key` field.

Frequency

For an example configuration file using this rule type, look at `example_rules/example_frequency.yaml`.

`frequency`: This rule matches when there are at least a certain number of events in a given time frame. This may be counted on a per-`query_key` basis.

This rule requires two additional options:

`num_events`: The number of events which will trigger an alert.

`timeframe`: The time that `num_events` must occur within.

Optional:

use_count_query: If true, ElastAlert will poll Elasticsearch using the count api, and not download all of the matching documents. This is useful is you care only about numbers and not the actual data. It should also be used if you expect a large number of query hits, in the order of tens of thousands or more. doc_type must be set to use this.

doc_type: Specify the _type of document to search for. This must be present if use_count_query or use_terms_query is set.

use_terms_query: If true, ElastAlert will make an aggregation query against Elasticsearch to get counts of documents matching each unique value of query_key. This must be used with query_key and doc_type. This will only return a maximum of terms_size, default 50, unique terms.

terms_size: When used with use_terms_query, this is the maximum number of terms returned per query. Default is 50.

query_key: Counts of documents will be stored independently for each value of query_key. Only num_events documents, all with the same value of query_key, will trigger an alert.

attach_related: Will attach all the related events to the event that triggered the frequency alert. For example in an alert triggered with num_events: 3, the 3rd event will trigger the alert on itself and add the other 2 events in a key named related_events that can be accessed in the alerter.

Spike

spike: This rule matches when the volume of events during a given time period is spike_height times larger or smaller than during the previous time period. It uses two sliding windows to compare the current and reference frequency of events. We will call this two windows "reference" and "current".

This rule requires three additional options:

spike_height: The ratio of number of events in the last timeframe to the previous timeframe that when hit will trigger an alert.

spike_type: Either 'up', 'down' or 'both'. 'Up' meaning the rule will only match when the number of events is spike_height times higher. 'Down' meaning the reference number is spike_height higher than the current number. 'Both' will match either.

timeframe: The rule will average out the rate of events over this time period. For example, hours: 1 means that the 'current' window will span from present to one hour ago, and the 'reference' window will span from one hour ago to two hours ago. The rule will not be active until the time elapsed from the first event is at least two timeframes. This is to prevent an alert being triggered before a baseline rate has been established. This can be overridden using alert_on_new_data.

Optional:

threshold_ref: The minimum number of events that must exist in the reference window for an alert to trigger. For example, if spike_height: 3 and threshold_ref: 10, then the 'reference' window must contain at least 10 events and the 'current' window at least three times that for an alert to be triggered.

threshold_cur: The minimum number of events that must exist in the current window for an alert to trigger. For example, if spike_height: 3 and threshold_cur: 60, then an alert will occur if the current window has more than 60 events and the reference window has less than a third as many.

To illustrate the use of threshold_ref, threshold_cur, alert_on_new_data, timeframe and spike_height together, consider the following examples:

```
" Alert if at least 15 events occur within two hours and less than a quarter of that␣
↪number occurred within the previous two hours. "
timeframe: hours: 2
spike_height: 4
spike_type: up
```

```
threshold_cur: 15

hour1: 5 events (ref: 0, cur: 5) - No alert because (a) threshold_cur not met, (b)␣
↪ref window not filled
hour2: 5 events (ref: 0, cur: 10) - No alert because (a) threshold_cur not met, (b)␣
↪ref window not filled
hour3: 10 events (ref: 5, cur: 15) - No alert because (a) spike_height not met, (b)␣
↪ref window not filled
hour4: 35 events (ref: 10, cur: 45) - Alert because (a) spike_height met, (b)␣
↪threshold_cur met, (c) ref window filled

hour1: 20 events (ref: 0, cur: 20) - No alert because ref window not filled
hour2: 21 events (ref: 0, cur: 41) - No alert because ref window not filled
hour3: 19 events (ref: 20, cur: 40) - No alert because (a) spike_height not met, (b)␣
↪ref window not filled
hour4: 23 events (ref: 41, cur: 42) - No alert because spike_height not met

hour1: 10 events (ref: 0, cur: 10) - No alert because (a) threshold_cur not met, (b)␣
↪ref window not filled
hour2: 0 events (ref: 0, cur: 10) - No alert because (a) threshold_cur not met, (b)␣
↪ref window not filled
hour3: 0 events (ref: 10, cur: 0) - No alert because (a) threshold_cur not met, (b)␣
↪ref window not filled, (c) spike_height not met
hour4: 30 events (ref: 10, cur: 30) - No alert because spike_height not met
hour5: 5 events (ref: 0, cur: 35) - Alert because (a) spike_height met, (b) threshold_
↪cur met, (c) ref window filled

" Alert if at least 5 events occur within two hours, and twice as many events occur␣
↪within the next two hours. "
timeframe: hours: 2
spike_height: 2
spike_type: up
threshold_ref: 5

hour1: 20 events (ref: 0, cur: 20) - No alert because (a) threshold_ref not met, (b)␣
↪ref window not filled
hour2: 100 events (ref: 0, cur: 120) - No alert because (a) threshold_ref not met,␣
↪(b) ref window not filled
hour3: 100 events (ref: 20, cur: 200) - No alert because ref window not filled
hour4: 100 events (ref: 120, cur: 200) - No alert because spike_height not met

hour1: 0 events (ref: 0, cur: 0) - No alert because (a) threshold_ref not met, (b)␣
↪ref window not filled
hour2: 20 events (ref: 0, cur: 20) - No alert because (a) threshold_ref not met, (b)␣
↪ref window not filled
hour3: 100 events (ref: 0, cur: 120) - No alert because (a) threshold_ref not met,␣
↪(b) ref window not filled
hour4: 100 events (ref: 20, cur: 200) - Alert because (a) spike_height met, (b)␣
↪threshold_ref met, (c) ref window filled

hour1: 1 events (ref: 0, cur: 1) - No alert because (a) threshold_ref not met, (b)␣
↪ref window not filled
hour2: 2 events (ref: 0, cur: 3) - No alert because (a) threshold_ref not met, (b)␣
↪ref window not filled
hour3: 2 events (ref: 1, cur: 4) - No alert because (a) threshold_ref not met, (b)␣
↪ref window not filled
hour4: 1000 events (ref: 3, cur: 1002) - No alert because threshold_ref not met
hour5: 2 events (ref: 4, cur: 1002) - No alert because threshold_ref not met
```

```
hour6: 4 events: (ref: 1002, cur: 6) - No alert because spike_height not met

hour1: 1000 events (ref: 0, cur: 1000) - No alert because (a) threshold_ref not met,␣
↪(b) ref window not filled
hour2: 0 events (ref: 0, cur: 1000) - No alert because (a) threshold_ref not met, (b)␣
↪ref window not filled
hour3: 0 events (ref: 1000, cur: 0) - No alert because (a) spike_height not met, (b)␣
↪ref window not filled
hour4: 0 events (ref: 1000, cur: 0) - No alert because spike_height not met
hour5: 1000 events (ref: 0, cur: 1000) - No alert because threshold_ref not met
hour6: 1050 events (ref: 0, cur: 2050)- No alert because threshold_ref not met
hour7: 1075 events (ref: 1000, cur: 2125) Alert because (a) spike_height met, (b)␣
↪threshold_ref met, (c) ref window filled

" Alert if at least 100 events occur within two hours and less than a fifth of that␣
↪number occurred in the previous two hours. "
timeframe: hours: 2
spike_height: 5
spike_type: up
threshold_cur: 100

hour1: 1000 events (ref: 0, cur: 1000) - No alert because ref window not filled

hour1: 2 events (ref: 0, cur: 2) - No alert because (a) threshold_cur not met, (b)␣
↪ref window not filled
hour2: 1 events (ref: 0, cur: 3) - No alert because (a) threshold_cur not met, (b)␣
↪ref window not filled
hour3: 20 events (ref: 2, cur: 21) - No alert because (a) threshold_cur not met, (b)␣
↪ref window not filled
hour4: 81 events (ref: 3, cur: 101) - Alert because (a) spike_height met, (b)␣
↪threshold_cur met, (c) ref window filled

hour1: 10 events (ref: 0, cur: 10) - No alert because (a) threshold_cur not met, (b)␣
↪ref window not filled
hour2: 20 events (ref: 0, cur: 30) - No alert because (a) threshold_cur not met, (b)␣
↪ref window not filled
hour3: 40 events (ref: 10, cur: 60) - No alert because (a) threshold_cur not met, (b)␣
↪ref window not filled
hour4: 80 events (ref: 30, cur: 120) - No alert because spike_height not met
hour5: 200 events (ref: 60, cur: 280) - No alert because spike_height not met
```

`alert_on_new_data`: This option is only used if `query_key` is set. When this is set to true, any new `query_key` encountered may trigger an immediate alert. When set to false, baseline must be established for each new `query_key` value, and then subsequent spikes may cause alerts. Baseline is established after `timeframe` has elapsed twice since first occurrence.

`use_count_query`: If true, ElastAlert will poll Elasticsearch using the count api, and not download all of the matching documents. This is useful is you care only about numbers and not the actual data. It should also be used if you expect a large number of query hits, in the order of tens of thousands or more. `doc_type` must be set to use this.

`doc_type`: Specify the `_type` of document to search for. This must be present if `use_count_query` or `use_terms_query` is set.

`use_terms_query`: If true, ElastAlert will make an aggregation query against Elasticsearch to get counts of documents matching each unique value of `query_key`. This must be used with `query_key` and `doc_type`. This will only return a maximum of `terms_size`, default 50, unique terms.

`terms_size`: When used with `use_terms_query`, this is the maximum number of terms returned per query. Default is 50.

query_key: Counts of documents will be stored independently for each value of query_key.

Flatline

flatline: This rule matches when the total number of events is under a given threshold for a time period.

This rule requires two additional options:

threshold: The minimum number of events for an alert not to be triggered.

timeframe: The time period that must contain less than threshold events.

Optional:

use_count_query: If true, ElastAlert will poll Elasticsearch using the count api, and not download all of the matching documents. This is useful is you care only about numbers and not the actual data. It should also be used if you expect a large number of query hits, in the order of tens of thousands or more. doc_type must be set to use this.

doc_type: Specify the _type of document to search for. This must be present if use_count_query or use_terms_query is set.

use_terms_query: If true, ElastAlert will make an aggregation query against Elasticsearch to get counts of documents matching each unique value of query_key. This must be used with query_key and doc_type. This will only return a maximum of terms_size, default 50, unique terms.

terms_size: When used with use_terms_query, this is the maximum number of terms returned per query. Default is 50.

query_key: With flatline rule, query_key means that an alert will be triggered if any value of query_key has been seen at least once and then falls below the threshold.

New Term

new_term: This rule matches when a new value appears in a field that has never been seen before. When ElastAlert starts, it will use an aggregation query to gather all known terms for a list of fields.

This rule requires one additional option:

fields: A list of fields to monitor for new terms. query_key will be used if fields is not set. Each entry in the list of fields can itself be a list. If a field entry is provided as a list, it will be interpreted as a set of fields that compose a composite key used for the ElasticSearch query.

Note: The composite fields may only refer to primitive types, otherwise the initial ElasticSearch query will not properly return the aggregation results, thus causing alerts to fire every time the ElastAlert service initially launches with the rule. A warning will be logged to the console if this scenario is encountered. However, future alerts will actually work as expected after the initial flurry.

Optional:

terms_window_size: The amount of time used for the initial query to find existing terms. No term that has occurred within this time frame will trigger an alert. The default is 30 days.

window_step_size: When querying for existing terms, split up the time range into steps of this size. For example, using the default 30 day window size, and the default 1 day step size, 30 invididvual queries will be made. This helps to avoid timeouts for very expensive aggregation queries. The default is 1 day.

alert_on_missing_field: Whether or not to alert when a field is missing from a document. The default is false.

use_terms_query: If true, ElastAlert will use aggregation queries to get terms instead of regular search queries. This is faster than regular searching if there is a large number of documents. If this is used, you may only specify a single field, and must also set query_key to that field. Also, note that terms_size (the number of buckets returned per query) defaults to 50. This means that if a new term appears but there are at least 50 terms which appear more frequently, it will not be found.

Cardinality

cardinality: This rule matches when a the total number of unique values for a certain field within a time frame is higher or lower than a threshold.

This rule requires:

timeframe: The time period in which the number of unique values will be counted.

cardinality_field: Which field to count the cardinality for.

This rule requires one of the two following options:

max_cardinality: If the cardinality of the data is greater than this number, an alert will be triggered. Each new event that raises the cardinality will trigger an alert.

min_cardinality: If the cardinality of the data is lower than this number, an alert will be triggered. The timeframe must have elapsed since the first event before any alerts will be sent. When a match occurs, the timeframe will be reset and must elapse again before additional alerts.

Optional:

query_key: Group cardinality counts by this field. For each unique value of the query_key field, cardinality will be counted separately.

Metric Aggregation

metric_aggregation: This rule matches when the value of a metric within the calculation window is higher or lower than a threshold. By default this is buffer_time.

This rule requires:

metric_agg_key: This is the name of the field over which the metric value will be calculated. The underlying type of this field must be supported by the specified aggregation type.

metric_agg_type: The type of metric aggregation to perform on the metric_agg_key field. This must be one of 'min', 'max', 'avg', 'sum', 'cardinality', 'value_count'.

doc_type: Specify the _type of document to search for.

This rule also requires at least one of the two following options:

max_threshold: If the calculated metric value is greater than this number, an alert will be triggered. This threshold is exclusive.

min_threshold: If the calculated metric value is less than this number, an alert will be triggered. This threshold is exclusive.

Optional:

query_key: Group metric calculations by this field. For each unique value of the query_key field, the metric will be calculated and evaluated separately against the threshold(s).

use_run_every_query_size: By default the metric value is calculated over a buffer_time sized window. If this parameter is true the rule will use run_every as the calculation window.

allow_buffer_time_overlap: This setting will only have an effect if use_run_every_query_size is false and buffer_time is greater than run_every. If true will allow the start of the metric calculation window to overlap the end time of a previous run. By default the start and end times will not overlap, so if the time elapsed since the last run is less than the metric calculation window size, rule execution will be skipped (to avoid calculations on partial data).

bucket_interval: If present this will divide the metric calculation window into bucket_interval sized segments. The metric value will be calculated and evaluated against the threshold(s) for each segment. If bucket_interval is specified then buffer_time must be a multiple of bucket_interval. (Or run_every if use_run_every_query_size is true).

sync_bucket_interval: This only has an effect if bucket_interval is present. If true it will sync the start and end times of the metric calculation window to the keys (timestamps) of the underlying date_histogram buckets. Because of the way elasticsearch calculates date_histogram bucket keys these usually round evenly to nearest minute, hour, day etc (depending on the bucket size). By default the bucket keys are offset to allign with the time elastalert runs, (This both avoid calculations on partial data, and ensures the very latest documents are included). See: https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-datehistogram-aggregation.html#_offset for a more comprehensive explaination.

Percentage Match

percentage_match: This rule matches when the percentage of document in the match bucket within a calculation window is higher or lower than a threshold. By default the calculation window is buffer_time.

This rule requires:

match_bucket_filter: ES filter DSL. This defines a filter for the match bucket, which should match a subset of the documents returned by the main query filter.

doc_type: Specify the _type of document to search for.

This rule also requires at least one of the two following options:

min_percentage: If the percentage of matching documents is less than this number, an alert will be triggered.

max_percentage: If the percentage of matching documents is greater than this number, an alert will be triggered.

Optional:

query_key: Group percentage by this field. For each unique value of the query_key field, the percentage will be calculated and evaluated separately against the threshold(s).

use_run_every_query_size: See use_run_every_query_size in Metric Aggregation rule

allow_buffer_time_overlap: See allow_buffer_time_overlap in Metric Aggregation rule

bucket_interval: See bucket_interval in Metric Aggregation rule

sync_bucket_interval: See sync_bucket_interval in Metric Aggregation rule

Alerts

Each rule may have any number of alerts attached to it. Alerts are subclasses of Alerter and are passed a dictionary, or list of dictionaries, from ElastAlert which contain relevant information. They are configured in the rule configuration file similarly to rule types.

To set the alerts for a rule, set the alert option to the name of the alert, or a list of the names of alerts:

```
alert:  email
```

or

```
alert:
- email
- jira
```

E-mail subjects, JIRA issue summaries, and PagerDuty alerts can also be customized by adding an `alert_subject` that contains a custom summary. It can be further formatted using standard Python formatting syntax:

```
alert_subject: "Issue {0} occurred at {1}"
```

The arguments for the formatter will be fed from the matched objects related to the alert. The field names whose values will be used as the arguments can be passed with `alert_subject_args`:

```
alert_subject_args:
- issue.name
- "@timestamp"
```

It is mandatory to enclose the `@timestamp` field in quotes since in YAML format a token cannot begin with the `@` character. Not using the quotation marks will trigger a YAML parse error.

In case the rule matches multiple objects in the index, only the first match is used to populate the arguments for the formatter.

If the field(s) mentioned in the arguments list are missing, the email alert will have the text `<MISSING VALUE>` in place of its expected value. This will also occur if `use_count_query` is set to true.

Alert Content

There are several ways to format the body text of the various types of events. In EBNF:

```
rule_name             = name
alert_text            = alert_text
ruletype_text         = Depends on type
top_counts_header     = top_count_key, ":"
top_counts_value      = Value, ": ", Count
top_counts            = top_counts_header, LF, top_counts_value
field_values          = Field, ": ", Value
```

Similarly to `alert_subject`, `alert_text` can be further formatted using standard Python formatting syntax. The field names whose values will be used as the arguments can be passed with `alert_text_args` or `alert_text_kw`. You may also refer to any top-level rule property in the `alert_subject_args`, `alert_text_args`, and `alert_text_kw fields`. However, if the matched document has a key with the same name, that will take preference over the rule property.

By default:

```
body                  = rule_name

                        [alert_text]

                        ruletype_text

                        {top_counts}

                        {field_values}
```

With `alert_text_type: alert_text_only`:

```
body                = rule_name

                      alert_text
```

With `alert_text_type: exclude_fields`:

```
body                = rule_name

                      [alert_text]

                      ruletype_text

                      {top_counts}
```

ruletype_text is the string returned by RuleType.get_match_str.

field_values will contain every key value pair included in the results from Elasticsearch. These fields include "@timestamp" (or the value of `timestamp_field`), every key in `included`, every key in `top_count_keys`, `query_key`, and `compare_key`. If the alert spans multiple events, these values may come from an individual event, usually the one which triggers the alert.

Command

The command alert allows you to execute an arbitrary command and pass arguments or stdin from the match. Arguments to the command can use Python format string syntax to access parts of the match. The alerter will open a subprocess and optionally pass the match, or matches in the case of an aggregated alert, as a JSON array, to the stdin of the process.

This alert requires one option:

`command`: A list of arguments to execute or a string to execute. If in list format, the first argument is the name of the program to execute. If passed a string, the command is executed through the shell.

Strings can be formatted using the old-style format (`%`) or the new-style format (`.format()`). When the old-style format is used, fields are accessed using `%(field_name)s`. When the new-style format is used, fields are accessed using `{match[field_name]}`. New-style formatting allows accessing nested fields (e.g., `{match[field_1_name][field_2_name]}`).

In an aggregated alert, these fields come from the first match.

Optional:

`new_style_string_format`: If True, arguments are formatted using `.format()` rather than `%`. The default is False.

`pipe_match_json`: If true, the match will be converted to JSON and passed to stdin of the command. Note that this will cause ElastAlert to block until the command exits or sends an EOF to stdout.

Example usage using old-style format:

```
alert:
  - command
command: ["/bin/send_alert", "--username", "%(username)s"]
```

Warning: Executing commmands with untrusted data can make it vulnerable to shell injection! If you use formatted data in your command, it is highly recommended that you use a args list format instead of a shell string.

Example usage using new-style format:

```
alert:
  - command
command: ["/bin/send_alert", "--username", "{match[username]}"]
```

Email

This alert will send an email. It connects to an smtp server located at `smtp_host`, or localhost by default. If available, it will use STARTTLS.

This alert requires one additional option:

`email`: An address or list of addresses to sent the alert to.

Optional:

`email_from_field`: Use a field from the document that triggered the alert as the recipient. If the field cannot be found, the `email` value will be used as a default. Note that this field will not be available in every rule type, for example, if you have `use_count_query` or if it's `type: flatline`. You can optionally add a domain suffix to the field to generate the address using `email_add_domain`. It can be a single recipient or list of recipients. For example, with the following settings:

```
email_from_field: "data.user"
email_add_domain: "@example.com"
```

and a match `{"@timestamp": "2017", "data": {"foo": "bar", "user": "qlo"}}`

an email would be sent to `qlo@example.com`

`smtp_host`: The SMTP host to use, defaults to localhost.

`smtp_port`: The port to use. Default is 25.

`smtp_ssl`: Connect the SMTP host using TLS, defaults to `false`. If `smtp_ssl` is not used, ElastAlert will still attempt STARTTLS.

`smtp_auth_file`: The path to a file which contains SMTP authentication credentials. It should be YAML formatted and contain two fields, `user` and `password`. If this is not present, no authentication will be attempted.

`smtp_cert_file`: Connect the SMTP host using the given path to a TLS certificate file, default to `None`.

`smtp_key_file`: Connect the SMTP host using the given path to a TLS key file, default to `None`.

`email_reply_to`: This sets the Reply-To header in the email. By default, the from address is ElastAlert@ and the domain will be set by the smtp server.

`from_addr`: This sets the From header in the email. By default, the from address is ElastAlert@ and the domain will be set by the smtp server.

`cc`: This adds the CC emails to the list of recipients. By default, this is left empty.

`bcc`: This adds the BCC emails to the list of recipients but does not show up in the email message. By default, this is left empty.

Jira

The JIRA alerter will open a ticket on jira whenever an alert is triggered. You must have a service account for ElastAlert to connect with. The credentials of the service account are loaded from a separate file. The ticket number will be written to the alert pipeline, and if it is followed by an email alerter, a link will be included in the email.

This alert requires four additional options:

`jira_server`: The hostname of the JIRA server.

`jira_project`: The project to open the ticket under.

`jira_issuetype`: The type of issue that the ticket will be filed as. Note that this is case sensitive.

`jira_account_file`: The path to the file which contains JIRA account credentials.

For an example JIRA account file, see `example_rules/jira_acct.yaml`. The account file is also yaml formatted and must contain two fields:

`user`: The username.

`password`: The password.

Optional:

`jira_component`: The name of the component or components to set the ticket to. This can be a single string or a list of strings. This is provided for backwards compatibility and will eventually be deprecated. It is preferable to use the plural `jira_components` instead.

`jira_components`: The name of the component or components to set the ticket to. This can be a single string or a list of strings.

`jira_description`: Similar to `alert_text`, this text is prepended to the JIRA description.

`jira_label`: The label or labels to add to the JIRA ticket. This can be a single string or a list of strings. This is provided for backwards compatibility and will eventually be deprecated. It is preferable to use the plural `jira_labels` instead.

`jira_labels`: The label or labels to add to the JIRA ticket. This can be a single string or a list of strings.

`jira_priority`: The index of the priority to set the issue to. In the JIRA dropdown for priorities, 0 would represent the first priority, 1 the 2nd, etc.

`jira_watchers`: A list of user names to add as watchers on a JIRA ticket. This can be a single string or a list of strings.

`jira_bump_tickets`: If true, ElastAlert search for existing tickets newer than `jira_max_age` and comment on the ticket with information about the alert instead of opening another ticket. ElastAlert finds the existing ticket by searching by summary. If the summary has changed or contains special characters, it may fail to find the ticket. If you are using a custom `alert_subject`, the two summaries must be exact matches, except by setting `jira_ignore_in_title`, you can ignore the value of a field when searching. For example, if the custom subject is "foo occured at bar", and "foo" is the value field X in the match, you can set `jira_ignore_in_title` to "X" and it will only bump tickets with "bar" in the subject. Defaults to false.

`jira_ignore_in_title`: ElastAlert will attempt to remove the value for this field from the JIRA subject when searching for tickets to bump. See `jira_bump_tickets` description above for an example.

`jira_max_age`: If `jira_bump_tickets` is true, the maximum age of a ticket, in days, such that ElastAlert will comment on the ticket instead of opening a new one. Default is 30 days.

`jira_bump_not_in_statuses`: If `jira_bump_tickets` is true, a list of statuses the ticket must **not** be in for ElastAlert to comment on the ticket instead of opening a new one. For example, to prevent comments being added to resolved or closed tickets, set this to 'Resolved' and 'Closed'. This option should not be set if the `jira_bump_in_statuses` option is set.

Example usage:

```
jira_bump_not_in_statuses:
  - Resolved
  - Closed
```

`jira_bump_in_statuses`: If `jira_bump_tickets` is true, a list of statuses the ticket *must be in* for ElastAlert to comment on the ticket instead of opening a new one. For example, to only comment on 'Open' tickets – and thus not 'In Progress', 'Analyzing', 'Resolved', etc. tickets – set this to 'Open'. This option should not be set if the `jira_bump_not_in_statuses` option is set.

Example usage:

```
jira_bump_in_statuses:
  - Open
```

`jira_bump_after_inactivity`: If this is set, ElastAlert will only comment on tickets that have been inactive for at least this many days. It only applies if `jira_bump_tickets` is true. Default is 0 days.

Arbitrary Jira fields:

ElastAlert supports setting any arbitrary JIRA field that your jira issue supports. For example, if you had a custom field, called "Affected User", you can set it by providing that field name in `snake_case` prefixed with `jira_`. These fields can contain primitive strings or arrays of strings. Note that when you create a custom field in your JIRA server, internally, the field is represented as `customfield_1111`. In elastalert, you may refer to either the public facing name OR the internal representation.

Example usage:

```
jira_arbitrary_singular_field: My Name
jira_arbitrary_multivalue_field:
  - Name 1
  - Name 2
jira_customfield_12345: My Custom Value
jira_customfield_9999:
  - My Custom Value 1
  - My Custom Value 2
```

OpsGenie

OpsGenie alerter will create an alert which can be used to notify Operations people of issues or log information. An OpsGenie `API` integration must be created in order to acquire the necessary `opsgenie_key` rule variable. Currently the OpsGenieAlerter only creates an alert, however it could be extended to update or close existing alerts.

It is necessary for the user to create an OpsGenie Rest HTTPS API integration page in order to create alerts.

The OpsGenie alert requires one option:

`opsgenie_key`: The randomly generated API Integration key created by OpsGenie.

Optional:

`opsgenie_account`: The OpsGenie account to integrate with.

`opsgenie_recipients`: A list OpsGenie recipients who will be notified by the alert.

`opsgenie_teams`: A list of OpsGenie teams to notify (useful for schedules with escalation).

`opsgenie_tags`: A list of tags for this alert.

opsgenie_message: Set the OpsGenie message to something other than the rule name. The message can be formatted with fields from the first match e.g. "Error occurred for {app_name} at {timestamp}.".

opsgenie_alias: Set the OpsGenie alias. The alias can be formatted with fields from the first match e.g "{app_name} error".

SNS

The SNS alerter will send an SNS notification. The body of the notification is formatted the same as with other alerters. The SNS alerter uses boto3 and can use credentials in the rule yaml, in a standard AWS credential and config files, or via environment variables. See http://docs.aws.amazon.com/cli/latest/userguide/cli-chap-getting-started.html for details.

SNS requires one option:

sns_topic_arn: The SNS topic's ARN. For example, arn:aws:sns:us-east-1:123456789:somesnstopic

Optional:

aws_access_key: An access key to connect to SNS with.

aws_secret_key: The secret key associated with the access key.

aws_region: The AWS region in which the SNS resource is located. Default is us-east-1

profile: The AWS profile to use. If none specified, the default will be used.

HipChat

HipChat alerter will send a notification to a predefined HipChat room. The body of the notification is formatted the same as with other alerters.

The alerter requires the following two options:

hipchat_auth_token: The randomly generated notification token created by HipChat. Go to https://XXXXX.hipchat.com/account/api and use 'Create new token' section, choosing 'Send notification' in Scopes list.

hipchat_room_id: The id associated with the HipChat room you want to send the alert to. Go to https://XXXXX.hipchat.com/rooms and choose the room you want to post to. The room ID will be the numeric part of the URL.

hipchat_msg_color: The color of the message background that is sent to HipChat. May be set to green, yellow or red. Default is red.

hipchat_domain: The custom domain in case you have HipChat own server deployment. Default is api.hipchat.com.

hipchat_ignore_ssl_errors: Ignore TLS errors (self-signed certificates, etc.). Default is false.

hipchat_proxy: By default ElastAlert will not use a network proxy to send notifications to HipChat. Set this option using hostname:port if you need to use a proxy.

hipchat_notify: When set to true, triggers a hipchat bell as if it were a user. Default is true.

hipchat_from: When humans report to hipchat, a timestamp appears next to their name. For bots, the name is the name of the token. The from, instead of a timestamp, defaults to empty unless set, which you can do here. This is optional.

hipchat_message_format: Determines how the message is treated by HipChat and rendered inside HipChat applications html - Message is rendered as HTML and receives no special treatment. Must be valid HTML and entities must be escaped (e.g.: '&' instead of '&'). May contain basic tags: a, b, i, strong, em, br, img, pre, code,

lists, tables. text - Message is treated just like a message sent by a user. Can include @mentions, emoticons, pastes, and auto-detected URLs (Twitter, YouTube, images, etc). Valid values: html, text. Defaults to 'html'.

MS Teams

MS Teams alerter will send a notification to a predefined Microsoft Teams channel.

The alerter requires the following options:

`ms_teams_webhook_url`: The webhook URL that includes your auth data and the ID of the channel you want to post to. Go to the Connectors menu in your channel and configure an Incoming Webhook, then copy the resulting URL. You can use a list of URLs to send to multiple channels.

`ms_teams_alert_summary`: Summary should be configured according to MS documentation, although it seems not displayed by Teams currently.

Optional:

`ms_teams_theme_color`: By default the alert will be posted without any color line. To add color, set this attribute to a HTML color value e.g. `#ff0000` for red.

`ms_teams_proxy`: By default ElastAlert will not use a network proxy to send notifications to MS Teams. Set this option using `hostname:port` if you need to use a proxy.

`ms_teams_alert_fixed_width`: By default this is `False` and the notification will be sent to MS Teams as-is. Teams supports a partial Markdown implementation, which means asterisk, underscore and other characters may be interpreted as Markdown. Currenlty, Teams does not fully implement code blocks. Setting this attribute to `True` will enable line by line code blocks. It is recommended to enable this to get clearer notifications in Teams.

Slack

Slack alerter will send a notification to a predefined Slack channel. The body of the notification is formatted the same as with other alerters.

The alerter requires the following option:

`slack_webhook_url`: The webhook URL that includes your auth data and the ID of the channel (room) you want to post to. Go to the Incoming Webhooks section in your Slack account https://XXXXX.slack.com/services/new/incoming-webhook , choose the channel, click 'Add Incoming Webhooks Integration' and copy the resulting URL. You can use a list of URLs to send to multiple channels.

Optional:

`slack_username_override`: By default Slack will use your username when posting to the channel. Use this option to change it (free text).

`slack_channel_override`: Incoming webhooks have a default channel, but it can be overridden. A public channel can be specified "#other-channel", and a Direct Message with "@username".

`slack_emoji_override`: By default ElastAlert will use the :ghost: emoji when posting to the channel. You can use a different emoji per ElastAlert rule. Any Apple emoji can be used, see http://emojipedia.org/apple/ . If slack_icon_url_override parameter is provided, emoji is ignored.

`slack_icon_url_override`: By default ElastAlert will use the :ghost: emoji when posting to the channel. You can provide icon_url to use custom image. Provide absolute address of the pciture, for example: http://some.address.com/image.jpg .

`slack_msg_color`: By default the alert will be posted with the 'danger' color. You can also use 'good' or 'warning' colors.

slack_proxy: By default ElastAlert will not use a network proxy to send notifications to Slack. Set this option using hostname:port if you need to use a proxy.

Telegram

Telegram alerter will send a notification to a predefined Telegram username or channel. The body of the notification is formatted the same as with other alerters.

The alerter requires the following two options:

telegram_bot_token: The token is a string along the lines of 110201543:AAHdqTcvCH1vGWJxfSeofSAs0K5PALDsaw that will be required to authorize the bot and send requests to the Bot API. You can learn about obtaining tokens and generating new ones in this document https://core.telegram.org/bots#botfather

telegram_room_id: Unique identifier for the target chat or username of the target channel (in the format @channelusername)

Optional:

telegram_api_url: Custom domain to call Telegram Bot API. Default to api.telegram.org

telegram_proxy: By default ElastAlert will not use a network proxy to send notifications to Telegram. Set this option using hostname:port if you need to use a proxy.

PagerDuty

PagerDuty alerter will trigger an incident to a predefined PagerDuty service. The body of the notification is formatted the same as with other alerters.

The alerter requires the following option:

pagerduty_service_key: Integration Key generated after creating a service with the 'Use our API directly' option at Integration Settings

pagerduty_client_name: The name of the monitoring client that is triggering this event.

Optional:

alert_subject: If set, this will be used as the Incident description within PagerDuty. If not set, ElastAlert will default to using the rule name of the alert for the incident.

alert_subject_args: If set, and alert_subject is a formattable string, ElastAlert will format the incident key based on the provided array of fields from the rule or match.

pagerduty_incident_key: If not set PagerDuty will trigger a new incident for each alert sent. If set to a unique string per rule PagerDuty will identify the incident that this event should be applied. If there's no open (i.e. unresolved) incident with this key, a new one will be created. If there's already an open incident with a matching key, this event will be appended to that incident's log.

pagerduty_incident_key_args: If set, and pagerduty_incident_key is a formattable string, Elastalert will format the incident key based on the provided array of fields from the rule or match.

pagerduty_proxy: By default ElastAlert will not use a network proxy to send notifications to PagerDuty. Set this option using hostname:port if you need to use a proxy.

Exotel

Developers in India can use Exotel alerter, it will trigger an incident to a mobile phone as sms from your exophone. Alert name along with the message body will be sent as an sms.

The alerter requires the following option:

`exotel_accout_sid`: This is sid of your Exotel account.

`exotel_auth_token`: Auth token assosiated with your Exotel account.

If you don't know how to find your accound sid and auth token, refer - http://support.exotel.in/support/solutions/articles/3000023019-how-to-find-my-exotel-token-and-exotel-sid-

`exotel_to_number`: The phone number where you would like send the notification.

`exotel_from_number`: Your exophone number from which message will be sent.

The alerter has one optional argument:

`exotel_message_body`: Message you want to send in the sms, is you don't specify this argument only the rule name is sent

Twilio

Twilio alerter will trigger an incident to a mobile phone as sms from your twilio phone number. Alert name will arrive as sms once this option is chosen.

The alerter requires the following option:

`twilio_account_sid`: This is sid of your twilio account.

`twilio_auth_token`: Auth token assosiated with your twilio account.

`twilio_to_number`: The phone number where you would like send the notification.

`twilio_from_number`: Your twilio phone number from which message will be sent.

VictorOps

VictorOps alerter will trigger an incident to a predefined VictorOps routing key. The body of the notification is formatted the same as with other alerters.

The alerter requires the following options:

`victorops_api_key`: API key generated under the 'REST Endpoint' in the Integrations settings.

`victorops_routing_key`: VictorOps routing key to route the alert to.

`victorops_message_type`: VictorOps field to specify severity level. Must be one of the following: INFO, WARNING, ACKNOWLEDGEMENT, CRITICAL, RECOVERY

Optional:

`victorops_entity_display_name`: Human-readable name of alerting entity. Used by VictorOps to correlate incidents by host througout the alert lifecycle.

`victorops_proxy`: By default ElastAlert will not use a network proxy to send notifications to VictorOps. Set this option using `hostname:port` if you need to use a proxy.

Gitter

Gitter alerter will send a notification to a predefined Gitter channel. The body of the notification is formatted the same as with other alerters.

The alerter requires the following option:

gitter_webhook_url: The webhook URL that includes your auth data and the ID of the channel (room) you want to post to. Go to the Integration Settings of the channel https://gitter.im/ORGA/CHANNEL#integrations , click 'CUSTOM' and copy the resulting URL.

Optional:

gitter_msg_level: By default the alert will be posted with the 'error' level. You can use 'info' if you want the messages to be black instead of red.

gitter_proxy: By default ElastAlert will not use a network proxy to send notifications to Gitter. Set this option using hostname:port if you need to use a proxy.

ServiceNow

The ServiceNow alerter will create a ne Incident in ServiceNow. The body of the notification is formatted the same as with other alerters.

The alerter requires the following options:

servicenow_rest_url: The ServiceNow RestApi url, this will look like https://instancename.service-now.com/api/now/v1/table/incident

username: The ServiceNow Username to access the api.

password: The ServiceNow password to access the api.

short_description: The ServiceNow password to access the api.

comments: Comments to be attached to the incident, this is the equivilant of work notes.

assignment_group: The group to assign the incident to.

category: The category to attach the incident to, use an existing category.

subcategory: The subcategory to attach the incident to, use an existing subcategory.

cmdb_ci: The configuration item to attach the incident to.

caller_id: The caller id (email address) of the user that created the incident (elastalert@somewhere.com).

Optional:

servicenow_proxy: By default ElastAlert will not use a network proxy to send notifications to ServiceNow. Set this option using hostname:port if you need to use a proxy.

Debug

The debug alerter will log the alert information using the Python logger at the info level. It is logged into a Python Logger object with the name elastalert that can be easily accessed using the getLogger command.

Stomp

This alert type will use the STOMP protocol in order to push a message to a broker like ActiveMQ or RabbitMQ. The message body is a JSON string containing the alert details. The default values will work with a pristine ActiveMQ installation.

Optional:

stomp_hostname: The STOMP host to use, defaults to localhost. stomp_hostport: The STOMP port to use, defaults to 61613. stomp_login: The STOMP login to use, defaults to admin. stomp_password: The

STOMP password to use, defaults to admin. `stomp_destination`: The STOMP destination to use, defaults to /queue/ALERT

The stomp_destination field depends on the broker, the /queue/ALERT example is the nomenclature used by ActiveMQ. Each broker has its own logic.

HTTP POST

This alert type will send results to a JSON endpoint using HTTP POST. The key names are configurable so this is compatible with almost any endpoint. By default, the JSON will contain al the items from the match, unless you specify http_post_payload, in which case it will only contain those items.

Required:

`http_post_url`: The URL to POST.

Optional:

`http_post_payload`: List of keys:values to use as the content of the POST. Example - ip:clientip will map the value from the clientip index of Elasticsearch to JSON key named ip. If not defined, all the Elasticsearch keys will be sent.

`http_post_static_payload`: Key:value pairs of static parameters to be sent, along with the Elasticsearch results. Put your authentication or other information here.

`http_post_proxy`: URL of proxy, if required.

`http_post_all_values`: Boolean of whether or not to include every key value pair from the match in addition to those in http_post_payload and http_post_static_payload. Defaults to True if http_post_payload is not specified, otherwise False.

Example usage:

```
alert: post
http_post_url: "http://example.com/api"
http_post_payload:
  ip: clientip
http_post_static_payload:
  apikey: abc123
```

Alerter

For all Alerter subclasses, you may reference values from a top-level rule property in your Alerter fields by referring to the property name surrounded by dollar signs. This can be useful when you have rule-level properties that you would like to reference many times in your alert. For example:

Example usage:

```
jira_priority: $priority$
jira_alert_owner: $owner$
```

CHAPTER 4

ElastAlert Metadata Index

ElastAlert uses Elasticsearch to store various information about its state. This not only allows for some level of auditing and debugging of ElastAlert's operation, but also to avoid loss of data or duplication of alerts when ElastAlert is shut down, restarted, or crashes. This cluster and index information is defined in the global config file with `es_host`, `es_port` and `writeback_index`. ElastAlert must be able to write to this index. The script, `elastalert-create-index` will create the index with the correct mapping for you, and optionally copy the documents from an existing ElastAlert writeback index. Run it and it will prompt you for the cluster information.

ElastAlert will create three different types of documents in the writeback index:

elastalert_status

`elastalert_status` is a log of the queries performed for a given rule and contains:

- `@timestamp`: The time when the document was uploaded to Elasticsearch. This is after a query has been run and the results have been processed.
- `rule_name`: The name of the corresponding rule.
- `starttime`: The beginning of the timestamp range the query searched.
- `endtime`: The end of the timestamp range the query searched.
- `hits`: The number of results from the query.
- `matches`: The number of matches that the rule returned after processing the hits. Note that this does not necessarily mean that alerts were triggered.
- `time_taken`: The number of seconds it took for this query to run.

`elastalert_status` is what ElastAlert will use to determine what time range to query when it first starts to avoid duplicating queries. For each rule, it will start querying from the most recent endtime. If ElastAlert is running in debug mode, it will still attempt to base its start time by looking for the most recent search performed, but it will not write the results of any query back to Elasticsearch.

elastalert

`elastalert` is a log of information about every alert triggered and contains:

- `@timestamp`: The time when the document was uploaded to Elasticsearch. This is not the same as when the alert was sent, but rather when the rule outputs a match.
- `rule_name`: The name of the corresponding rule.
- `alert_info`: This contains the output of Alert.get_info, a function that alerts implement to give some relevant context to the alert type. This may contain alert_info.type, alert_info.recipient, or any number of other sub fields.
- `alert_sent`: A boolean value as to whether this alert was actually sent or not. It may be false in the case of an exception or if it is part of an aggregated alert.
- `alert_time`: The time that the alert was or will be sent. Usually, this is the same as @timestamp, but may be some time in the future, indicating when an aggregated alert will be sent.
- `match_body`: This is the contents of the match dictionary that is used to create the alert. The subfields may include a number of things containing information about the alert.
- `alert_exception`: This field is only present when the alert failed because of an exception occurring, and will contain the exception information.
- `aggregate_id`: This field is only present when the rule is configured to use aggregation. The first alert of the aggregation period will contain an alert_time set to the aggregation time into the future, and subsequent alerts will contain the document ID of the first. When the alert_time is reached, all alerts with that aggregate_id will be sent together.

elastalert_error

When an error occurs in ElastAlert, it is written to both Elasticsearch and to stderr. The `elastalert_error` type contains:

- `@timestamp`: The time when the error occurred.
- `message`: The error or exception message.
- `traceback`: The traceback from when the error occurred.
- `data`: Extra information about the error. This often contains the name of the rule which caused the error.

silence

`silence` is a record of when alerts for a given rule will be suppressed, either because of a `realert` setting or from using –silence. When an alert with `realert` is triggered, a `silence` record will be written with `until` set to the alert time plus `realert`.

- `@timestamp`: The time when the document was uploaded to Elasticsearch.
- `rule_name`: The name of the corresponding rule.
- `until`: The timestamp when alerts will begin being sent again.
- `exponent`: The exponential factor which multiplies `realert`. The length of this silence is equal to `realert` * 2**exponent. This will be 0 unless `exponential_realert` is set.

Whenever an alert is triggered, ElastAlert will check for a matching `silence` document, and if the `until` timestamp is in the future, it will ignore the alert completely. See the *Running ElastAlert* section for information on how to silence an alert.

CHAPTER 5

Adding a New Rule Type

This document describes how to create a new rule type. Built in rule types live in `elastalert/ruletypes.py` and are subclasses of `RuleType`. At the minimum, your rule needs to implement `add_data`.

Your class may implement several functions from `RuleType`:

```
class AwesomeNewRule(RuleType):
    # ...
    def add_data(self, data):
        # ...
    def get_match_str(self, match):
        # ...
    def garbage_collect(self, timestamp):
        # ...
```

You can import new rule types by specifying the type as `module.file.RuleName`, where module is the name of a Python module, or folder containing `__init__.py`, and file is the name of the Python file containing a `RuleType` subclass named `RuleName`.

Basics

The `RuleType` instance remains in memory while ElastAlert is running, receives data, keeps track of its state, and generates matches. Several important member properties are created in the `__init__` method of `RuleType`:

`self.rules`: This dictionary is loaded from the rule configuration file. If there is a `timeframe` configuration option, this will be automatically converted to a `datetime.timedelta` object when the rules are loaded.

`self.matches`: This is where ElastAlert checks for matches from the rule. Whatever information is relevant to the match (generally coming from the fields in Elasticsearch) should be put into a dictionary object and added to `self.matches`. ElastAlert will pop items out periodically and send alerts based on these objects. It is recommended that you use `self.add_match(match)` to add matches. In addition to appending to `self.matches`, `self.add_match` will convert the datetime `@timestamp` back into an ISO8601 timestamp.

`self.required_options`: This is a set of options that must exist in the configuration file. ElastAlert will ensure that all of these fields exist before trying to instantiate a `RuleType` instance.

add_data(self, data):

When ElastAlert queries Elasticsearch, it will pass all of the hits to the rule type by calling `add_data`. `data` is a list of dictionary objects which contain all of the fields in `include`, `query_key` and `compare_key` if they exist, and `@timestamp` as a datetime object. They will always come in chronological order sorted by '@timestamp'.

get_match_str(self, match):

Alerts will call this function to get a human readable string about a match for an alert. Match will be the same object that was added to `self.matches`, and `rules` the same as `self.rules`. The `RuleType` base implementation will return an empty string. Note that by default, the alert text will already contain the key-value pairs from the match. This should return a string that gives some information about the match in the context of this specific RuleType.

garbage_collect(self, timestamp):

This will be called after ElastAlert has run over a time period ending in `timestamp` and should be used to clear any state that may be obsolete as of `timestamp`. `timestamp` is a datetime object.

Tutorial

As an example, we are going to create a rule type for detecting suspicious logins. Let's imagine the data we are querying is login events that contains IP address, username and a timestamp. Our configuration will take a list of usernames and a time range and alert if a login occurs in the time range. First, let's create a modules folder in the base ElastAlert folder:

```
$ mkdir elastalert_modules
$ cd elastalert_modules
$ touch __init__.py
```

Now, in a file named `my_rules.py`, add

```
import dateutil.parser

from elastalert.ruletypes import RuleType

# elastalert.util includes useful utility functions
# such as converting from timestamp to datetime obj
from elastalert.util import ts_to_dt

class AwesomeRule(RuleType):

    # By setting required_options to a set of strings
    # You can ensure that the rule config file specifies all
    # of the options. Otherwise, ElastAlert will throw an exception
    # when trying to load the rule.
    required_options = set(['time_start', 'time_end', 'usernames'])
```

```
    # add_data will be called each time Elasticsearch is queried.
    # data is a list of documents from Elasticsearch, sorted by timestamp,
    # including all the fields that the config specifies with "include"
    def add_data(self, data):
        for document in data:

            # To access config options, use self.rules
            if document['username'] in self.rules['usernames']:

                # Convert the timestamp to a time object
                login_time = document['@timestamp'].time()

                # Convert time_start and time_end to time objects
                time_start = dateutil.parser.parse(self.rules['time_start']).time()
                time_end = dateutil.parser.parse(self.rules['time_end']).time()

                # If the time falls between start and end
                if login_time > time_start and login_time < time_end:

                    # To add a match, use self.add_match
                    self.add_match(document)

    # The results of get_match_str will appear in the alert text
    def get_match_str(self, match):
        return "%s logged in between %s and %s" % (match['username'],
                                                   self.rules['time_start'],
                                                   self.rules['time_end'])

    # garbage_collect is called indicating that ElastAlert has already been run up to␣
↪timestamp
    # It is useful for knowing that there were no query results from Elasticsearch␣
↪because
    # add_data will not be called with an empty list
    def garbage_collect(self, timestamp):
        pass
```

In the rule configuration file, `example_rules/example_login_rule.yaml`, we are going to specify this rule by writing

```
name: "Example login rule"
es_host: elasticsearch.example.com
es_port: 14900
type: "elastalert_modules.my_rules.AwesomeRule"
# Alert if admin, userXYZ or foobaz log in between 8 PM and midnight
time_start: "20:00"
time_end: "24:00"
usernames:
- "admin"
- "userXYZ"
- "foobaz"
# We require the username field from documents
include:
- "username"
alert:
- debug
```

ElastAlert will attempt to import the rule with `from elastalert_modules.my_rules import`

`AwesomeRule`. This means that the folder must be in a location where it can be imported as a Python module.

An alert from this rule will look something like:

```
Example login rule

userXYZ logged in between 20:00 and 24:00

@timestamp: 2015-03-02T22:23:24Z
username: userXYZ
```

CHAPTER 6

Adding a New Alerter

Alerters are subclasses of `Alerter`, found in `elastalert/alerts.py`. They are given matches and perform some action based on that. Your alerter needs to implement two member functions, and will look something like this:

```
class AwesomeNewAlerter(Alerter):
    required_options = set(['some_config_option'])
    def alert(self, matches):
        ...
    def get_info(self):
        ...
```

You can import alert types by specifying the type as `module.file.AlertName`, where module is the name of a python module, and file is the name of the python file containing a `Alerter` subclass named `AlertName`.

Basics

The alerter class will be instantiated when ElastAlert starts, and be periodically passed matches through the `alert` method. ElastAlert also writes back info about the alert into Elasticsearch that it obtains through `get_info`. Several important member properties:

`self.required_options`: This is a set containing names of configuration options that must be present. ElastAlert will not instantiate the alert if any are missing.

`self.rule`: The dictionary containing the rule configuration. All options specific to the alert should be in the rule configuration file and can be accessed here.

`self.pipeline`: This is a dictionary object that serves to transfer information between alerts. When an alert is triggered, a new empty pipeline object will be created and each alerter can add or receive information from it. Note that alerters are called in the order they are defined in the rule file. For example, the JIRA alerter will add its ticket number to the pipeline and the email alerter will add that link if it's present in the pipeline.

alert(self, match):

ElastAlert will call this function to send an alert. `matches` is a list of dictionary objects with information about the match. You can get a nice string representation of the match by calling `self.rule['type'].get_match_str(match, self.rule)`. If this method raises an exception, it will be caught by ElastAlert and the alert will be marked as unsent and saved for later.

get_info(self):

This function is called to get information about the alert to save back to Elasticsearch. It should return a dictionary, which is uploaded directly to Elasticsearch, and should contain useful information about the alert such as the type, recipients, parameters, etc.

Tutorial

Let's create a new alert that will write alerts to a local output file. First, create a modules folder in the base ElastAlert folder:

```
$ mkdir elastalert_modules
$ cd elastalert_modules
$ touch __init__.py
```

Now, in a file named `my_alerts.py`, add

```
from elastalert.alerts import Alerter, BasicMatchString

class AwesomeNewAlerter(Alerter):

    # By setting required_options to a set of strings
    # You can ensure that the rule config file specifies all
    # of the options. Otherwise, ElastAlert will throw an exception
    # when trying to load the rule.
    required_options = set(['output_file_path'])

    # Alert is called
    def alert(self, matches):

        # Matches is a list of match dictionaries.
        # It contains more than one match when the alert has
        # the aggregation option set
        for match in matches:

            # Config options can be accessed with self.rule
            with open(self.rule['output_file_path'], "a") as output_file:

                # basic_match_string will transform the match into the default
                # human readable string format
                match_string = str(BasicMatchString(self.rule, match))

                output_file.write(match_string)

    # get_info is called after an alert is sent to get data that is written back
    # to Elasticsearch in the field "alert_info"
```

```
    # It should return a dict of information relevant to what the alert does
    def get_info(self):
        return {'type': 'Awesome Alerter',
                'output_file': self.rule['output_file_path']}
```

In the rule configuration file, we are going to specify the alert by writing

```
alert: "elastalert_modules.my_alerts.AwesomeNewAlerter"
output_file_path: "/tmp/alerts.log"
```

ElastAlert will attempt to import the alert with `from elastalert_modules.my_alerts import AwesomeNewAlerter`. This means that the folder must be in a location where it can be imported as a python module.

CHAPTER 7

Writing Filters For Rules

This document describes how to create a filter section for your rule config file.

The filters used in rules are part of the Elasticsearch query DSL, further documentation for which can be found at https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl.html This document contains a small subset of particularly useful filters.

The filter section is passed to Elasticsearch exactly as follows:

```
filter:
  and:
    filters:
      - [filters from rule.yaml]
```

Every result that matches these filters will be passed to the rule for processing.

Common Filter Types:

query_string

The query_string type follows the Lucene query format and can be used for partial or full matches to multiple fields. See http://lucene.apache.org/core/2_9_4/queryparsersyntax.html for more information:

```
filter:
- query:
    query_string:
      query: "username: bob"
- query:
    query_string:
      query: "_type: login_logs"
- query:
    query_string:
      query: "field: value OR otherfield: othervalue"
- query:
```

```
    query_string:
       query: "this: that AND these: those"
```

term

The term type allows for exact field matches:

```
filter:
- term:
    name_field: "bob"
- term:
    _type: "login_logs"
```

Note that a term query may not behave as expected if a field is analyzed. By default, many string fields will be tokenized by whitespace, and a term query for "foo bar" may not match a field that appears to have the value "foo bar", unless it is not analyzed. Conversely, a term query for "foo" will match analyzed strings "foo bar" and "foo baz". For full text matching on analyzed fields, use query_string. See https://www.elastic.co/guide/en/elasticsearch/guide/current/term-vs-full-text.html

terms

Terms allows for easy combination of multiple term filters:

```
filter:
- terms:
    field: ["value1", "value2"]
```

Using the minimum_should_match option, you can define a set of term filters of which a certain number must match:

```
- terms:
    fieldX: ["value1", "value2"]
    fieldY: ["something", "something_else"]
    fieldZ: ["foo", "bar", "baz"]
    minimum_should_match: 2
```

wildcard

For wildcard matches:

```
filter:
- query:
    wildcard:
      field: "foo*bar"
```

range

For ranges on fields:

```
filter:
- range:
    status_code:
```

```
    from: 500
    to: 599
```

Negation, and, or

Any of the filters can be embedded in `not`, `and`, and `or`:

```
filter:
- or:
    - term:
        field: "value"
    - wildcard:
        field: "foo*bar"
    - and:
        - not:
            term:
              field: "value"
        - not:
            term:
              _type: "something"
```

Loading Filters Directly From Kibana 3

There are two ways to load filters directly from a Kibana 3 dashboard. You can set your filter to:

```
filter:
  download_dashboard: "My Dashboard Name"
```

and when ElastAlert starts, it will download the dashboard schema from Elasticsearch and use the filters from that. However, if the dashboard name changes or if there is connectivity problems when ElastAlert starts, the rule will not load and ElastAlert will exit with an error like "Could not download filters for .."

The second way is to generate a config file once using the Kibana dashboard. To do this, run `elastalert-rule-from-kibana`.

```
$ elastalert-rule-from-kibana
Elasticsearch host: elasticsearch.example.com
Elasticsearch port: 14900
Dashboard name: My Dashboard

Partial Config file
-----------

name: My Dashboard
es_host: elasticsearch.example.com
es_port: 14900
filter:
- query:
    query_string: {query: '_exists_:log.message'}
- query:
    query_string: {query: 'some_field:12345'}
```

CHAPTER 8

Enhancements

Enhancements are modules which let you modify a match before an alert is sent. They should subclass `BaseEnhancement`, found in `elastalert/enhancements.py`. They can be added to rules using the `match_enhancements` option:

```
match_enhancements:
- module.file.MyEnhancement
```

where module is the name of a Python module, or folder containing `__init__.py`, and file is the name of the Python file containing a `BaseEnhancement` subclass named `MyEnhancement`.

A special exception class `` `DropMatchException` `` can be used in enhancements to drop matches if custom conditions are met. For example:

```
class MyEnhancement(BaseEnhancement):
    def process(self, match):
        # Drops a match if "field_1" == "field_2"
        if match['field_1'] == match['field_2']:
            raise DropMatchException()
```

Example

As an example enhancement, let's add a link to a whois website. The match must contain a field named domain and it will add an entry named domain_whois_link. First, create a modules folder for the enhancement in the ElastAlert directory.

```
$ mkdir elastalert_modules
$ cd elastalert_modules
$ touch __init__.py
```

Now, in a file named `my_enhancements.py`, add

```
from elastalert.enhancements import BaseEnhancement

class MyEnhancement(BaseEnhancement):

    # The enhancement is run against every match
    # The match is passed to the process function where it can be modified in any way
    # ElastAlert will do this for each enhancement linked to a rule
    def process(self, match):
        if 'domain' in match:
            url = "http://who.is/whois/%s" % (match['domain'])
            match['domain_whois_link'] = url
```

Enhancements will not automatically be run. Inside the rule configuration file, you need to point it to the enhancement(s) that it should run by setting the `match_enhancements` option:

```
match_enhancements:
- "elastalert_modules.my_enhancements.MyEnhancement"
```

CHAPTER 9

Signing requests to Amazon Elasticsearch service

When using Amazon Elasticsearch service, you need to secure your Elasticsearch from the outside. Currently, there is no way to secure your Elasticsearch using network firewall rules, so the only way is to signing the requests using the access key and secret key for a role or user with permissions on the Elasticsearch service.

You can sign requests to AWS using any of the standard AWS methods of providing credentials. - Environment Variables, `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` - AWS Config or Credential Files, `~/.aws/config` and `~/.aws/credentials` - AWS Instance Profiles, uses the EC2 Metadata service

Using an Instance Profile

Typically, you'll deploy ElastAlert on a running EC2 instance on AWS. You can assign a role to this instance that gives it permissions to read from and write to the Elasticsearch service. When using an Instance Profile, you will need to specify the `aws_region` in the configuration file or set the `AWS_DEFAULT_REGION` environment variable.

Using AWS profiles

You can also create a user with permissions on the Elasticsearch service and tell ElastAlert to authenticate itself using that user. First, create an AWS profile in the machine where you'd like to run ElastAlert for the user with permissions.

You can use the environment variables `AWS_DEFAULT_PROFILE` and `AWS_DEFAULT_REGION` or add two options to the configuration file: - `aws_region`: The AWS region where you want to operate. - `profile`: The name of the AWS profile to use to sign the requests.

CHAPTER 10

Indices and Tables

- genindex
- modindex
- search