alerta Documentation

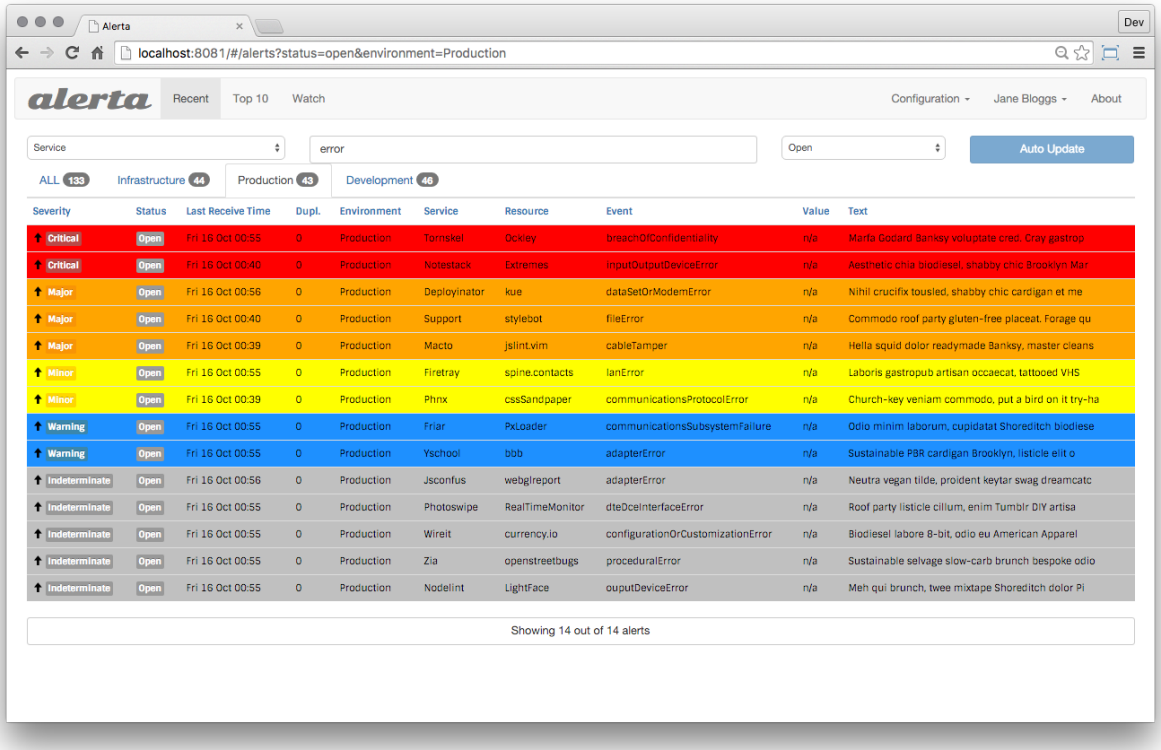
Release 6.0

Nick Satterly

Oct 02, 2018

Contents

The alerta monitoring system is a tool used to consolidate and de-duplicate alerts from multiple sources for quick 'at-a-glance' visualisation. With just one system you can monitor alerts from many other monitoring tools on a single screen.



Alerta combines a JSON API *server* for receiving, processing and rendering alerts with a simple, yet effective *Alerta Web UI* and *command-line tool*. There are numerous *integrations* with popular monitoring tools and it is easy to add your own using the *API* directly, the *Python SDK* or the same command-line tool to *send alerts*. Access to the API and command-line tool can be restricted using *API keys* and to the web console using *Basic Auth* or *OAuth2* providers Google, GitHub and GitLab.

Get started today!

CHAPTER 1

Demo Sites

There are two public web consoles available for demonstration and testing:

- https://try.alerta.io (Google OAuth)
- https://alerta.herokuapp.com (BasicAuth)

The web consoles are powered by a single public API which can be used as a sandbox for integration testing:

- https://alerta-api.herokuapp.com

The "API Explorer" can be used to query for and send alerts to the public API server:

- https://explorer.alerta.io

The `alerta` command-line tool can also be used to generate alerts. The required API key is `demo-key`.

1.1 Quickstart

This is a quick-start guide that will get you running Alerta in under 5 minutes.

1.1.1 Install MongoDB

For Debian/Ubuntu, run:

```
$ sudo apt-get install -y mongodb-org
$ mongod
```

If `apt-get` can't locate the "mongodb-org" metapackage package then follow these steps to add MongoDB package repository to apt sources list.

For other operating systems, see the installation steps on the MongoDB web site.

1.1.2 Install the Alerta Server

To install the alerta server:

```
$ pip install alerta-server
$ alertad run --port 8080
```

You should see something like:

```
* Running on http://127.0.0.1:8080/ (Press CTRL+C to quit)
```

1.1.3 Install the Web Console

To install the web console:

```
$ wget -O alerta-web.tgz https://github.com/alerta/angular-alerta-webui/tarball/master
$ tar zxvf alerta-web.tgz
$ cd alerta-angular-alerta-webui-*/app
$ python -m SimpleHTTPServer 8000

>> browse to http://localhost:8000
```

1.1.4 Send some alerts

To send an alert to the server:

```
$ alerta send -r web01 -e NodeDown -E Production -S Website -s major -t "Web server␣
↪is down." -v ERROR
```

The alert should appear almost immediately in the console. If it doesn't it's either a *CORS issues* or a bug.

1.1.5 What's next?

Take the *step-by-step tutorials* or dive straight into a *deployment*.

1.2 Design Principles

The following principles guided the design and development of the Alerta monitoring system.

1.2.1 Resource under alarm

A *resource* is any entity that it makes sense for you to receive alerts for. You shouldn't be forced to accept a certain "world view" when using a monitoring tool or to repurpose a "host" field for a service or application, or a even a URL. Host-centric monitoring tools belong in the 90's.

1.2.2 Many severity levels

You are free to use as many or as few as you like eg. if you plan to only integrate with Nagios then only use `critical`, `warning` and `ok`. If you are integrating with a fault management system for a telco you might want to use the six ISO perceived severity levels or alternatively, if you are pushing application alerts you might want to consider using the `debug` and `trace` severity levels.

1.2.3 Robust alert reception

In accordance with the robustness principle which is to "be liberal in what you accept from others", alerta will accept any alert as long as it meets the alert format specification. ie. no field values need to be defined in advance for it to be accepted, however the benefits of following a standard *convention* for such attributes as `environment`, `service`, `event` and `resource` (as internally defined by and useful to you) are many.

1.2.4 Self-clearing alerts

All alerts should have a corresponding `cleared` or `normal` state so that non-normal alerts can be automatically cleared down by the system. Where an alert cannot send a corresponding clear an alert should specify a `timeout` (or have a default assigned) after which it will be deleted.

1.2.5 Alerts are cheap

Alerts should be resent at regular intervals if they are still active which means that if all data is lost after a certain amount of time (eg. 2 hours?) you are back to where you were. This will be generally true though, for some alert sources this isn't possible eg. SNMP traps, log errors. Alerts in a normal state can be resent at a longer interval.

1.2.6 Tags and custom attributes

Dynamic 'scale up'/'scale down' environments are the defacto standard now; naming individual servers is lame. Use service discovery and dynamically generated metadata to tag alerts and assign custom attributes on the fly.

1.3 Server & API

The Alerta API receives alerts from multiple sources, *correlates*, *de-duplicates* or *suppresses* them, and makes the alerts available via a RESTful JSON API.

Alerts can be intercepted as they are received to modify, enhance or reject them using *pre-receive hooks*. Alerts can also be used to trigger actions in other systems after the alert has been processed using *post-receive hooks* or following an alert *status change* for bi-directional integration.

There are several *integrations* with popular monitoring tools available and *webhooks* can be used to trivially integrate with AWS Cloudwatch, Pingdom, PagerDuty and many more.

1.3.1 Event Processing

Alerta comes *out-of-the-box* with key features designed to reduce the burden of alert management. When an event is received it it is processed in the following way:

1. all plugin pre-receive hooks are run in alphabetical order, an alert is immediately rejected if any plugins return a `RejectException`
2. alert is checked against any active blackout periods, alert suppressed if any match
3. alert is checked if duplicate, if so duplicate count is increased and repeat set to *True*
4. alert is checked if correlated, if so change severity and/or status etc
5. alert is neither a duplicate or correlated so create new alert
6. all plugin post-receive hooks are run in alphabetical order
7. new or updated alert returned in response
8. timeout used to expire alerts from the console

Each of the above actions are explained in more detail in the following sections.

1.3.2 Plug-ins

Plug-ins are small python scripts that can run either before or after an alert is saved to the database, or before a status change update. This is achieved by registering *pre-receive hooks* for transformers, *post-receive hooks* for external notification and *status change hooks* for bi-directional integration.

Transformers

Using pre-receive hooks, plugins provide the ability to transform raw alert data from sources before alerts are created. For example, alerts can be *normalised* to ensure they all have specific attributes or tags or only have a specific value from a range of allowed values. This is demonstrated in the reject plugin that enforces an alert policy.

Plugins can also be used to *enhance* alerts – like the Geo location plugin which adds location data to alerts based on the remote IP address of the client, or the generic enhance plugin which adds a `customer` attribute based on information contained in the alert.

External Notification

Using post-receive hooks, plugin integrations can be used to provide downstream systems with alerts in realtime for external notification. For example, pushing alerts onto an AWS SNS topic, AMQP queue, logging to a Logstash/Kibana stack, or sending notifications to HipChat, Slack or Twilio and many more.

Bi-directional Integration

Using status change hooks, plugins can be used to complete a two way integration with an external system. That is, an external system like Prometheus Alertmanager that generates alerts that are forwarded to Alerta can be updated when the status of an alert changes in Alerta.

For example, if an operator "acknowledges" a Prometheus alert in the Alerta web UI then a status change hook could silence the corresponding alert in Alertmanager. This requires that external systems provide enough information in the alert created in Alerta for that alert to be uniquely identified at a later date.

1.3.3 Blackout Periods

An alert that is received during a blackout period is suppressed. That is, it is received by Alerta and a `202 Accepted` status code is returned however this means that even though the alert has been accepted, it won't be processed.

Alerta defines many different alert attributes that can be used to group alerts and it is these attributes that can be used to define blackout rules. For example, to suppress alerts from an entire environment, service or group, or a combination of these. However, it is possible to define blackout rules based only on resource and event attributes for situations that require that level of granularity.

Tags can also be used to define a blackout rule which should allow a lot of flexibility because tags can be added at source, using the `alerta` CLI, or using a plugin. Note that one or more tags can be required to match an alert for the suppression to apply.

In summary, blackout rules can be any of:

- an entire environment eg. `environment=Production`
- a particular resource eg. `resource=host55`
- an entire service eg. `service=Web`
- every occurrence of a specific event eg. `event=DiskFull`
- a group of events eg. `group=Syslog`
- a specific event for a resource eg. `resource=host55 and event=DiskFull`
- all events that have a specific set of tags eg. `tags=[ blackout, london ]`

Note that an `environment` is always required to be defined for a blackout rule.

1.3.4 De-Duplication

When an alert with the same `environment-resource-event` combination is received with the **same** `severity`, the alert is de-duplicated.

This means that information from the de-duplicated alert is used to update key attributes of the existing alert (like `duplicateCount`, `repeat` flag, `value`, `text` and `lastReceiveTime`) and the new alert is not shown.

Alerts are sorted in the Alerta web UI by `lastReceiveTime` by default so that the most recent alerts will be displayed at the top regardless of whether they were new alerts or de-duplicated alerts.

1.3.5 Simple Correlation

Alerta implements what we call "simple correlation" – as opposed to complex correlation which is much more involved. Simple correlation, in combination with de-duplication, provides straight-forward and effective ways to reduce the burden of managing an alert console.

With Alerta, there are two ways alerts can be correlated, namely:

1. When an alert with the same `environment-resource-event` combination is received with a **different** `severity`, then the alert is correlated.
2. When a alert with the same `environment-resource` combination is received with an `event` in the `correlate` list of related events with **any** severity, then the alert is correlated.

In both cases, this means that information from the correlated alert is used to update key attributes of the existing alert (like `severity`, `event`, `value`, `text` and `lastReceiveTime`) and the new alert is not shown.

1.3.6 State-based Browser

Alerta is called state-based because it will **automatically** *change the alert status* based on the current and previous severity of alerts and subsequent user actions.

The Alerta API will:

- only show the most recent state of any alert
- change the status of an alert to `closed` if a `normal`, `ok` or `cleared` is received
- change the status of a `closed` alert to `open` if the event reoccurs
- change the status of an `acknowledged` alert to `open` if the new severity is higher than the current `severity`
- update the `severity` and other key attributes of an alert when a more recent alert is received (see *correlation* and *deduplication*)
- update the `trendIndication` attribute based on `previousSeverity` and current `severity` with either `moreSevere`, `lessSevere` or `noChange`
- update the `history` log following a `severity` or `status` change (see *alert history*)

All of these automatic actions combine to ensure that important alerts are given the priority they deserve.

Note: To take full advantage of the state-based browser it is recommended to implement the timeout of `expired` alerts using the *House Keeping* script.

1.3.7 Alert History

Whenever an alert status or severity changes, that change is recorded in the alert *history* log. This is to allow operations staff follow the lifecycle of a particular alert, if necessary.

The alert history is visible in the *Alert Details* page of any alert and also by using the `alerta` command-line tool `history` sub-command.

For example, it will show whether an alert status change happened as a result of operator (external) action or an automatic *correlation* (auto) action.

1.3.8 Heartbeats

An Alerta *heartbeat* is a periodic HTTP request sent to the Alerta API to indicate normal operation of the origin of the heartbeat.

They can be used to ensure components of the Alerta monitoring system are operating normally or sent from any other source. As well as an `origin` they include a `timeout` in seconds (after which they will be considered stale), and optional `tags`.

They are visible in the Alerta console (*About* page) and via the `alerta` command-line tool using the `heartbeat` sub-command to send them, and the `heartbeats` sub-command to view them.

Alerts can be generated from stale heartbeats using `alerta heartbeats --alert`.

1.4 Alerta Web UI

The Alerta web UI console takes full advantage of the *state-based Alerta API* to ensure that the most important events at any given time are brought to the attention of operators.

1.4.1 Configuration

To configure the Alerta web UI modify "in place" the default `config.json` file that is supplied with the web application. It uses simple JSON syntax.

Note: The Alerta web UI before version 6.0 used an AngularJS configuration block for configuration settings which has now been deprecated.

The three main areas for configuration are:

- defining the Alerta API endpoint
- enforcing a use authentication strategy
- selecting colors for severity, highlighting, text and sound

The default web UI `config.json` configuration file is included below. It assumes that the Alerta API is running on the same host (but different port) that the web UI static html files are being served from (line 2):

```
1 {
2   "endpoint": "http://localhost:8080"
3 }
```

1.4.2 Configuration from API Server

Starting from version 6.0, client configuration is supplied by the API server. This includes configuration for the web UI and the command-line tool.

Configuration settings are made on the API server and when the web UI console is bootstrapping it reads the endpoint setting and downloads the rest of the configuration.

The remote configuration from the API server is merged with the local configuration settings to provide the final configuration used by clients.

Example

The following API server settings generate the JSON client configuration shown below that.

```
1 AUTH_PROVIDER = 'google'
2 AUTH_REQUIRED = True
3 CUSTOMER_VIEWS = True
4 GOOGLE_TRACKING_ID = 'UA-44644195-5'
5 OAUTH2_CLIENT_ID = '736147134702-glkb1pesv716j1utg4llg7c3rr7nnhli.apps.
  ↪googleusercontent.com'
6 OAUTH2_CLIENT_SECRET = 'secret'
```

```
1 {
2   "audio": {},
3   "auth_required": true,
```

(continues on next page)

(continued from previous page)

```
4   "client_id": "736147134702-glkb1pesv716j1utg4llg7c3rr7nnhli.apps.googleusercontent.
    ↪com",
5   "colors": {},
6   "customer_views": true,
7   "dates": {
8     "longDate": "EEEE, MMMM d, yyyy h:mm:ss.sss a (Z)",
9     "mediumDate": "medium",
10    "shortTime": "shortTime"
11  },
12  "endpoint": "https://alerta-api.herokuapp.com",
13  "github_url": null,
14  "gitlab_url": "https://gitlab.com",
15  "keycloak_realm": null,
16  "keycloak_url": null,
17  "pingfederate_url": null,
18  "provider": "google",
19  "refresh_interval": 5000,
20  "severity": {
21    "cleared": 5,
22    "critical": 1,
23    "debug": 7,
24    "indeterminate": 5,
25    "informational": 6,
26    "major": 2,
27    "minor": 3,
28    "normal": 5,
29    "ok": 5,
30    "security": 0,
31    "trace": 8,
32    "unknown": 9,
33    "warning": 4
34  },
35  "signup_enabled": true,
36  "tracking_id": "UA-44644195-5"
37 }
```

Note: For completeness, the `OAUTH2_CLIENT_ID` and `OAUTH2_CLIENT_SECRET` configuration settings are included in the example above however it should be noted that only the client id is sent to the client (line 4) as sending the client secret is not necessary and would compromise security.

Client Settings

Full list of API server settings that can be used to configure clients.

`AUTH_REQUIRED`

`CUSTOMER_VIEWS`

`AUTH_PROVIDER`

`SIGNUP_ENABLED`

`OAUTH2_CLIENT_ID`

`GITHUB_URL`

GITLAB_URL

KEYCLOAK_URL

KEYCLOAK_REALM

PINGFEDERATE_URL

COLOR_MAP

SEVERITY_MAP

GOOGLE_TRACKING_ID

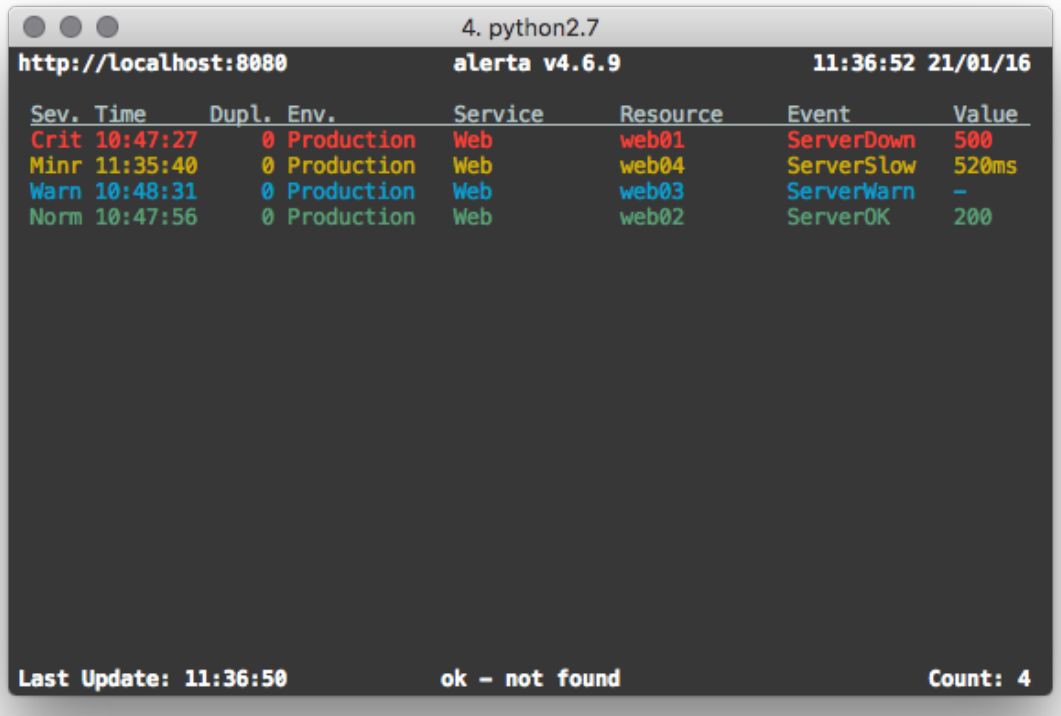
AUTO_REFRESH_INTERVAL

Note: It is not currently possible to configure dates or audio.

1.5 Alerta CLI

`alerta` is the unified command-line tool, terminal GUI and Python SDK for the alerta monitoring system.

It can be used to send and query alerts, tag alerts and change alert status, delete alerts, dump alert history or see the raw alert data. It can also be used to send heartbeats to the alerta server, and generate alerts based on missing or slow heartbeats.



1.5.1 Installation

The `alerta` client tool can be installed using pip:

```
$ pip install alerta
```

Or, by cloning the git repository:

```
$ git clone https://github.com/alerta/python-alerta-client.git
$ cd python-alerta-client
$ pip install .
```

1.5.2 Configuration

Options can be set in a configuration file, as environment variables or on the command line. Profiles can be used to easily switch between different configuration settings.

Option	Config File	Environment Variable	Optional Argument	Default
file	n/a	`ALERTA_CONF_FILE`	`--config-file FILE`	`~/.alerta.conf`
profile	profile	`ALERTA_DEFAULT_PROFILE`	`--profile PROFILE`	None
endpoint	endpoint	`ALERTA_ENDPOINT`	`--endpoint-url URL`	`http://localhost:8080`
key	key	`ALERTA_API_KEY`	n/a	None
provider	provider	`ALERTA_API_KEY`	n/a	basic
client id	client_id	n/a	n/a	None
GitHub URL	github_url	n/a	n/a	`https://github.com`
GitLab URL	gitlab_url	n/a	n/a	`https://gitlab.com`
timezone	timezone	n/a	n/a	Europe/London
timeout	timeout	n/a	n/a	5s TCP connection timeout
ssl verify	sslverify	`REQUESTS_CA_BUNDLE`	n/a	verify SSL certificates
output	output	n/a	`--output FORMAT`, `--json`	text
color	color	`CLICOLOR`	`--color`, `--no-color`	color on
debug	debug	`DEBUG`	`--debug`	no debug

Note: The `profile` option can only be set in the `[DEFAULT]` section.

Example

Configuration file `~/.alerta.conf`:

```
[DEFAULT]
timezone = Australia/Sydney
output = json
```

(continues on next page)

(continued from previous page)

```
[profile development]
endpoint = https://localhost:8443
key = demo-key
sslverify = off
timeout = 10.0
debug = yes
```

Set environment variables:

```
$ export ALERTA_CONF_FILE=~/.alerta.conf
$ export ALERTA_DEFAULT_PROFILE=production
```

Use production configuration settings by default:

```
$ alerta query
```

Switch to development configuration settings when required:

```
$ alerta --profile development query
```

1.5.3 Precedence

Command-line configuration options have precedence over environment variables, which have precedence over the configuration file. Within the configuration file, profile-specific sections have precedence over the `[DEFAULT]` section.

1.5.4 Authentication

If the Alerta API enforces authentication, then the `alerta` command-line tool can be configured to present an API key or Bearer token to the API when accessing secured endpoints.

API Keys

API keys can be generated in the web UI, or by an authenticated user using the `alerta` CLI, and should be added to the configuration file as the "key" setting as shown in the following example:

```
[profile production]
endpoint = https://api.alerta.io
key = LMvzLsfJyGpSuLmaB9kp-8gCl4I3YZkV4i7IGb6S
```

Bearer Tokens

Alternatively, a user can "login" to the API and retrieve a Bearer token if the Alerta API is configured to use either `basic`, `github`, `gitlab` or `google` as the authentication provider. An OAuth Client ID is required if not using `basic` and settings should be added to the configuration file as shown in the example below:

```
[profile cloud]
endpoint = https://alerta-api.herokuapp.com
provider = google
client_id = 736147134702-glkb1pesv716j1utg4llg7c3rr7nnhli.apps.googleusercontent.com
```

1.5.5 Commands

The `alerta` tool is invoked by specifying a command using the following format:

```
$ alerta [OPTIONS] COMMAND [ARGS]...
```

The following group of commands are related to creating, querying and managing alerts.

send

Send an alert.

```
$ alerta send [OPTIONS]

Options:
-r, --resource RESOURCE        Resource under alarm
-e, --event EVENT              Event name
-E, --environment ENVIRONMENT  Environment eg. Production, Development
-s, --severity SEVERITY        Severity eg. critical, major, minor, warning
-C, --correlate EVENT          List of related events eg. node_up, node_down
-S, --service SERVICE          List of affected services eg. app name, Web,
                                Network, Storage, Database, Security
-g, --group GROUP              Group event by type eg. OS, Performance
-v, --value VALUE              Event value
-t, --text DESCRIPTION         Description of alert
-T, --tag TAG                  List of tags eg. London, os:linux, AWS/EC2
-A, --attributes KEY=VALUE     List of attributes eg. priority=high
-O, --origin ORIGIN            Origin of alert in form app/host
--type EVENT_TYPE              Event type eg. exceptionAlert,
                                performanceAlert, nagiosAlert
--timeout SECONDS              Seconds before an open alert will be expired
--raw-data STRING              Raw data of orignal alert eg. SNMP trap PDU.
                                '@' to read from file, '-' to read from stdin
--customer STRING              Customer
-h, --help                     Show this message and exit.
```

The only mandatory options are `resource` and `event`. All the others will be set to sensible defaults.

Attention: If the `reject` plugin is enabled (which it is by default) then alerts must have an `environment` attribute that is one of either `Production` or `Development` and it must define a `service` attribute. For more information on configuring or disabling this plugin see *Plugin Settings*.

Attribute	Default
environment	empty string
severity	`normal`
correlate	empty list
status	`unknown`
service	empty list
group	`Misc`
value	`n/a`
text	empty string
tags	empty list
attributes	empty dictionary
origin	program/host
type	`exceptionAlert`
timeout	86400 (1 day)
raw data	empty string

Examples

To send a `minor` alert followed by a `normal` alert that correlates:

```
$ alerta send --resource web01 --event HttpError --correlate HttpOK --group Web --
↪severity minor
$ alerta send --resource web01 --event HttpOK --correlate HttpError --group Web --
↪severity normal
```

To send an alert with custom attribute called `customer`:

```
$ alerta send -r web01 -e HttpError -g Web -s major --attributes customer="Tyrell Corp
↪"
```

To query for major and minor open alerts for the Production environment of the Mobile API service:

```
$ alerta query --filters severity=major severity=minor status=open␣
↪environment=Production service="Mobile API"
```

To query for all alerts with "disk" in the alert text:

```
$ alerta query --filters text=~disk
```

query

Query for alerts based on search filter criteria.

```
$ alerta query [OPTIONS]

Options:
-i, --ids UUID       List of alert IDs (can use short 8-char id)
-f, --filter FILTER  KEY=VALUE eg. serverity=warning resource=web
--tabular            Tabular output
--compact            Compact output
--details            Compact output with details
-h, --help           Show this message and exit.
```

watch

Watch for new alerts.

```
$ alerta watch [OPTIONS]

Options:
-i, --ids UUID          List of alert IDs (can use short 8-char id)
-f, --filter FILTER     KEY=VALUE eg. serverity=warning resource=web
--details               Compact output with details
-n, --interval SECONDS  Refresh interval
-h, --help              Show this message and exit.
```

top

Display alerts like unix "top" command.

```
$ alerta top [OPTIONS]

Options:
-h, --help  Show this message and exit.
```

raw

Show raw data for alerts.

```
$ alerta raw [OPTIONS]

Options:
-i, --ids UUID       List of alert IDs (can use short 8-char id)
-f, --filter FILTER  KEY=VALUE eg. serverity=warning resource=web
-h, --help           Show this message and exit.
```

history

Show status and severity changes for alerts.

```
$ alerta history [OPTIONS]

Options:
-i, --ids UUID       List of alert IDs (can use short 8-char id)
-f, --filter FILTER  KEY=VALUE eg. serverity=warning resource=web
-h, --help           Show this message and exit.
```

tag

Add tags to alerts.

```
$ alerta tag [OPTIONS]

Options:
-i, --ids UUID       List of alert IDs (can use short 8-char id)
```

(continues on next page)

(continued from previous page)

```
-f, --filter FILTER  KEY=VALUE eg. serverity=warning resource=web
-T, --tag TEXT       List of tags  [required]
-h, --help           Show this message and exit.
```

untag

Remove tags from alerts.

```
$ alerta untag [OPTIONS]

Options:
-i, --ids UUID       List of alert IDs (can use short 8-char id)
-f, --filter FILTER  KEY=VALUE eg. serverity=warning resource=web
-T, --tag TEXT       List of tags  [required]
-h, --help           Show this message and exit.ntag alerts ie. remove an assigned␣
↪tag from alert tag list::
```

update

Update alert attributes.

```
$ alerta update [OPTIONS]

Options:
-i, --ids UUID               List of alert IDs (can use short 8-char id)
-f, --filter FILTER          KEY=VALUE eg. serverity=warning resource=web
-A, --attributes KEY=VALUE   List of attributes eg. priority=high  [required]
-h, --help                   Show this message and exit.
```

ack

Acknowlege alerts ie. change alert `status` to `ack`:

```
:command:`unack`
```

Unacknowledge alerts ie. change alert `status` to `open`:

```
:command:`close`
```

Close alerts ie. change alert `status` to `closed`:

```
:command:`delete`
```

Delete alerts from server:

```
:command:`blackout`
```

Blackout alerts based on attributes:

```
:command:`blackouts`
```

List all blackout periods:

```
:command:`heartbeat`
```

Send a heartbeat to the server:

```
:command:`heartbeats`
```

List all heartbeats:

```
:command:`user`
```

Manage user details (Basic Auth only):

```
:command:`users`
```

List all users:

```
:command:`key`
```

Create API key:

```
:command:`keys`
```

List all API keys:

```
:command:`revoke`
```

Revoke API key:

```
.. _cli_status:
```

status

Show status and metrics:

```
:command:`uptime`
```

Show server uptime:

```
:command:`version`
```

Show version information for `alerta` and dependencies.

help

Show all `OPTIONS`, `COMMANDS` and some example `FILTERS`.

1.5.6 Bugs

Log any issues on GitHub or submit a pull request.

1.6 Integrations & Plugins

There are several different ways to integrate other alert sources into Alerta.

Firstly, *integrations* with well known monitoring tools like Nagios, Zabbix and Sensu make use of the Alerta API and demonstrate how to build integrations with other monitoring tools.

Secondly, there are built-in *webhooks* for AWS Cloudwatch, Pingdom, PagerDuty, Google Stackdriver, Prometheus Alertmanager and more which provide 'out-of-the-box' integrations for some of the most popular monitoring systems available.

Thirdly, *alert severity indicators* or widgets can be placed on any web page using oEmbed for easy integration with existing dashboards.

Lastly, *plugins* can be used to quickly and easily forward alerts to or notify other systems like Slack or Hipchat.

1.6.1 Integrations

Core

There are a few core integrations which have been developed to showcase how easy it is to get alerts or events from other tools into Alerta. They are:

- Nagios Event Broker - forward host/service check results with suppression during downtime
- InfluxData Kapacitor - forward alerts for metric anomalies and dynamic thresholds
- Zabbix Alert Script - forward problems, acknowledged and OK events
- Sensu Plugin - forward sensu events
- Riemann Plugin - generate alerts from thresholds defined against metric streams
- Kibana Logging - log alerts to Elasticsearch for historical visualisation of alert trends

Contrib

There are several more integrations available in the contrib repo which may be useful. They are:

- Amazon SQS - receive alerts from SQS that were sent using the SNS core plugin
- E-mail - send emails after a hold-time has expired (requires the AMQP message queue core plugin)
- Opsweekly - query Alerta to generate Opsweekly reports
- Pinger - generate ping alerts from list of network resources being pinged
- SNMP Trap - generate alerts from SNMPv1 and SNMPv2 sources
- Supervisor - trigger alerts and heartbeats based on process deamon events
- Syslog Forwarder - receive **RFC 5424**, **RFC 3164** syslog and Cisco syslog messages
- URL monitor - trigger alerts from web service query responses

1.6.2 Webhooks

Webhooks are a way of integrating with other systems by triggering HTTP callbacks to the Alerta server API when an event occurs.

AWS CloudWatch

Alerta can be configured to receive AWS CloudWatch alarms by subscribing the Alerta API endpoint to an SNS topic.

For details on how to set this up see the Sending Amazon SNS Messages to HTTP/HTTPS Endpoints page and in the *Endpoint* input box append `/webhooks/cloudwatch` to the Alerta API URL.

Example AWS CloudWatch Webhook URL

```
https://alerta.example.com/api/webhooks/cloudwatch
```

Pingdom

Alerta can be configured to receive Pingdom URL check alerts by adding a webhook alerting endpoint that calls the Alerta API.

For details on how to set this up see the Pingdom webhook page and in the *webhook URL* input box append `/webhooks/pingdom` to the Alerta API URL.

Example Pingdom Webhook URL

```
https://alerta.example.com/api/webhooks/pingdom
```

PagerDuty

Alerta can be configured to receive PagerDuty incident-based webhooks – any change to the `status` or `assigned_to_user` of an incident will cause an outgoing message to be sent.

For details on how to set this up see the PagerDuty webhook page and where it requires the webhook URL append `/webhooks/pagerduty` to the Alerta API URL.

Example PagerDuty Webhook URL

```
https://alerta.example.com/api/webhooks/pagerduty
```

Prometheus Alertmanager

Alerta can be configured as a webhook receiver in Alertmanager.

For details on how to set this up see the Prometheus Config GitHub Repo

Google Stackdriver

Alerta can be configured to receive Google Stackdriver incidents by adding a webhook endpoint to the notifications configuration.

For details on how to set this up see Stackdriver webhook page and in the *ENDPOINT URL* input box append `/webhooks/stackdriver` to the Alerta API URL.

Example Stackdriver Webhook URL

```
https://alerta.example.com/api/webhooks/stackdriver
```

SeverDensity

Alerta can be configured to receive SeverDensity alerts by adding a webhook endpoint to the Notification Preferences.

For details on how to set this up see SeverDensity webhook page and in the *Endpoint URL* input box append `/webhooks/serverdensity` to the Alerta API URL.

Example SeverDensity Webhook URL

```
https://alerta.example.com/api/webhooks/serverdensity
```

netdata

https://github.com/firehol/netdata/wiki/Alerta-monitoring-system

New Relic

Alerta can be configured to receive New Relic incidents by adding a webhook endpoint to the Notification Channels.

For details on how to set this up see New Relic webhook page and in the *Endpoint URL* input box append `/webhooks/newrelic` to the Alerta API URL.

Example New Relic Webhook URL

```
https://alerta.example.com/api/webhooks/newrelic
```

Grafana

Alerta can be configured to receive Grafana alerts by adding a webhook endpoint to the Notification Channels.

For details on how to set this up see Grafana webhook page and in the *Endpoint URL* input box append `/webhooks/grafana` to the Alerta API URL.

Example Grafana Webhook URL

```
https://alerta.example.com/api/webhooks/grafana
```

The following parameters can be set in the url environment, event_type, group, origin, service, severity, timeout

:file:‘https://alerta.example.com/api/webhooks/grafana?api-key=xxx &environment=Production &event_type=performanceAlert &group=Performance &origin=Grafana &service=Grafana &severity=major &timeout=86400‘

Telegram

Alerta can be configured to receive Telegram callback queries from the inline buttons in the Telegram Bot plugin.

For details on how to set this up see Telegram Bot page and for the `TELEGRAM_WEBHOOK_URL` setting append `/webhooks/telegram` to the Alerta API URL.

Example Telegram Webhook URL

```
https://alerta.example.com/api/webhooks/telegram
```

Tick Stack

https://docs.influxdata.com/kapacitor/v1.5/event_handlers/alerta/

Riemann

Alerta can be configured to receive Riemann events. The integration makes no assumptions about the format of the Riemann events and consumes standard events. If events are decorated with additional metadata (eg. tags, environment, group, etc) then these will be used.

Example Riemann Webhook URL

```
https://alerta.example.com/api/webhooks/riemann
```

1.6.3 Widgets

Add an alert severity indicator (aka. widget) to any dashboard using the Oembed API endpoint. The severity indicator is coloured with the maximum severity for that alert query filter and has a count for the total number of matching alerts for each severity.

Multiple severity indicators can be placed on the same page each for a different environment, service or group. See the example oembed web page.

1.6.4 Plugins

Plugin extensions are an easy way of adding new features to Alerta that meet a specific end-user requirement.

Core

Core plugins have been developed as examples of common use-cases.

- Reject - reject alerts before processing. used to enforce custom alert format policies

Contrib

Contributed plugins are made available for popular tools but implementation-specific requirements.

- AMQP - publish alerts to an AMQP fanout topic after processing
- Cachet - create incidents for display on Cachet status page
- Enhance - add new information to an alert based on existing information
- GeoIP Location - use remote IP address to submitted alert to add location data
- HipChat - send alerts to HipChat room
- InfluxDB - send alerts to InfluxDB for graphing with Grafana
- Logstash/Kibana - send alerts to logstash agent after processing
- Normalise - ensure alerts a formatted in a consistent manner
- PagerDuty Plugin - send alerts to PagerDuty (webhooks used to receive callbacks)
- Prometheus Silencer - silence alerts in Prometheus Alertmanager if ack'ed in Alerta
- Pushover.net - send alerts to Pushover.net
- Slack - send alerts to Slack room
- AWS SNS - publish alerts to SNS topic after processing
- Syslog Logger - send alerts via syslog
- Telegram Bot - send alerts to Telegram channel
- Twilio SMS - send alerts via SMS using Twilio

1.7 Authentication

By default, authentication is not enabled, however there are some features that are not available unless users login such as watching alerts.

Alerta supports three authentication mechanisms for the web UI and `alerta` command-line tool.

- *Basic Auth*
- *Google OAuth2*
- *GitHub OAuth2*
- *GitLab OAuth2*
- *Keycloak OAuth2*
- *SAML 2.0*
- *API Keys*

To enforce authentication set `AUTH_REQUIRED` to `True` and set the `SECRET_KEY` to some random string in the `alertad.conf` server configuration settings file:

```
AUTH_REQUIRED = True
SECRET_KEY = 'UszE5hI_hx5pXKcsCP_2&1DIs&9_Ve*k'
```

Note: Ensure that the `SECRET_KEY` that is used to encode tokens and API keys is a unique, randomly generated sequence of ASCII characters. The following command generates a suitable 32-character random string on Mac or Linux:

```
$ LC_CTYPE=C tr -dc A-Za-z0-9_\!\@\#\$\%\^\&\*\(\)-+= < /dev/urandom | head -c 32 &&␣
↪echo
```

1.7.1 Basic Auth

The most straight-forward authentication strategy to implement of the three is HTTP Basic Authentication because there is no additional configuration required of the Alerta server to use it other than setting `AUTH_REQUIRED` to `True`.

Note: HTTP Basic Auth does not provide any encryption of the username or password so it is strongly advised to only use Basic Auth over HTTPS.

1.7.2 OAuth2 Authentication

OAuth authentication is provided by Google OpenID Connect, GitHub, GitLab OAuth 2.0 or Keycloak OAuth 2.0 and configuration is more involved than the Basic Auth setup.

Note: If Alerta is deployed to a publicly accessible web server it is important to configure the OAuth2 settings correctly to ensure that only authorised users can access and modify your alerts.

Ensure `AUTH_REQUIRED` and `SECRET_KEY` are set and that the `AUTH_PROVIDER` setting is

Then follow the steps below for the chosen OAuth provider to create an OAuth client ID and client secret. The client ID and client secret will need to be added to the `alertad.conf` file for the Alerta server.

Google OAuth2

To use Google as the OAuth2 provider for Alerta, login to Google Developer Console and create a new project for alerta.

- Project Name: alerta
- Project ID: (automatically assigned)

Go to *APIs and auth -> APIs* and set *Google+ API* to **ON**. Next go to *APIs and auth -> Credentials* and click **Create New Client ID** and choose **Web Application**.

- Authorized Javscript Origins: http://alerta.example.com
- Authorized Redirect URIs: http://alerta.example.com

Click **Create Client ID** and take note of the Client ID and Client Secret. The configuration settins for `alerta` server are as follows:

```
AUTH_PROVIDER = 'google'
OAUTH2_CLIENT_ID = '379647311730-sj130ru952o3o7ig8u0ts8np2ojivr8d.apps.
↪googleusercontent.com'
OAUTH2_CLIENT_SECRET = '8HrqJhbrYn9oDtaJqExample'
```

To restrict access to users with particular Google apps domains use:

```
ALLOWED_EMAIL_DOMAINS = ['example.org', 'mycompany.com']
```

Note: `ALLOWED_EMAIL_DOMAINS` can be an asterisk (`*`) to force login but *not* restrict who can login.

GitHub OAuth2

To use GitHub as the OAuth2 provider for Alerta, login to GitHub and go to *Settings* -> *Applications* -> *Register New Application*.

- Application Name: Alerta
- Homepage URL: http://alerta.io
- Application description (optional): Guardian Alerta monitoring system
- Authorization callback URL: http://alerta.example.com

Note: The *Authorization callback URL* is the most important setting and it is nothing more than the URL domain (ie. without any path) where the alerta Web UI is being hosted.

Click Register Application and take note of the Client ID and Client Secret. Then configuration settings for `alerta` server are as follows:

```
AUTH_PROVIDER = 'github'
OAUTH2_CLIENT_ID = 'f7b0c15e2b722e0e38f4'
OAUTH2_CLIENT_SECRET = '7aa9094369b72937910badab0424dc7393x8mpl3'
```

To restrict access to users who are members of particular GitHub organisations use:

```
ALLOWED_GITHUB_ORGS = ['example', 'mycompany']
```

Note: `ALLOWED_GITHUB_ORGS` can be an asterisk (`*`) to force login but *not* restrict who can login.

Important: To revoke access of your instance of alerta to your GitHub user info at any time go to *Settings* -> *Applications* -> *Authorized* applications, find alerta in the list of applications and click the **Revoke** button.

GitLab OAuth2

To use GitLab as the OAuth2 provider for Alerta, login to GitLab and go to *Profile Settings* -> *Applications* -> *New Application*.

- Name: Alerta

- Redirect URL: http://alerta.example.com

Note: The *Redirect URL* is the most important setting and it is nothing more than the URL domain (ie. without any path) where the alerta Web UI is being hosted.

Click *Submit* and take note of the Application ID and Secret. Then configuration settings for `alerta` server are as follows (replacing the values shown below with the values generated by GitLab):

```
AUTH_PROVIDER = 'gitlab'
GITLAB_URL = 'https://gitlab.com'  # or your own GitLab server
OAUTH2_CLIENT_ID = 'd31e9caa131f72901b16d22289c824f423bd5cbf187a11245f402e8b2707d591'
OAUTH2_CLIENT_SECRET =
↪'42f1de369ec706996cadda234986779eeb65c0201a6f286b9751b1f845d62c8a'
```

To restrict access to users who are members of particular GitLab groups use:

```
ALLOWED_GITLAB_GROUPS = ['group1', 'group2']
```

Note: `ALLOWED_GITLAB_GROUPS` can be an asterisk (`*`) to force login but *not* restrict who can login.

Important: To revoke access of your instance of alerta to your GitLab user info at any time go to *Profile Settings* -> *Applications* -> *Authorized appliations*, find alerta in the list of applications and click the **Revoke** button.

Keycloak OAuth2

To use Keycloak as the OAuth2 provider for Alerta, login to Keycloak admin interface, select the realm and go to *Clients* -> *Create*.

- Client ID: alerta-ui
- Client protocol: openid-connect
- Root URL: http://alerta.example.org

After the client is created, edit it and change the following properties:

- Access Type: confindential

Add the following mapper under the *Mappers* tab:

```
Name: role memberships
Mapper type: User Realm Role
Token Claim Name: roles
Claim JSON type: String
Add to userinfo: ON
```

Now go to *Installation* and generate it by selecting 'Keycloak OIDC JSON'. You should get something like this:

```
{
  "realm": "master",
  "auth-server-url": "https://keycloak.example.org/auth",
  "ssl-required": "external",
  "resource": "alerta-ui",
```

(continues on next page)

(continued from previous page)

```
  "credentials": {
    "secret": "418bbf31-aef-33d1-a471-322a60276879"
  },
  "use-resource-role-mappings": true
}
```

Take note of the realm, resource and secret. Then configuration settings for `alerta` server are as follows (replacing the values shown below with the values generated by Keycloak):

```
AUTH_PROVIDER = 'keycloak'
KEYCLOAK_URL = 'https://keycloak.example.org'
KEYCLOAK_REALM = 'master'
OAUTH2_CLIENT_ID = 'alerta-ui'
OAUTH2_CLIENT_SECRET = '418bbf31-aef-33d1-a471-322a60276879'
```

To restrict access to users who are associated with a particular Keycloak role use:

```
ALLOWED_KEYCLOAK_ROLES = ['role1', 'role2']
```

Note: `ALLOWED_KEYCLOAK_ROLES` can be an asterisk (`*`) to force login but *not* restrict who can login.

Cross-Origin

If the Alerta API is not being served from the same domain as the Alerta Web UI then the `CORS_ORIGINS` setting needs to be updated to prevent modern browsers from blocking the cross-origin requests.

```
CORS_ORIGINS = [
    'http://try.alerta.io',
    'http://explorer.alerta.io',
    'chrome-extension://jplkjnjaegjgacpfafdopnpnhmobhlaf',
    'http://localhost'
]
```

1.7.3 SAML 2.0 Authentication

OAuth authentication is provided by Google OpenID Connect, GitHub, GitLab OAuth 2.0 or Keycloak OAuth 2.0 and configuration is more involved than the Basic Auth setup.

1.7.4 SAML 2.0

Generate private/public key pair

```
openssl req -utf8 -new -x509 -days 3652 -nodes -out "alerta.cert" -keyout "alerta.key"
```

Note: This key pair is not related to HTTPS.

Configure pysaml2

Bare-minimum config example:

```
AUTH_PROVIDER = 'saml2'
SAML2_CONFIG = {
    'metadata': {
        'local': ['/path/to/federationmetadata.xml']
    },
    'key_file': '/path/to/alerta.key',
    'cert_file': '/path/to/alerta.cert'
}
```

metadata IdP metadata (refer to saml2 documentation for possible ways of specifying it)

key_file, cert_file path to aforementioned keys

Refer to pysaml2 documentation and source code if you need additional options:

- https://pysaml2.readthedocs.io/en/latest/howto/config.html
- https://github.com/rohe/pysaml2/blob/master/src/saml2/config.py

Note: entityid and service provider endpoints are configured by default based on your BASE_URL value which is mandatory if you use SAML (see *General Settings*)

ALLOWED_SAML2_GROUPS

To restrict access to users who are members of particular group use:

```
ALLOWED_SAML2_GROUPS = ['alerta_ro', 'alerta_rw']
```

Note: Ensure that pysaml2 authn response identity object contains `groups` attribute. You can do this by writing proper attribute map which will convert your IdP-specific attribute name to `groups`.

Example:

```
MAP = {
    ...
    'fro': {
        ...
        'http://schemas.xmlsoap.org/claims/group': 'groups',
        ...
    },
    'to': {
        ...
        'groups': 'http://schemas.xmlsoap.org/claims/group',
        ...
    }
}
```

See pysaml2 attribute-map-dir documentation. `attribute-map-dir` can be specified in the `SAML2_CONFIG`, see *Configure pysaml2*

SAML2_USER_NAME_FORMAT

This is a python string template which is used to generate user's name based on attributes (make sure that attribute-map-dir is properly configured in case default does not fit). Default is `'{givenName} {surname}'`.

Cross-Origin

You also need to add your IdP origin to CORS headers:

```
CORS_ORIGINS = [
    ...
    'https://sso.example.com',
    ...
]
```

Add trusted Service Provider to your Identity Provider

Your metadata url is: `{BASE_URL}/auth/saml/metadata.xml`, pass it to your IdP administrator.

1.7.5 API Keys

If authentication is enforced, then an API key is needed to access the alerta API programatically or to use the *alerta CLI*. Keys can be easily generated from the Alerta web UI and can be *read-write* or *read-only*. They are valid for 1 year but this period is configurable using `API_KEY_EXPIRE_DAYS` in the *server configuration*.

See the *example CLI config* for how to set the API key for the command-line tool.

To use an API key in an API query you must set the correct HTTP `Authorization` header:

```
curl 'http://api.alerta.io/alerts' -H 'Authorization: Key demo-key' -H 'Accept:␣
↪application/json'
```

or use the `api-key` GET parameter:

```
curl 'http://api.alerta.io/alerts?api-key=demo-key' -H 'Accept: application/json'
```

Note: Using the HTTP `Authorization` header is preferred so that API keys are not inadvertently captured in log files and accidentally exposed.

1.7.6 User Authorisation

Google, GitHub, GitLab OAuth, Keycloak OAuth are used for user authentication, not user authorisation. Authentication proves that you are who you say you are. Authorization says that you are allowed to access what you have requested.

To control who has access to Alerta you can restrict access to users with a *certain email domain name* by setting `ALLOWED_EMAIL_DOMAINS` when using Google OAuth2, or who belong to a *particular GitHub organisation* by setting `ALLOWED_GITHUB_ORGS` when using GitHub OAuth, or who belong to a *particular GitLab group* by setting `ALLOWED_GITLAB_GROUPS` when using GitLab OAuth2. belong to a *particular Keycloak role* by setting `ALLOWED_KEYCLOAK_ROLES` when using Keycloak OAuth2

For those situations where it is not possible to group users in this way it is possible to selectively allow access on a per-user basis. How this is done depends on whether you are using Google, GitHub, GitLab or Keycloak as OAuth2 provider for user login.

1.7.7 User Roles

TBC

1.8 Authorization

Authorization is used to limit access to Alerta API resources. The authorization model is based on Role Based Access Control (RBAC) which assigns permissions to functional roles and then users are assigned to one or more of those roles.

This "role-based access" allows for fine-grained control over exactly what resources are accessible to which users and exactly what type of access is allowed – in a way that is scalable.

For example, to create a new alert the sender will need to be assigned to a role with `write:alerts` permissions. If the sender is not a member of a role with those permisssions then the request will be rejected with a `403 Forbidden` response code.

Note: All access is through roles. Permissions can not be assigned directly to users. The only exception to this is the `ADMIN_USERS` setting which overrides all other roles a user might belong to.

1.8.1 Configuration

There are two ways to configure role-based access; default and custom configuration.

Default Authorization

If *authentication* is enabled then the default authorization is used which defines two roles:

- `user` role - everyone is a "user" unless listed in the `ADMIN_USERS` setting
- `admin` role - only admins can delete alerts and heartbeats, create users etc.

Custom Authorization

To use custom authorization simply define one or more permission scope lookups.

As an "admin" user go to *Configuration -> Permissions* and add a new role with the required scopes. See below for list of valid scopes.

1.8.2 Scopes and Permissions

Use these scopes to request access to API resources.

Scope	Permissions
`read`	Grants read-only access to all scopes.
`write`	Grants read/write access to all scopes.
`admin`	Grants admin, read, write and delete access to all scopes.
`read:alerts`	Read-only access to alerts.
`write:alerts`	Grants read/write access to alerts.
`admin:alerts`	Grants read, write and delete access to alerts.
`read:blackouts`	Grants read-only access to blackouts.
`write:blackouts`	Grants read/write access to blackouts.
`read:heartbeats`	Read-only access to heartbeats.
`write:heartbeats`	Grants read/write access to heartbeats.
`admin:heartbeats`	Grants read, write and delete access to heartbeats.
`admin:users`	Fully manage users.
`admin:customers`	Fully manage customers.
`read:keys`	List and view API keys.
`write:keys`	Create, list and view API keys.
`admin:keys`	Fully manage API keys.
`write:webhooks`	Grants write access to webhooks.
`read:oembed`	Grants read-only to oembed endpoints.
`read:management`	Grants read-only access to management endpoints.
`admin:management`	Fully manage management endpoints.
`read:userinfo`	Grants read-only access to userinfo.

Note: `write` implicitly includes `read`, and `admin` implicitly includes `read` and `write`.

1.9 Configuration

The following settings **only** apply to the Alerta server. For `alerta` CLI configuration options see *command-line reference* and for Web UI configuration options see *web UI reference*.

The configuration file uses standard python syntax for setting variables. The default settings (defined in *settings.py*) **should not** be modified directly. To change any of these settings create a configuration file that overrides these default settings. The default location for the server configuration file is `/etc/alertad.conf` however the location itself can be overridden by using a environment variable `ALERTA_SVR_CONF_FILE`.

For example, to set the blackout period default duration to 1 day (ie. 86400 seconds):

```
$ export ALERTA_SVR_CONF_FILE=~/.alertad.conf
$ echo "BLACKOUT_DURATION = 86400" >$ALERTA_SVR_CONF_FILE
```

1.9.1 Config File Settings

General Settings

```
DEBUG = False
BASE_URL = ''
LOGGER_NAME = 'alerta'
LOG_FILE = None
```

DEBUG debug mode. Set to `True` for increased logging.

BASE_URL if API served behind a proxy use `BASE_URL` to fix relative links

LOGGER_NAME name of logger used by python `logging` module

LOG_FILE full path to write rotating server log file

API Settings

```
DEFAULT_PAGE_SIZE = 1000
HISTORY_LIMIT = 100
API_KEY_EXPIRE_DAYS = 365
```

DEFAULT_PAGE_SIZE maximum number of alerts returned in a single query.

HISTORY_LIMIT number of history entries returned in alert details.

API_KEY_EXPIRE_DAYS number of days an API key is valid for.

Database Settings

There is a choice of either Postgres or MongoDB as the backend database.

The database is defined using the standard database connection URL formats. Many database configuration options are supported as connection URL parameters.

Postgres Example

```
DATABASE_URL = 'postgresql://other@localhost/otherdb?connect_timeout=10&application_
↪name=myapp'
DATABASE_NAME = 'monitoring'
```

See Postgres connection strings for more information.

MongoDB Example

```
DATABASE_URL = 'mongodb://db1.example.net,db2.example.net:2500/?replicaSet=test&
↪connectTimeoutMS=300000'
DATABASE_NAME = 'monitoring'
```

See MongoDB connection strings for more information.

DATABASE_URL database connection URI string.

DATABASE_NAME database name can be used to override default database defined in `DATABASE_URL`.

If the document-oriented datastore MongoDB is used for persistent data, then it can be set-up as a stand-alone server or in a replica set for high availability.

Authentication Settings

If enabled, authentication provides additional benefits beyond just security, such as auditing, and features like the ability to assign and watch alerts.

```
SECRET_KEY = 'changeme'
AUTH_REQUIRED = False

ADMIN_USERS = []
CUSTOMER_VIEWS = False

OAUTH2_CLIENT_ID = None  # Google or GitHub OAuth2 client ID and secret
OAUTH2_CLIENT_SECRET = None
ALLOWED_EMAIL_DOMAINS = ['*']

GITHUB_URL = None
ALLOWED_GITHUB_ORGS = ['*']

GITLAB_URL = None
ALLOWED_GITLAB_GROUPS = ['*']

KEYCLOAK_URL = None
KEYCLOAK_REALM = None
ALLOWED_KEYCLOAK_ROLES = ['*']

SAML2_CONFIG = None
ALLOWED_SAML2_GROUPS = ['*']
SAML2_USER_NAME_FORMAT = '{givenName} {surname}'

TOKEN_EXPIRE_DAYS = 14
```

SECRET_KEY a unique, randomly generated sequence of ASCII characters.

AUTH_REQUIRED set to `True` to force users to authenticate when using web UI or command-line tool

ADMIN_USERS list of user email addresses or accounts that should be given admin rights.

CUSTOMER_VIEWS enable alert views partitioned by customer

OAUTH2_CLIENT_ID client ID required by OAuth2 provider for Google, Github, GitLab or Keycloak.

OAUTH2_CLIENT_SECRET client secret required by OAuth2 provider for Google, Github, GitLab or Keycloak.

ALLOWED_EMAIL_DOMAINS list of authorised email domains when using Google as OAuth2 provider.

GITHUB_URL GitHub Enteprise URL for privately run GitHub server when using GitHub as OAuth2 provider.

ALLOWED_GITHUB_ORGS list of authorised GitHub organisations a user must belong to when using Github as OAuth2 provider.

GITLAB_URL GitLab website URL for public or privately run GitLab server when using GitLab as OAuth2 provider.

ALLOWED_GITLAB_GROUPS list of authorised GitLab groups a user must belong to when using GitLab as OAuth2 provider.

KEYCLOAK_URL Keycloak website URL when using Keycloak as OAuth2 provider.

KEYCLOAK_REALM Keycloak realm when using Keycloak as OAuth2 provider.

ALLOWED_KEYCLOAK_ROLES list of authorised Keycloak roles a user must belong to when using Keycloak as OAuth2 provider.

SAML2_CONFIG `pysaml2` configuration `dict`. See *SAML 2.0 Authentication*.

ALLOWED_SAML2_GROUPS list of authorised groups a user must belong to. See *SAML 2.0 Authentication* for details.

SAML2_USER_NAME_FORMAT Python format string which will be rendered to user's name using SAML attributes. See *SAML 2.0 Authentication*.

Switch Settings

Server-side switches used to control and limit access to the API by clients for reasons related to security, performance or availability.

```
AUTO_REFRESH_ALLOW = 'ON'
SENDER_API_ALLOW = 'ON'
```

AUTO_REFRESH_ALLOW set to 'OFF' to reduce load on API server by forcing clients to manually refresh

SENDER_API_ALLOW set to 'OFF' to block clients from sending new alerts to API server

CORS Settings

```
CORS_ORIGINS = [
    'http://try.alerta.io',
    'http://explorer.alerta.io',
    'http://localhost'
]
```

CORS_ORIGINS list of URL origins that can access the API

Severity Settings

The severities and their order are customisable to fit with the environment in which Alerta is deployed.

```
SEVERITY_MAP = {
    'security': 0,
    'critical': 1,
    'major': 2,
    'minor': 3,
    'warning': 4,
    'indeterminate': 5,
    'cleared': 5,
    'normal': 5,
    'ok': 5,
    'informational': 6,
    'debug': 7,
    'trace': 8,
    'unknown': 9
}
DEFAULT_SEVERITY = 'indeterminate'
```

SEVERITY_MAP severity names and levels are fully customisable.

DEFAULT_SEVERITY the previous severity assigned to new alerts.

Blackout Periods Settings

Alerts can be suppressed based on alert attributes for arbitrary durations known as "blackout periods".

```
BLACKOUT_DURATION = 3600
```

BLACKOUT_DURATION default period for an alert blackout

Email Settings

If email verification is enabled then emails are sent to users when they sign up via BasicAuth. They must click on the provided link to verify their email address before they can login.

```
EMAIL_VERIFICATION = False
SMTP_HOST = 'smtp.gmail.com'
SMTP_PORT = 587
MAIL_FROM = 'your@gmail.com'
SMTP_PASSWORD = ''
```

EMAIL_VERIFICATION set to `True` to enable email verification of new users.

SMTP_HOST SMTP host of mail server.

SMTP_PORT SMTP port of mail server.

MAIL_FROM valid email address from which verification emails are sent.

SMTP_PASSWORD password for `MAIL_FROM` email account, Gmail uses application-specific passwords

Plugin Settings

Plugins are used to extend the behaviour of the Alerta server without having to modify the core application. The only plugin that is installed and enabled by default is the `reject` plugin. Other plugins are available in the contrib repo.

```
# Plugins
PLUGINS = ['reject']

ORIGIN_BLACKLIST = ['foo/bar$', '.*/qux']  # reject all foo alerts from bar, and
↪everything from qux
ALLOWED_ENVIRONMENTS = ['Production', 'Development']  # reject alerts without allowed
↪environments
```

PLUGINS list of enabled plugins

ORIGIN_BLACKLIST `reject` plugin list of alert origins blacklisted from submitting alerts. useful for rouge alert sources.

ALLOWED_ENVIRONMENTS `reject` plugin list of allowed environments. useful for enforcing discrete set of environments.

Note: To completely disable the `reject` plugin simply remove it from the list of enabled plugins in the `PLUGINS` configuration setting to override the default.

1.9.2 Environment Variables

Some configuration settings are special because they can be overridden by environment variables. This is to make deployment to different platforms and managed environments such as Heroku, Kubernetes and AWS easier, or to make use of managed Postgres or MongoDB services.

Note: Environment variables are read after configuration files so they will always override any other setting.

General Settings

DEBUG see above

BASE_URL see above

SECRET_KEY see above

AUTH_REQUIRED see above

ADMIN_USERS see above

CUSTOMER_VIEWS see above

OAUTH2_CLIENT_ID see above

OAUTH2_CLIENT_SECRET see above

ALLOWED_EMAIL_DOMAINS see above

GITHUB_URL see above

ALLOWED_GITHUB_ORGS see above

GITLAB_URL see above

ALLOWED_GITLAB_GROUPS see above

CORS_ORIGINS see above

MAIL_FROM see above

SMTP_PASSWORD see above

PLUGINS see above

Database Settings

DATABASE_URL used by both Postgres and MongoDB for database connection strings

DATABASE_NAME database name can be used to override default database defined in `DATABASE_URL`

MongoDB Settings

Deprecated since version 5.0: Use `DATABASE_URL` and `DATABASE_NAME` instead.

MONGO_URI used to override `MONGO_URI` config variable using the standard connection string format

MONGODB_URI alternative name for `MONGO_URI` environment variable which is used by some managed services

MONGOHQ_URL automatically set when using Heroku MongoHQ managed service

MONGOLAB_URI automatically set when using Heroku MongoLab managed service

MONGO_PORT automatically set when deploying Alerta to a Docker linked mongo container

1.9.3 Dynamic Settings

Using the *management switchboard* on the API some dynamic settings can be switched on and off without restarting the Alerta server daemon.

Currently, there is only one setting that can be toggled in this way and it is the Auto-refresh allow switch.

Auto-Refresh Allow

The Alerta Web UI will automatically referesh the list of alerts in the alert console every 5 seconds.

If for whatever reason, the Alerta API is experiencing heavy load the `auto_refresh_allow` switch can be turned off and the Web UI will respect that and switch to manual refresh mode. The Alerta web UI will start auto-refereshing again if the `auto_refresh_allow` switch is turned back on.

1.10 Deployment

1.10.1 WSGI Server

There are many ways to deploy Alerta. It can be run as `alertad` during development or testing but when run in a production environment, it should always be deployed as a WSGI application. See the list of *real world* examples below for different ways to run Alerta as a WSGI application.

When deploying with Apache mod_wsgi, be aware that by default Apache strips the Authentication header. This will cause you to receive "Missing authorization API Key or Bearer Token" errors. This can be fixed by setting `WSGIPassAuthorization On` in the configuration file for the site.

1.10.2 Web Proxy

Running the Alerta API behind a web proxy can greatly simplify the Web UI setup which means you can completely avoid the potential for any cross-origin issues.

Also, if you run the API on an HTTPS/SSL endpoint then it can reduce the possibility of mixed content errors when a web application hosted on a HTTP endpoint tries to access resources on an HTTPS endpoint.

Example API configuration (extract)

This example nginx server is configured to serve the web UI from the root `/` path and reverse-proxy API requests to `/api` to the WSGI application running on port 8080:

```
server {

        listen 80 default_server deferred;

        access_log /dev/stdout main;

        location /api/ {
                proxy_pass http://backend/;
                proxy_set_header Host $host:$server_port;
                proxy_set_header X-Real-IP $remote_addr;
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        }

        location / {
```

(continues on next page)

(continued from previous page)

```
                root /app;
        }
}

upstream backend {
        server localhost:8080 fail_timeout=0;
}
```

The web UI configuration file `config.js` for this setup would simply be `/api` for the `endpoint` value, as follows:

```
'use strict';

angular.module('config', [])
  .constant('config', {
    'endpoint'    : "/api",
    'provider'    : "basic"
  });
```

1.10.3 Static Website

The Alerta web UI is just a directory of static assets that can be served from any location. An easy and cheap way to serve the web UI is from an Amazon S3 bucket as a static website.

Note: Serving the Alerta web UI from a static web hosting site **will not work** unless that domain is listed in the `CORS_ORIGINS` Alerta API server configuration settings.

1.10.4 Authentication & SSL

Alerta supports several authentication mechanisms for both the API and the web UI and some key features of the web UI, like watching alerts, are only available if authentication is enabled.

The API can be secured using *API Keys* and the web UI can be secured using *Basic Auth* or an *OAuth* provider from either Google or Github.

If you plan to make the web UI accessible from a public URL it is strongly advised to *enforce authentication* and use HTTPS/SSL connections to the Alerta API to protect private alert data.

1.10.5 Authorisation & Customer Views

To restrict access to certain features use *roles* and *customer views*.

1.10.6 Scalability

Alerta can scale horizontally, in the same way any other web application scales horizontally – a load balancer handles the HTTP requests and distributes those requests between all available application servers.

Note: If using multiple API servers ensure the same `SECRET_KEY` is used across all servers otherwise there will be problems with web UI user logins.

1.10.7 High Availability

To achieve high system availability the Alerta API should be deployed to scale out *horizontally* and the MongoDB database should be deployed as a replica set.

1.10.8 House Keeping

There are some jobs that should be run periodically to keep the Alerta console clutter free. To timeout *expired* alerts and delete old *closed* alerts you need to trigger housekeeping.

This can be done with the `alerta` command-line tool:

```
$ alerta housekeeping
```

This was not supported by earlier versions of the command-line tool and cURL has to be used to access `/management/housekeeping`.

The API key needs an admin scope if AUTH_REQUIRED is set to True.

It is suggested that you run housekeeping at regular intervals via `cron`. Every minute is a suitable interval.

By default, when you run housekeeping, Alerta will remove any alerts that have been expired or closed for 2 hours and any info messages that are 12 hours old. In some cases, these retention periods may be too long or too short for your needs. Bear in mind that Alerta is intended to reflect the here and now, so long deletion thresholds should be avoided. Where you do need to depart from the defaults, you can specify like this:

    $ alerta housekeeping –expired 2 –info 12

In earlier versions of Alerta, a script called housekeepingAlerts.js was used for housekeeping. This is now deprecated.

Heartbeats can be sent from any source to ensure that a system is 'alive'. To generate alerts for stale heartbeats the `alerta` command-line tool can be used:

```
$ alerta heartbeats --alert
```

Again, this should be run at regular intervals via `cron` or some other scheduler.

1.10.9 Management & Metrics

Use the management endpoint `/management/status` to keep track of realtime statistics on the performance of the Alerta API like alert counts and average processing time. For convenience, these statistics can be viewed in the *About* page of the Alerta web UI or using the `alerta` command-line tool status command.

1.10.10 Web UI Analytics

Google analytics can be used to track usage of the Alerta web UI console. Just create a new tracking code with the Google analytics console and add it to the `config.js` web console configuration file:

```
'use strict';

angular.module('config', [])
  .constant('config', {
    'endpoint'     : "/api",
    'provider'     : "basic",
    'tracking_id' : "UA-NNNNNN-N"  // Google Analytics tracking ID
  });
```

1.10.11 Real World Examples

Below are several different examples of how to run Alerta in production from a Debian vagrant box, an AWS EC2 instance, Heroku PaaS to a Docker container.

- Vagrant - deploy Alerta stand-alone or with Nagios, Zabbix, Riemann, Sensu or Kibana
- Heroku - deploy the Alerta API and the web ui to Heroku PaaS
- AWS EC2 - deploy Alerta to EC2 using AWS Cloudformation
- Docker - deploy Alerta to a docker container
- Docker Alpine - full Alerta installation (including Mongo) based on Alpine Linux
- Packer - deploy Alerta to EC2 using Amazon AMIs
- Flask deploy - deploy Alerta as a generic Flask app
- Ansible - deploy Alerta using ansible on Centos 7
- Terraform - single instance of alerta for quick demo on AWS
- Puppet - Alerta recipe on top of cfweb module

1.11 Customer Views

Multitenancy is acheived using Customer views that are a way of ensuring logged in users only see alerts that relate to their organisation.

This is for Alerta deployments that are used to manage multiple customer sites.

1.11.1 Roles

The role of an API key is assumed to be "user" if it is a customer generated key. The role of admin

You can have a customer/user API key and an admin API key, but not a customer/admin API key – this makes no sense.

1.11.2 How it works

A new top-level alert attribute called `customer` is used to partition alerts for particular customers within the same alert database.

When a user logs in to the Alert console, a customer lookup is done to determine what customer value should be assigned to that user.

The customer value is then used as an implicit additional filter for all alert and heartbeat queries.

It is also assigned to any API keys generated by that user and the `customer` field is automatically used whenever that API key is used to generate or query for alerts.

1.11.3 Configuration

To configure customer views follow these three easy steps:

1. Authentication must be enforced and customer views enabled so in `alertad.conf`:

```
AUTH_REQUIRED = True
CUSTOMER_VIEWS = True
```

2. Define administrators that will have a global view of all customers and will have no restrictions on generating API keys or blackout periods, so in `alertad.conf`:

```
ADMIN_USERS = ['foo@bar.com']
```

3. Populate the Customer Lookup table in the web console to map Google email domains or Github/Gitlab orgs and groups to customers

1.11.4 Web Console for Users

Users that have a *customer view* are limited to what they can do in the web console (and via the API). They cannot create other users, configure blackout periods or modify the customer lookup table.

1.11.5 Web Console for Administrators

Administrators are not limited in what they can do in the web console (or via the API). Importantly, they can configure the customer lookup table.

1.12 Conventions

Always favour convention over configuration. And any configuration should have sensible defaults.

1.12.1 Naming Conventions

Resources

The key alert attribute name of `resource` was specifically chosen so as not to be host centric. A resource *can* be a hostname, but it might also be an EC2 instance ID, a Docker container ID or some other type of non-host unique identifier.

Environments & Services

The environment attribute is used to namespace the alert resource. This allows you to have two resources with the same name (eg. `web01`) but that are differentiated by their environments (eg. `Production` and `Development`).

Choose a set of environments and enforce them. ie. `PROD`, `DEV` or `Production`, `Development` but not both. The same for services eg. `MobileAPI`, `Mobile-API` and `mobile api` are all valid but needlessly different and impossible to query for consistently or generate aggregate metrics for.

Note that the service attribute is a **list** because it is common for infrastructure (ie. some resource) to be used by more than one service. That is, if a component failure occurs that problem could cause an outage in multiple services.

Event Names

It can be useful to define a convention when it comes to naming events. Possible options are:

- Camel case - `DiskUtilHigh`
- Hierarchy - `NW:INTERFACE:DOWN`
- SNMP - `cpuAlarmHigh`

Querying for all Disk utilisation alerts using the `alerta` CLI is then relatively straight-forward:

```
$ alerta query --filter event=~DiskUtil
```

Event Groups

Another consideration is to ensure you make use of the event group which gives you the ability to group related alerts.

Some suggested event groups with possible events are listed below.

Event Groups	Events (examples)
`Service`	failures with entire services
`Application`	errors from application logs
`OS`	disk space, time sync failing
`Performance`	system load, swap utilisation high
`Configuration`	config mgmt tool alerts eg. Puppet or Chef
`Web`	web server errors
`Syslog`	unix system log messages
`Hardware`	hardware errors
`Storage`	NFS, SAN, NAS storage infrastructure
`Database`	database errors, table space utilisation
`Security`	security/authorization messages
`Network`	network devices and infrastructure
`Cloud`	cloud-based services or infrastructure

Querying for all performance-related alerts using the `alerta` CLI could then become:

```
$ alerta query --filter group=Performance
```

1.12.2 Severity Levels

Agree on a subset of severity levels and be consistent with what they mean. For example, if severity levels are used consistently then integrating with a paging or email system becomes easier.

Severity	Service Level	Notification
`critical`	service unavailable	immediate page out
`major`	service impaired still available	page during business hours
`minor`	component failure	email only
`warning`	everything else	consolidate into daily email

1.12.3 Enforcing Conventions

Once a set of naming conventions are agreed, they can be enforced by using a simple *pre-receive* plugin.

A full working example called reject can be found in the plugins directory of the project code repository and is installed by default. The server configuration settings `ORIGIN_BLACKLIST` and `ALLOWED_ENVIRONMENTS` can be used to tailor it for your circumstances.

1.13 Development

1.13.1 Python SDK

Alerta is developed in Python so the Python SDK is a core component of the monitoring system.

Installation

Install using pip:

```
$ pip install alerta
```

Install master branch directly from GitHub:

```
$ pip install git+https://github.com/alerta/python-alerta-client.git@master
```

Examples

Initialise the alerta API client:

```
>>> from alertaclient.api import Client
>>> client = Client(endpoint='https://alerta-api.herokuapps.com', key='demo-key')
```

Send an alert:

```
>>> client.send_alert(resource='foo', event='bar')
...
alertaclient.exceptions.UnknownError: [POLICY] Alert environment does not match one␣
↪of Production, Development
```

Send an alert again, this time including the required `environment` and `service`:

```
>>> client.send_alert(resource='foo', event='bar', environment='Development',␣
↪service=['Web'])
('fd3ecad4-6558-4ec7-96cc-aff6cdf1fabc', Alert(id='fd3ecad4-6558-4ec7-96cc-
↪aff6cdf1fabc', environment='Development', resource='foo', event='bar', severity=
↪'normal', status='closed', customer=None), None)
```

Query for the alert just sent, by alert ID:

```
>>> client.get_alert('fd3ecad4-6558-4ec7-96cc-aff6cdf1fabc')
Alert(id='fd3ecad4-6558-4ec7-96cc-aff6cdf1fabc', environment='Development', resource=
↪'foo', event='bar', severity='normal', status='closed', customer=None)
```

Search for alerts by attributes:

```
>>> client.get_alerts([('resource','foo'),('environment','Development')])
[Alert(id='fd3ecad4-6558-4ec7-96cc-aff6cdf1fabc', environment='Development', resource=
↪'foo', event='bar', severity='normal', status='closed', customer=None)]
```

Send a heartbeat:

```
>>> client.heartbeat(origin='baz')
Heartbeat(id='98c220e6-5148-4b19-8ae8-e1c078b7d68c', origin='baz', create_
↪time=datetime.datetime(2018, 9, 6, 8, 48, 48, 817000), timeout=86400, customer=None)
```

For more details, visit the Alerta Python SDK page.

1.13.2 Ruby SDK

The Ruby SDK is a work-in-progress. For more details, visit the Alerta Ruby SDK page.

1.13.3 Haskell SDK

This SDK supplies bindings to the Alerta REST API so that it can be used from Haskell.

For more details, visit the Haskell Package page.

1.13.4 Gource Visualization

View the development of Alerta over the years as an animated tree Gource visualization.

1.14 Getting Started

The following tutorials are designed to get you started deploying and using Alerta in common scenarios.

1.14.1 Tutorials

- Deploy an Alerta Server
- Alert timeouts, heartbeats and housekeeping
- Use plugins to enhance Alerta
- Alerts explored in-depth
- Suppressing Alerts using Blackouts
- Authentication & Authorization

Note: If you require help with any of the above tutorials post a question on Gitter.

1.14.2 How-to Guides

How To Monitor Nagios Alerts with Alerta on Ubuntu 16.04 by Vadym Kalsin

How To Monitor Zabbix Alerts with Alerta on CentOS 7 by Vadym Kalsin

OpenSource Metric Based Monitoring by Christian Eichelmann

Installing Alerta on Debian | Ubuntu in Cyber Defence Monitoring Course Suite (CDMCS)

SRE Engineering Practice – Alarm Based on Time Series to Store Data on Docker Mail

Simple tutorial for wetting your appetite on using alerta.io by deeplook

1.15 Resources

1.15.1 Webinars & Slides

How to avoid failing at failure detection by Alex Tavgen, Technical Architect at Playtech

Winning the metrics battle (finally) [Slides] at Velocity Europe 2012

1.15.2 Articles

Winning the metrics battle by Simon Hildrew and Nick Satterly, The Guardian

Never fail twice by Alex Tavgen

Make better use of Prometheus with Grafana, Telegraf, and Alerta [$] by Martin Loschwitz, Linux Magazin [DE]

Grafana, Telegraf, Alerta – Prometheus besser nutzen (in German) [$] by Martin Loschwitz, Linux Magazin [DE]

Riemann Learnings by Antonio Terreno, CTO The Labrador,

1.15.3 Papers

Frankenstack: Toward Real-time Red Team Feedback by Markus Kont, Mauno Pihelgas, Bernhards Blumbergs of NATO Cooperative Cyber Defence Centre of Excellence and Kaie Maennel and Toomas Lepik of the Tallinn University of Technology at 2017 IEEE Military Communications Conference

EVE and ADAM: Situation Awareness Tools for NATO CCDCOE Cyber Exercises by Francisco Jesús Rubio Melón of Ingeniería de Sistemas para la Defensa de España, Teemu Uolevi Väisänen of VTT Technical Research Centre of Finland and Mauno Pihelgas of NATO Cooperative Cyber Defence Centre of Excellence

1.15.4 References

Event Correlation Engine [Master's Thesis] by Andreas Müller (2009) at Institut für Technische Informatik und Kommunikationsnetze

ANSI/ISA 18.2 Management of Alarm Systems for the Process Industries by American National Standards Institute

1.16 API Reference

Resource Types

Note: All `datetime` parameters must be in ISO 8601 format in UTC time (using time zone designator "Z") and expressed to millisecond precision as recommended by the W3C Date and Time Formats Note eg. `2017-06-19T11:16:19.744Z`

1.16.1 Alerts

Create an alert

Creates a new alert, or updates an existing alert if the `environment- resource-event` combination already exists.

```
POST /alert
```

Input

Name	Type	Description
`resource`	string	**Required** resource under alarm
`event`	string	**Required** event name
`environment`	string	environment, used to namespace the resource
`severity`	string	see severity_table table
`correlate`	list	list of related event names
`status`	string	see status_table table
`service`	list	list of effected services
`group`	string	used to group events of similar type
`value`	string	event value
`text`	string	freeform text description
`tags`	set	set of tags
`attributes`	dict	dictionary of key-value pairs
`origin`	string	monitoring component that generated the alert
`type`	string	event type
`createTime`	datetime	time alert was generated at the origin
`timeout`	integer	seconds before alert is considered stale
`rawData`	string	unprocessed raw data

Note: Only `resource` and `event` are mandatory. The `status` can be dynamically assigned by the Alerta API based on the `severity`.

Example Request

```
$ curl -XPOST http://localhost:8080/alert \
-H 'Authorization: Key demo-key' \
-H 'Content-type: application/json' \
-d '{
      "attributes": {
        "region": "EU"
      },
      "correlate": [
        "HttpServerError",
        "HttpServerOK"
      ],
      "environment": "Production",
      "event": "HttpServerError",
      "group": "Web",
      "origin": "curl",
      "resource": "web01",
      "service": [
        "example.com"
      ],
      "severity": "major",
      "tags": [
        "dc1"
      ],
      "text": "Site is down.",
```

(continues on next page)

(continued from previous page)

```
      "type": "exceptionAlert",
      "value": "Bad Gateway (501)"
    }'
```

Example Response

```
201 CREATED
```


```
{
  "alert": {
    "attributes": {
      "flapping": false,
      "ip": "127.0.0.1",
      "notify": true,
      "region": "EU"
    },
    "correlate": [
      "HttpServerError",
      "HttpServerOK"
    ],
    "createTime": "2018-01-27T21:00:12.999Z",
    "customer": null,
    "duplicateCount": 0,
    "environment": "Production",
    "event": "HttpServerError",
    "group": "Web",
    "history": [
      {
        "event": "HttpServerError",
        "href": "http://localhost:8080/alert/17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
        "id": "17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
        "severity": "major",
        "status": null,
        "text": "Site is down.",
        "type": "severity",
        "updateTime": "2018-01-27T21:00:12.999Z",
        "value": "Bad Gateway (501)"
      }
    ],
    "href": "http://localhost:8080/alert/17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
    "id": "17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
    "lastReceiveId": "17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
    "lastReceiveTime": "2018-01-27T21:00:13.070Z",
    "origin": "curl",
    "previousSeverity": "indeterminate",
    "rawData": null,
    "receiveTime": "2018-01-27T21:00:13.070Z",
    "repeat": false,
    "resource": "web01",
    "service": [
      "example.com"
    ],
    "severity": "major",
    "status": "open",
```


(continues on next page)

(continued from previous page)


```
    "tags": [
      "dc1"
    ],
    "text": "Site is down.",
    "timeout": 86400,
    "trendIndication": "moreSevere",
    "type": "exceptionAlert",
    "value": "Bad Gateway (501)"
  },
  "id": "17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
  "status": "ok"
}
```


Example Response (during blackout period)

```
202 ACCEPTED
```


```
{
  "message": "Suppressed alert during blackout period",
  "id": "1711c57e-5c6a-4c39-881b-9d8d174feafe",
  "status": "ok"
}
```


Retrieve an alert

Retrieves an alert with the given alert ID.

```
GET /alert/:id
```

Example Request

```
$ curl http://localhost:8080/alert/17d8e7ea-b3ba-4bb1-9c5a-29e60865f258 \
-H 'Authorization: Key demo-key'
```

Example Response

```
200 OK
```


```
{
  "alert": {
    "attributes": {
      "flapping": false,
      "ip": "127.0.0.1",
      "notify": true,
      "region": "EU"
    },
    "correlate": [
      "HttpServerError",
```


(continues on next page)

(continued from previous page)


```
      "HttpServerOK"
    ],
    "createTime": "2018-01-27T21:00:12.999Z",
    "customer": null,
    "duplicateCount": 0,
    "environment": "Production",
    "event": "HttpServerError",
    "group": "Web",
    "history": [
      {
        "event": "HttpServerError",
        "href": "http://localhost:8080/alert/17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
        "id": "17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
        "severity": "major",
        "status": null,
        "text": "Site is down.",
        "type": "severity",
        "updateTime": "2018-01-27T21:00:12.999Z",
        "value": "Bad Gateway (501)"
      }
    ],
    "href": "http://localhost:8080/alert/17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
    "id": "17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
    "lastReceiveId": "17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
    "lastReceiveTime": "2018-01-27T21:00:13.070Z",
    "origin": "curl",
    "previousSeverity": "indeterminate",
    "rawData": null,
    "receiveTime": "2018-01-27T21:00:13.070Z",
    "repeat": false,
    "resource": "web01",
    "service": [
      "example.com"
    ],
    "severity": "major",
    "status": "open",
    "tags": [
      "dc1"
    ],
    "text": "Site is down.",
    "timeout": 86400,
    "trendIndication": "moreSevere",
    "type": "exceptionAlert",
    "value": "Bad Gateway (501)"
  },
  "status": "ok",
  "total": 1
}
```


Set alert status

Sets the status of an alert, and logs the status change to the alert history.

```
PUT /alert/:id/status
```

Input

Name	Type	Description
`status`	string	**Required** New status from `open`, `assign`, `ack`, `closed`, `expired`
`text`	string	reason for status change

Example Request

```
$ curl -XPUT http://localhost:8080/alert/17d8e7ea-b3ba-4bb1-9c5a-29e60865f258/status \
-H 'Authorization: Key demo-key' \
-H 'Content-type: application/json' \
-d '{
      "status": "ack",
      "text": "disk needs replacing."
    }'
```

Tag and untag alerts

Adds or removes tag values from the set of tags for an alert.

```
PUT /alert/:id/tag
PUT /alert/:id/untag
```

Input

Name	Type	Description
`tags`	set	tags to add or remove

Example Request

```
$ curl -XPUT http://localhost:8080/alert/17d8e7ea-b3ba-4bb1-9c5a-29e60865f258/tag \
-H 'Authorization: Key demo-key' \
-H 'Content-type: application/json' \
-d '{
      "tags": [
        "linux",
        "linux2.6",
        "dell"
      ]
    }'
```

Update alert attributes

Adds, deletes or modifies alert attributes. To delete an attribute assign "null" to the attribute.

```
PUT /alert/:id/attributes
```

Input

Name	Type	Description
`attributes`	dict	dictionary of key-value attributes

Example Request

```
$ curl -XPUT http://localhost:8080/alert/17d8e7ea-b3ba-4bb1-9c5a-29e60865f258/
↪attributes \
-H 'Authorization: Key demo-key' \
-H 'Content-type: application/json' \
-d '{
      "attributes": {
        "incidentKey": "1234abcd",
        "ip": "10.1.1.1",
        "region": null
      }
    }'
```

Delete an alert

Permanently deletes an alert. It cannot be undone.

```
DELETE /alert/:id
```

Example Request

```
$ curl -XDELETE http://localhost:8080/alert/17d8e7ea-b3ba-4bb1-9c5a-29e60865f258 \
-H 'Authorization: Key demo-key'
```

Search alerts

Find alerts using various alert attributes or a mongo-type query parameter to filter results.

```
GET /alerts
```

Parameters

Name	Type	Description
`<attr>`	string	any alert attribute. eg. `status=open`
`q` (*)	json	mongo query (see Mongo Query Operators)
`fields` (*)	list	show or hide alert attributes
`from-date`	datetime	`lastReceiveTime` > `from-date`
`to-date`	datetime	`lastReceiveTime` <= `to-date` (now)
`sort-by`	string	attr to sort by (default:`lastReceiveTime`)
`reverse`	boolean	change direction of default sort order
`page`	integer	number between 1 and total pages (default: 1)
`page-size`	integer	default: 1000 (set `DEFAULT_PAGE_SIZE`)

The `<attr>` search parameter works as follows:

- Any combination of valid alert attributes can be used to narrow down results.
- Search syntax is = (equals), `!=` (not equals), =~ (regex match) and `!=~` (regex exclude).
- When searching for alert `id` the query will attempt to match against `id` and `lastReceiveId`. The "short id" (ie. first 8-characters) can be used. eg. `id=ba358336` instead of `id=ba358336-802d-40ee-8ace-bf5fa8529280`.
- Use "dot notation" to query custom attributes. eg. `attributes.city=Berlin`
- Alert `history` is limited to the 100 most recent status or severity changes. (set using `HISTORY_LIMIT`)
- If "customer views" is enabled then the appropriate `customer` filter for that user will be automatically applied.

The `q` search parameter works as follows:

- Any valid JSON-compliant Mongo query using Mongo Query Operators. Useful when there is a need to "or" several attributes to get the required search filter eg. `q={"$or":[{"service":"Web"}, {"resource":{"$regex":"web"}}]}`

Warning: In the next major release of Alerta (5.0) the `fields` parameter will be removed. Also the `q` search term may change and no longer be mongo-specific.

Example Request

```
$ curl http://localhost:8080/alerts?group=Web \
-H 'Authorization: Key demo-key'
```

Example Response

```
200 OK
```

```
{
  "alerts": [
    {
      "attributes": {
        "flapping": false,
        "incidentKey": "1234abcd",
        "ip": "10.1.1.1",
        "notify": true
      },
      "correlate": [
        "HttpServerError",
        "HttpServerOK"
      ],
      "createTime": "2018-01-27T21:00:12.999Z",
      "customer": null,
      "duplicateCount": 0,
      "environment": "Production",
      "event": "HttpServerError",
      "group": "Web",
      "history": [
        {
          "event": "HttpServerError",
          "href": "http://localhost:8080/alert/17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
          "id": "17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
          "severity": "major",
          "status": null,
          "text": "Site is down.",
          "type": "severity",
          "updateTime": "2018-01-27T21:00:12.999Z",
          "value": "Bad Gateway (501)"
        },
        {
          "event": "HttpServerError",
          "href": "http://localhost:8080/alert/17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
          "id": "17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
          "severity": null,
          "status": "ack",
          "text": "disk needs replacing.",
          "type": "status",
          "updateTime": "2018-01-27T21:04:42.412Z",
          "value": null
        }
      ],
      "href": "http://localhost:8080/alert/17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
      "id": "17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
      "lastReceiveId": "17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
      "lastReceiveTime": "2018-01-27T21:00:13.070Z",
      "origin": "curl",
      "previousSeverity": "indeterminate",
      "rawData": null,
      "receiveTime": "2018-01-27T21:00:13.070Z",
      "repeat": false,
      "resource": "web01",
      "service": [
        "example.com"
      ],
      "severity": "major",
```


(continues on next page)

(continued from previous page)


```
      "status": "ack",
      "tags": [
        "dc1",
        "linux",
        "linux2.6",
        "dell"
      ],
      "text": "Site is down.",
      "timeout": 86400,
      "trendIndication": "moreSevere",
      "type": "exceptionAlert",
      "value": "Bad Gateway (501)"
    }
  ],
  "autoRefresh": true,
  "lastTime": "2018-01-27T21:00:13.070Z",
  "more": false,
  "page": 1,
  "pageSize": 1000,
  "pages": 1,
  "severityCounts": {
    "major": 1
  },
  "status": "ok",
  "statusCounts": {
    "ack": 1
  },
  "total": 1
}
```


List all alert history

Returns a list of alert severity and status changes.

```
GET /alerts/history
```

Parameters

Name	Type	Description
<attr>	string	

Example Request

```
$ curl http://localhost:8080/alerts/history?service=example.com \
-H 'Authorization: Key demo-key'
```

Example Response

```
200 OK
```


```
{
  "history": [
    {
      "attributes": {
        "flapping": false,
        "incidentKey": "1234abcd",
        "ip": "10.1.1.1",
        "notify": true
      },
      "customer": null,
      "environment": "Production",
      "event": "HttpServerError",
      "group": "Web",
      "href": "http://localhost:8080/alert/17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
      "id": "17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
      "origin": "curl",
      "resource": "web01",
      "service": [
        "example.com"
      ],
      "severity": "major",
      "tags": [
        "dc1",
        "linux",
        "linux2.6",
        "dell"
      ],
      "text": "Site is down.",
      "type": "severity",
      "updateTime": "2018-01-27T21:00:12.999Z",
      "value": "Bad Gateway (501)"
    },
    {
      "attributes": {
        "flapping": false,
        "incidentKey": "1234abcd",
        "ip": "10.1.1.1",
        "notify": true
      },
      "customer": null,
      "environment": "Production",
      "event": "HttpServerError",
      "group": "Web",
      "href": "http://localhost:8080/alert/17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
      "id": "17d8e7ea-b3ba-4bb1-9c5a-29e60865f258",
      "origin": "curl",
      "resource": "web01",
      "service": [
        "example.com"
      ],
      "status": "ack",
      "tags": [
        "dc1",
```


(continues on next page)

(continued from previous page)


```
        "linux",
        "linux2.6",
        "dell"
      ],
      "text": "disk needs replacing.",
      "type": "status",
      "updateTime": "2018-01-27T21:04:42.412Z"
    }
  ],
  "status": "ok",
  "total": 2
}
```


Get severity and status counts for alerts

Returns a count of alerts grouped by severity and status.

```
GET /alerts/count
```

Parameters

Name	Type	Description
<attr>	string	

Example Request

```
$ curl http://localhost:8080/alerts/count?environment=Production \
-H 'Authorization: Key demo-key'
```

Example Response

```
200 OK
```


```
{
  "severityCounts": {
    "critical": 1,
    "major": 1
  },
  "status": "ok",
  "statusCounts": {
    "ack": 1,
    "open": 1
  },
  "total": 2
}
```

Top 10 alerts by resource

Returns a list of the top 10 resources grouped by an alert attribute. By default it is grouped by `event` but this can be any valid attribute.

```
GET /alerts/top10/count
GET /alerts/top10/flapping
```

Parameters

Name	Type	Description
`<attr>`	string	
`q`	dict	mongo query see Mongo Query Operators
`group-by`	string	any valid alert attribute. Default:`event`

Example Request

```
$ curl http://localhost:8080/alerts/top10?group-by=group \
-H 'Authorization: Key demo-key'
```

Example Response

```
200 OK
```

```
{
  "status": "ok",
  "top10": [
    {
      "count": 2,
      "duplicateCount": 0,
      "environments": [
        "Production"
      ],
      "group": "Web",
      "resources": [
        {
          "href": "http://localhost:8080/alert/0099bae5-9683-48a1-a49d-f566fe143770",
          "id": "0099bae5-9683-48a1-a49d-f566fe143770",
          "resource": "web02"
        },
        {
          "href": "http://localhost:8080/alert/e9fb05a0-b65c-4faa-8868-6f06a74a2b5b",
          "id": "e9fb05a0-b65c-4faa-8868-6f06a74a2b5b",
          "resource": "web01"
        }
      ],
      "services": [
        "example.com"
      ]
    }
```

(continues on next page)

(continued from previous page)

```
  ],
  "total": 1
}
```

1.16.2 Environments

An environment cannot be created – it is a dynamically derived resource based on existing alerts.

List all environments

Returns a list of environments and an alert count for each.

```
GET /environments
```

Parameters

Name	Type	Description
<attr>	string	

Example Request

```
$ curl http://localhost:8080/environments \
-H 'Authorization: Key demo-key'
```

Example Response

```
200 OK
```

```
{
  "environments": [
    {
      "count": 2,
      "environment": "Production"
    }
  ],
  "status": "ok",
  "total": 1
}
```

1.16.3 Services

A service cannot be created – it is a dynamically derived resource based on existing alerts.

List all services

Returns a list of services grouped by environment and an alert count for each.

```
GET /services
```

Parameters

Name	Type	Description
<attr>	string	

Example Request

```
$ curl http://localhost:8080/services?environment=Production \
-H 'Authorization: Key demo-key'
```

Example Response

```
200 OK
```

```
{
  "services": [
    {
      "count": 2,
      "environment": "Production",
      "service": "example.com"
    }
  ],
  "status": "ok",
  "total": 1
}
```

1.16.4 Tags

A tag cannot be created – it is a dynamically derived resource based on existing alerts.

List all tags

Returns a list of tags grouped by environment and an alert count for each.

```
GET /tags
```

Parameters

Name	Type	Description
<attr>	string	

Example Request

```
$ curl http://localhost:8080/tags?environment=Production \
-H 'Authorization: Key demo-key'
```

Example Response

```
200 OK
```


```
{
    "status": "ok",
    "tags": [
        {
            "count": 2,
            "environment": "Production",
            "tag": "linux"
        },
        {
            "count": 1,
            "environment": "Production",
            "tag": "dc2"
        },
        {
            "count": 1,
            "environment": "Production",
            "tag": "hp"
        },
        {
            "count": 2,
            "environment": "Production",
            "tag": "dell"
        },
        {
            "count": 2,
            "environment": "Production",
            "tag": "dc1"
        },
        {
            "count": 2,
            "environment": "Production",
            "tag": "linux2.6"
        }
    ],
    "total": 6
}
```


1.16.5 Blackout Periods

Create a blackout

Create a new blackout period for alert suppression.

```
POST /blackout
```

Input

Name	Type	Description
`environment`	string	**Required**
`resource`	string	
`service`	list	
`event`	string	
`group`	string	
`tags`	list	
`startTime`	datetime	start time of blackout. Default: now
`endTime`	datetime	end time. Default: now + `BLACKOUT_DURATION`
`duration`	integer	seconds. Default: `BLACKOUT_DURATION` Only used if `endTime` not defined

Example Request

```
$ curl -XPOST http://localhost:8080/blackout \
-H 'Authorization: Key demo-key' \
-H 'Content-type: application/json' \
-d '{
      "environment": "Production",
      "service": ["example.com"],
      "group": "Web"
    }'
```

Example Response

```
201 CREATED
```

```
{
  "blackout": {
    "customer": null,
    "duration": 3600,
    "endTime": "2018-01-27T22:10:31.705Z",
    "environment": "Production",
    "event": null,
    "group": "Web",
    "href": "http://localhost:8080/blackout/79d12b79-45b9-4419-80e4-1f6c17478eb6",
    "id": "79d12b79-45b9-4419-80e4-1f6c17478eb6",
    "priority": 3,
    "remaining": 3599,
    "resource": null,
    "service": [
      "example.com"
    ],
    "startTime": "2018-01-27T21:10:31.705Z",
    "status": "active",
    "tags": []
```

(continues on next page)

(continued from previous page)

```
  },
  "id": "79d12b79-45b9-4419-80e4-1f6c17478eb6",
  "status": "ok"
}
```

List all blackouts

Returns a list of blackout periods, including expired blackouts.

```
GET /blackouts
```

Example Request

```
$ curl http://localhost:8080/blackouts \
-H 'Authorization: Key demo-key'
```

Example Response

```
200 OK
```

```
{
  "blackouts": [
    {
      "customer": null,
      "duration": 3600,
      "endTime": "2018-01-27T22:10:31.705Z",
      "environment": "Production",
      "event": null,
      "group": "Web",
      "href": "http://localhost:8080/blackout/79d12b79-45b9-4419-80e4-1f6c17478eb6",
      "id": "79d12b79-45b9-4419-80e4-1f6c17478eb6",
      "priority": 3,
      "remaining": 3345,
      "resource": null,
      "service": [
        "example.com"
      ],
      "startTime": "2018-01-27T21:10:31.705Z",
      "status": "active",
      "tags": []
    },
    {
      "customer": null,
      "duration": 3600,
      "endTime": "2018-01-27T22:14:32.377Z",
      "environment": "Development",
      "event": null,
      "group": "Performance",
      "href": "http://localhost:8080/blackout/c17832d4-c477-4eb1-b2d5-662e7a3600be",
      "id": "c17832d4-c477-4eb1-b2d5-662e7a3600be",
```

(continues on next page)

(continued from previous page)

```
      "priority": 5,
      "remaining": 3586,
      "resource": null,
      "service": [],
      "startTime": "2018-01-27T21:14:32.377Z",
      "status": "active",
      "tags": []
    }
  ],
  "status": "ok",
  "total": 2
}
```

Delete a blackout

Permanently deletes a blackout period. It cannot be undone.

```
DELETE /blackout/:id
```

Example Request

```
$ curl -XDELETE http://localhost:8080/blackout/c17832d4-c477-4eb1-b2d5-662e7a3600be \
-H 'Authorization: Key demo-key'
```

1.16.6 Heartbeats

Send a heartbeat

Creates a new heartbeat, or updates an existing heartbeat if a heartbeat from the `origin` already exists.

```
POST /heartbeat
```

Input

Name	Type	Description
`origin`	string	
`tags`	list	
`timeout`	integer	Seconds.

Example Request

```
$ curl -XPOST http://localhost:8080/heartbeat \
-H 'Authorization: Key demo-key' \
-H 'Content-type: application/json' \
-d '{
      "origin": "cluster05",
```

(continues on next page)

(continued from previous page)

```
      "timeout": 120,
      "tags": ["db05", "dc2"]
    }'
```

Example Response

```
201 CREATED
```


```
{
  "heartbeat": {
    "createTime": "2018-01-27T21:15:38.675Z",
    "customer": null,
    "href": "http://localhost:8080/heartbeat/1a3f2e0a-3c65-4199-84ae-a21fb892ccc0",
    "id": "1a3f2e0a-3c65-4199-84ae-a21fb892ccc0",
    "latency": 0,
    "origin": "cluster05",
    "receiveTime": "2018-01-27T21:15:38.675Z",
    "since": 0,
    "status": "ok",
    "tags": [
      "db05",
      "dc2"
    ],
    "timeout": 120,
    "type": "Heartbeat"
  },
  "id": "1a3f2e0a-3c65-4199-84ae-a21fb892ccc0",
  "status": "ok"
}
```


Get a heartbeat

Retrieves a heartbeat based on the heartbeat ID.

```
GET /heartbeat/:id
```

Example Request

```
$ curl http://localhost:8080/heartbeat/1a3f2e0a-3c65-4199-84ae-a21fb892ccc0 \
-H 'Authorization: Key demo-key'
```

Example Response

```
200 OK
```


```
{
  "heartbeat": {
    "createTime": "2018-01-27T21:15:38.675Z",
```


(continues on next page)

(continued from previous page)


```
    "customer": null,
    "href": "http://localhost:8080/heartbeat/1a3f2e0a-3c65-4199-84ae-a21fb892ccc0",
    "id": "1a3f2e0a-3c65-4199-84ae-a21fb892ccc0",
    "latency": 0,
    "origin": "cluster05",
    "receiveTime": "2018-01-27T21:15:38.675Z",
    "since": 34,
    "status": "ok",
    "tags": [
      "db05",
      "dc2"
    ],
    "timeout": 120,
    "type": "Heartbeat"
  },
  "status": "ok",
  "total": 1
}
```


List all heartbeats

Returns a list of all heartbeats.

```
GET /heartbeats
```

Example Request

```
$ curl http://localhost:8080/heartbeats \
-H 'Authorization: Key demo-key'
```

Example Response

```
200 OK
```


```
{
  "heartbeats": [
    {
      "createTime": "2018-01-27T21:17:13.922Z",
      "customer": null,
      "href": "http://localhost:8080/heartbeat/f5eb11ef-e02b-42f2-9013-6efca6eca22a",
      "id": "f5eb11ef-e02b-42f2-9013-6efca6eca22a",
      "latency": 0,
      "origin": "web02",
      "receiveTime": "2018-01-27T21:17:13.922Z",
      "since": 45,
      "status": "ok",
      "tags": [
        "linux",
        "dc1"
      ],
```


(continues on next page)

(continued from previous page)


```
      "timeout": 120,
      "type": "Heartbeat"
    },
    {
      "createTime": "2018-01-27T21:17:55.936Z",
      "customer": null,
      "href": "http://localhost:8080/heartbeat/e0582765-ee64-4944-8a94-1869a079d81f",
      "id": "e0582765-ee64-4944-8a94-1869a079d81f",
      "latency": 0,
      "origin": "cluster05",
      "receiveTime": "2018-01-27T21:17:55.936Z",
      "since": 3,
      "status": "ok",
      "tags": [
        "db05",
        "dc2"
      ],
      "timeout": 120,
      "type": "Heartbeat"
    }
  ],
  "status": "ok",
  "total": 2
}
```


Delete a heartbeat

Permanently deletes a heartbeat. It cannot be undone.

```
DELETE /heartbeat/:id
```

Example Request

```
$ curl -XDELETE http://localhost:8080/heartbeat/e0582765-ee64-4944-8a94-1869a079d81f \
-H 'Authorization: Key demo-key'
```

1.16.7 API Keys

Create an API key

Creates a new API key.

```
POST /key
```

Input

Name	Type	Description
`user`	string	username
`scopes`	string	`admin`, `write`, or `read`
`text`	string	freeform description text
`expireTime`	string	
`customer`	string	**Admin use only**

Example Request

```
$ curl -XPOST http://localhost:8080/key \
-H 'Authorization: Key demo-key' \
-H 'Content-type: application/json' \
-d '{
      "user": "admin@alerta.io",
      "scopes": ["write"],
      "text": "API key for external system"
    }'
```

Example Response

```
201 CREATED
```


```
{
  "data": {
    "count": 0,
    "customer": null,
    "expireTime": "2019-01-27T22:18:42.245Z",
    "href": "http://localhost:8080/key/_Jwm-qaGa0kBM9R1CyyQn-0qxLtBtij4ToQf6beL",
    "id": "ca931aec-4e56-496f-a8d6-be11d93ddaed",
    "key": "_Jwm-qaGa0kBM9R1CyyQn-0qxLtBtij4ToQf6beL",
    "lastUsedTime": null,
    "scopes": [
      "write"
    ],
    "text": "API key for external system",
    "type": "read-write",
    "user": "admin@alerta.io"
  },
  "key": "_Jwm-qaGa0kBM9R1CyyQn-0qxLtBtij4ToQf6beL",
  "status": "ok"
}
```


List all API keys

Returns a list of API keys.

```
GET /keys
```

Example Request

```
$ curl http://localhost:8080/keys \
-H 'Authorization: Key demo-key'
```

Example Response

```
200 OK
```


```
{
  "keys": [
    {
      "count": 0,
      "customer": null,
      "expireTime": "2019-01-27T22:18:42.245Z",
      "href": "http://localhost:8080/key/_Jwm-qaGa0kBM9R1CyyQn-0qxLtBtij4ToQf6beL",
      "id": "ca931aec-4e56-496f-a8d6-be11d93ddaed",
      "key": "_Jwm-qaGa0kBM9R1CyyQn-0qxLtBtij4ToQf6beL",
      "lastUsedTime": null,
      "scopes": [
        "write"
      ],
      "text": "API key for external system",
      "type": "read-write",
      "user": "admin@alerta.io"
    },
    {
      "count": 21,
      "customer": null,
      "expireTime": "2019-01-27T19:22:27.120Z",
      "href": "http://localhost:8080/key/demo-key",
      "id": "532c9b59-9e90-40d4-8a3b-887362a79e9c",
      "key": "demo-key",
      "lastUsedTime": "2018-01-27T22:19:04.113Z",
      "scopes": [
        "admin",
        "write",
        "read"
      ],
      "text": "Admin key created by alertad script",
      "type": "read-write",
      "user": "foo@foobar.com"
    }
  ],
  "status": "ok",
  "total": 2
}
```


Delete an API key

Permanently deletes an API key. It cannot be undone.

```
DELETE /key/:key
```

Example Request

```
$ curl -XDELETE http://localhost:8080/key/532c9b59-9e90-40d4-8a3b-
↪887362a79e9cO8rhJSKrdfQWXqRhvSwJQJRZg9yU0s2Z4VLP4855 \
-H 'Authorization: Key demo-key'
```

1.16.8 Users

Create a user

Creates a new Basic Auth user.

```
POST /auth/signup
```

Input

Name	Type	Description
name	string	
email	string	
password	string	
text	string	

Example Request

```
$ curl -XPOST http://localhost:8080/auth/signup \
-H 'Authorization: Key demo-key' \
-H 'Content-type: application/json' \
-d '{
      "name": "Joe Bloggs",
      "email": "joe.bloggs@example.com",
      "password": "secret",
      "text": "demo user"
    }'
```

Example Response

```
200 OK
```

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
↪eyJzdWIiOiI4Y2IwYjYyNC0zY2Q3LTQ1YjktOThhNS01ZGZhYzVmMDE2NmMiLCJyb2xlcyI6WyJ1c2VyIl0sImlzcyI6Imh0dHA
↪c5jpr8YksoJmoZ6KUwsYP5fgwZr-jdA4W3JUCbv1vXU"
}
```

Update a user

Updates the specified user by setting the values of the parameters passed. Any parameters not provided will be left unchanged.

```
PUT /user/:user
```

Input

Name	Type	Description
name	string	
email	string	
password	string	
status	string	
roles	set	set of roles
attributes	dict	dictionary of key-value pairs
text	string	
email_verified	boolean	

Example Request

```
$ curl -XPUT http://localhost:8080/user/0a35bfd8-1175-4cd2-96f6-eda9861fd15d \
-H 'Authorization: Key demo-key' \
-H 'Content-type: application/json' \
-d '{
      "password": "p8ssw0rd",
      "text": "test user",
      "email_verified": false
    }'
```

List all users

Returns a list of users.

```
GET /users
```

Example Request

```
$ curl http://localhost:8080/users \
-H 'Authorization: Key demo-key'
```

Example Response

```
200 OK
```

```
{
  "domains": [
    "*"
  ],
  "groups": [
    "*"
  ],
  "orgs": [
    "*"
  ],
  "status": "ok",
  "time": "2017-01-02T00:24:00.393Z",
  "total": 2,
  "users": [
    {
      "createTime": "2017-01-01T23:49:38.486Z",
      "email_verified": false,
      "id": "b91811e7-52dd-4a8f-adae-b4d5c628d6f8",
      "login": "jane.doe@example.org",
      "name": "Jane Doe",
      "provider": "basic",
      "role": "user",
      "text": "demo user"
    },
    {
      "createTime": "2017-01-02T00:23:24.487Z",
      "email_verified": true,
      "id": "166b41d6-849f-440d-ba30-1a5345d86fb6",
      "login": "joe.bloggs@example.com",
      "name": "Joe Bloggs",
      "provider": "basic",
      "role": "user",
      "text": "demo user"
    }
  ]
}
```


Delete a user

Permanently deletes a user. It cannot be undone.

```
DELETE /user/:user
```

Example Request

```
$ curl -XDELETE http://localhost:8080/user/166b41d6-849f-440d-ba30-1a5345d86fb6 \
-H 'Authorization: Key demo-key'
```

1.16.9 Permissions

Create permission

Creates a new permission lookup. Used to match user groups/roles to scopes.

```
POST /perm
```

Input

Name	Type	Description
scopes	string	
match	regex	

Example Request

```
$ curl -XPOST http://localhost:8080/perm \
-H 'Authorization: Key demo-key' \
-H 'Content-type: application/json' \
-d '{
      "scopes": ["read", "write", "admin:alerts"],
      "match": "alerta_ops"
    }'
```

Example Response

```
201 CREATED
```

```
{
  "id": "40c2daee-1d77-44d5-b62d-e3e446396cef",
  "permission": {
    "id": "40c2daee-1d77-44d5-b62d-e3e446396cef",
    "match": "alerta_ops",
    "scopes": [
      "read",
      "write",
      "admin:keys"
    ]
  },
  "status": "ok"
}
```

List all permissions

Returns a list of permissions.

```
GET /perms
```

Example Request

```
$ curl http://localhost:8080/perms \
-H 'Authorization: Key demo-key'
```

Example Response

```
200 OK
```


```
{
  "permissions": [
    {
      "id": "5b726183-019f-4add-b6dc-caba87e873f7",
      "match": "alerta_ro",
      "scopes": [
        "read"
      ]
    },
    {
      "id": "f4c91af3-5222-4201-9da0-02c40122f4c4",
      "match": "alerta_rw",
      "scopes": [
        "read",
        "write"
      ]
    },
    {
      "id": "1f84f919-c07a-4bd1-93b0-26e28871257f",
      "match": "alerta_ops",
      "scopes": [
        "read",
        "write",
        "admin:keys"
      ]
    }
  ],
  "status": "ok",
  "time": "2017-07-29T21:42:30.500Z",
  "total": 3
}
```


Delete a permission

Permanently delete a permission. It cannot be undone.

```
DELETE /perm/:perm
```

Example Request

```
$ curl -XDELETE http://localhost:8080/perm/1f84f919-c07a-4bd1-93b0-26e28871257f \
-H 'Authorization: Key demo-key'
```

1.16.10 Customers

Create a customer

Creates a new customer lookup. Used to match user logins to customers.

```
POST /customer
```

Input

Name	Type	Description
customer	string	
match	regex	

Example Request

```
$ curl -XPOST http://localhost:8080/customer \
-H 'Authorization: Key demo-key' \
-H 'Content-type: application/json' \
-d '{
      "customer": "Example Corp.",
      "match": "example.com"
    }'
```

Example Response

```
201 CREATED
```

```
{
  "customer": {
    "customer": "Example Corp.",
    "id": "289ca657-ea2c-4775-9e07-cc96844cc717",
    "match": "example.com"
  },
  "id": "289ca657-ea2c-4775-9e07-cc96844cc717",
  "status": "ok"
}
```

List all customers

Returns a list of customers.

```
GET /customers
```

Example Request

```
$ curl http://localhost:8080/customers \
-H 'Authorization: Key demo-key'
```

Example Response

```
200 OK
```


```
{
  "customers": [
    {
      "customer": "Example Corp.",
      "id": "289ca657-ea2c-4775-9e07-cc96844cc717",
      "match": "example.com"
    },
    {
      "customer": "Example Org.",
      "id": "90f4e211-c815-4112-9e1a-6e53de5a59c6",
      "match": "example.org"
    }
  ],
  "status": "ok",
  "time": "2017-01-02T01:21:38.958Z",
  "total": 2
}
```


Delete a customer

Permanently delete a customer. It cannot be undone.

```
DELETE /customer/:customer
```

Example Request

```
$ curl -XDELETE http://localhost:8080/customer/90f4e211-c815-4112-9e1a-6e53de5a59c6 \
-H 'Authorization: Key demo-key'
```

1.17 Alert Format

Alerts received and sent by Alerta conform to a common alert format. All components of alerta use this message format and any external systems must produce or consume messages in this format also.

1.17.1 Attributes

The following alert attributes are populated at source:

Attribute	Description
`resource`	resource under alarm, deliberately not host-centric
`event`	event name eg. `NodeDown`, `QUEUE:LENGTH:EXCEEDED`
`environment`	effected environment, used to namespace the resource
`severity`	severity of alert (default `normal`). see *Alert Severities* table
`correlate`	list of related event names
`status`	status of alert (default `open`). see *Alert Status* table
`service`	list of effected services
`group`	event group used to group events of similar type
`value`	event value eg. `100%`, `Down`, `PingFail`, `55ms`, `ORA-1664`
`text`	freeform text description
`tags`	set of tags in any format eg. `aTag`, `aDouble:Tag`, `a:Triple=Tag`
`attributes`	dictionary of key-value pairs
`origin`	name of monitoring component that generated the alert
`type`	alert type eg. snmptrapAlert, syslogAlert, gangliaAlert
`createTime`	UTC date-time the alert was generated in ISO8601 format
`timeout`	number of seconds before alert is considered stale
`rawData`	unprocessed data eg. full syslog message or SNMP trap

Note: Only `event` and `resource` are mandatory.

Attention: If the `reject` plugin is enabled (which it is by default) then alerts must have an `environment` attribute that is one of either `Production` or `Development` and it must define a `service` attribute. For more information on configuring or disabling this plugin see *Plugin Settings*.

1.17.2 Attributes added when processing alerts

Attribute	Description
`id`	globally unique random UUID
`duplicateCount`	a count of the number of times this event has been received for a resource
`repeat`	if duplicateCount is 0 or the alert status has changed then repeat is False, otherwise it is True
`previousSeverity`	the previous severity of the same event for this resource. if no event or `correlate` events exist in the database for this resource then it will be `unknown`
`trendIndication`	based on `severity` and `previousSeverity` will be one of `moreSevere`, `lessSevere` or `noChange`
`receiveTime`	UTC datetime the alert was received by the Alerta server daemon
`lastReceiveId`	the last alert `id` received for this event
`lastReceiveTime`	the last time this alert was received. only different to receiveTime if the alert is a duplicate
`customer`	assigned based on the owner of the API key used when submitting the alert, if "Customer Views" is enabled, or can be set if `admin` user
`history`	whenever an alert changes severity or status then a list of key alert attributes are appended to the history log

1.17.3 Alert Status

Status	Status Code
`open`	1
`assign`	2
`ack`	3
`closed`	4
`expired`	5
`blackout`	6
`shelved`	7
`unknown`	9

1.17.4 Alert Severities

The Alarms in Syslog **RFC 5674** was referenced when defining alert severities.

Severity	Severity Code	Colour
`security`	0	Black
`critical`	1	Red
`major`	2	Orange
`minor`	3	Yellow
`warning`	4	Blue
`informational`	5	Green
`debug`	6	Purple
`trace`	7	Grey
`indeterminate`	8	Silver
`cleared`	9	Green
`normal`	9	Green
`ok`	9	Green
`unknown`	10	Grey

1.17.5 History Entries

History log entries can be for either severity or status changes.

Attribute	Description
`id`	alert id that history log entry relates to
`event`	event name of alert changing severity or status
`severity` (*)	new severity of alert changing severity
`status` (+)	new status of alert changing status
`value` (*)	event value of alert changing severity
`text`	text describing reason for severity or status change
`type`	history type eg. `action`, `status`, `severity` or `value` change
`updateTime`	UTC date-time the alert triggering the change was created

Note: The `severity` and `value` attributes are only added to the history log for alerts with `event` changes (See * above). And the `status` attribute is only added to the history log for alerts with `status` changes (See + above).

1.18 Heartbeat Format

Heartbeats received by Alerta conform to the following format.

1.18.1 Attributes

The following heartbeat attributes are populated at source:

Attribute	Description
`id`	globally unique random UUID
`origin`	name of monitoring component that generated the heartbeat
`tags`	set of tags in any format eg. `aTag`, `aDouble:Tag`, `a:Triple=Tag`
`type`	heartbeat type. only `Heartbeat` is currently supported
`createTime`	UTC date and time the heartbeat was generated in ISO 8601 format
`timeout`	number of seconds before heartbeat is considered stale

Note: Only `origin` is mandatory.

1.18.2 Attributes added when processing heartbeats

Attribute	Description
`receiveTime`	UTC date and time the heartbeat was received by the Alerta server daemon
`customer`	assigned based on the owner of the API key used when submitting the heartbeat, if "Customer Views" are enabled

1.18.3 Example

```
{
  "origin": "macbook",
  "tags": [
    "foo",
    "bar",
    "baz"
  ],
  "createTime": "2015-10-03T00:00:59.055Z",
  "href": "http://api.alerta.io/heartbeat/a8b97056-8415-4b4f-83c8-e84ffcc676a3",
  "timeout": 300,
  "receiveTime": "2015-10-03T00:00:59.681Z",
  "type": "Heartbeat",
  "id": "a8b97056-8415-4b4f-83c8-e84ffcc676a3"
}
```

CHAPTER 2

Contribute

- Core project: https://github.com/alerta/alerta
- Web UI: https://github.com/alerta/angular-alerta-webui
- Python SDK: https://github.com/alerta/python-alerta-client
- Contributions and integrations: https://github.com/alerta/alerta-contrib

CHAPTER 3

Support

- Gitter chat room: https://gitter.im/alerta/chat
- *Frequently Asked Questions*
- Issue Tracker: https://github.com/alerta/alerta/issues

3.1 Frequently Asked Questions

3.1.1 Alerta

Why can't I see any alerts in the web browser?

If you can send and query for alerts using the `alerta` CLI tool this problem is almost certainly related to cross-origin browser errors. Open up the Javascript developer console in your browser of choice and look for CORS errors like:

```
XMLHttpRequest cannot load http://api.alerta.io/alerts?status=open.
No 'Access-Control-Allow-Origin' header is present on the requested
resource. Origin 'http://web.alerta.io' is therefore not allowed access.
```

To fix this you can either serve the web UI from the same origin as the API using a web server to *reverse proxy* the web UI or ensure that the API server allows the origin where the web UI is hosted by adding it to the `CORS_ORIGINS` *server configuration* setting.

Why do I need to set an `environment` and `service` when they are not mandatory?

Only `resource` and `event` are technically required to ensure that Alerta can process alerts correctly. However, the "out-of-the-box" default server setting for `PLUGINS` has the `reject` plugin enabled. This plugin enforces an alert "policy" of requiring an `environment` attribute of either `Production` or `Development` and a value for the `service` attribute.

This is to encourage good habits early in defining useful alert attributes that can be used to "namespace" alerts (this is what the `environment` attribute is for) and so that the web console can organise by `environemnt` and filter alerts by `service`.

However, one of the principles of Alerta is not to enforce its view of the world on users so the plugin can be *easily configured*, modified or completely disabled. It's up to you.

Can I define custom severity codes and levels?

Yes, you can now completely change the severity names, severity levels and colours. See *Alerta Web UI* for more information.

How can I add a priority to an alert eg. High, Medium, Low?

Use a custom attribute called `priority` when sending alerts to Alerta:

```
$ alerta send ... --attributes priority=high ...
```

Alerts of differing priority could be queried by `alerta` command-line tool using:

```
$ alerta query --filters attributes.priority=high
```

Using the web console to sort alerts by priority as well as severity would require some development effort.

What's the difference between *ack*, *close* and *delete*?

Alerts are meant to auto-close when a corresponding *normal* or *cleared* alert is received for that event-resource combination. If no *normal* alert exists for a particular event (which may be the case for alerts from log files or SNMP traps, for example) then the alert will be deleted when the timeout period has expired. Alerts timeout after 1 day by default but that is configurable on a per-alert basis.

If, as an operator, you want to remove an event from view then you can either *ack* the alert or DELETE it. If the alert is DELETED a new alert with the same event-resource combination will trigger a new notification email (if configured) whereas an *ack*'ed alert will not.

Why don't you have a plugin or integration for X, where X is whatever you use in your job?

We could spend countless hours writing plugins for everything and never finish or we could focus on building an easily extensible system with great documentation and let the end-user build the plugins they need. Having said that, we have still created many plugins and integrations as working examples and we are not against writing more if there is popular demand. We are also happy to accept submissions.

What's this MongoDB "ServerSelectionTimeoutError"?

With the update to PyMongo 3.x multiprocessing applications "parent process and each child process must create their own instances of MongoClient".

For Apache WSGI applications, an example Apache "vhost" configuration for the Alerta API would look like this:

```
<VirtualHost *:8080>
  ServerName localhost
  WSGIDaemonProcess alerta processes=5 threads=5
  WSGIProcessGroup alerta
  WSGIApplicationGroup %{GLOBAL}
  WSGIScriptAlias / /var/www/api.wsgi
  WSGIPassAuthorization On
  <Directory /opt/alerta>
    Require all granted
  </Directory>
</VirtualHost>
```

Full examples are available on GitHub and more information on why this is necessary is available on stackoverflow and the PyMongo where they discussion PyMongo in relation to forking and mod_wsgi site.

Does Alerta support Python 2.7 or Python 3?

Alerta *Release 5.2* is the last version to support Python 2.7 and from 31 August, 2018 it will only receive crtical bug fixes and security patches.

Alerta *Release 6* supports Python 3.6+ and is recommended for new production environments and existing installations should be switched to Python 3 well before 1 January, 2020 when Python 2.7 becomes End-of-Life.

CHAPTER 4

License

This project is licensed under the Apache license, Version 2.0 .

4.1 Releases

4.1.1 Roadmap

- Web UI redesign using Google material design
- Custom alert filters and dashboard views
- Use alarm model based on ISA 18.2 / IEC 62682

4.1.2 Release History

Release 6.0.0 (01-09-2018)

- First release to support Python 3 only
- Add static type checking to build pipeline and start type annotations
- Add audit info for blackouts including user and reason
- Support every combination of alert attribute for blackouts

Release 5.2.0 (25-04-2018)

- First release to support Python 3.6+ only
- Final release to support Python 2.7
- LDAP authentication support for BasicAuth logins

- Change "status" endpoints to "action" endpoints
- Allow admin to override customer assigned to an alert

Release 5.1.0 (08-04-2018)

- alarm shelving for temporarily removing alerts from the main alert list
- new blackout status that don't trigger plugins to keep track of suppressed alerts
- add history entry for de-duplicated alerts with a value change
- multiple customers for auth providers that allow membership of more than one group
- Python 3 support only (no breaking changes for Python 2, yet)

Release 5.0.0 (07-10-2017)

- Support for PostgreSQL (including Amazon RDS and Google Cloud SQL)
- API responses are Gzipped to make everything faster
- Development command line has changed from *alertad* to *alertad run*
- Major code refactor with flatter structure (beware imports! see next)
- WSGI import has changed from *from alerta.app import app* to simply *from alerta import app*
- Plugins import has changed from *from alerta.app import app* to *from alerta.plugins import app*
- Blackout is now a plugin so it can be disabled and replaced with a custom blackout handler
- Switched to using wheels for distribution via PyPI See http://pythonwheels.com/
- Alerta API now supports multiple roles for BasicAuth (though not supported in the web UI yet)
- Alert format: *value* is now always cast to a string.
- Added */management/housekeeping* URL to replace *housekeepingAlerts.js* cron job script
- *DATABASE_URL* connection URI setting replaces every other MongoDB setting with a non-mongo specific variable

Release 4.10 (27-07-2017)

- Scope-based permissions model based on RBAC
- SAML2 authentication user logins
- Prometheus webhook updated to support version 4
- Plugin result chaining for tags and attributes

Release 4.9 (16-03-2017)

- LDAP authentication via Keycloak support
- MongoDB SSL connection support
- Pingdom webhook changed to use new "State change" webhook

Release 4.8 (05-09-2016)

- Use GitHub Enterprise for OAuth2 login
- Riemann webhook integration
- Telegram webhook and related plugin for bi-directional integration
- Grafana webhook integration
- Switch to MongoDB URI connection string format
- Added simple *good-to-go* health check
- Added "flap detection" utility method for use in plugins
- Fix oEmbed API endpoint
- Default severity changed from "unknown" to "indeterminate"
- Add routing rules for plugins

Release 4.7 (24-01-2016)

- Prometheus webhook integration
- Google Stackdriver webhook integration
- Configurable severities
- Blackout periods by customer
- Status change hook for plugins
- Require authentication on webhooks if auth enabled
- Limit alert history in MongoDB
- Send email confirmation for Basic Auth sign-ups
- Removed support for Twitter OAuth1

Release 4.6 (26-11-2015)

- Customer views for multitenancy support
- Authorisation using *Admin* and *User* roles

Release 4.5 (9-9-2015)

- Added ability to blackout alerts for defined periods
- Use GitLab for OAuth2 login
- Python 3 support (both `alerta` client and WSGI server)

Release 4.4 (11-6-2015)

- MongoDB version 3 support

Release 4.3 (12-5-2015)

- Support Basic Auth for user logins

Release 4.2 (13-3-2015)

- PagerDuty webhook integration
- API keys can be *read-only* as well as *read-write*

Release 4.1 (25-2-2015)

- Twitter OAuth login
- API response pagination

Release 4.0 (15-1-2015)

- Change web browser authentication to use JWT tokens
- Improve Google OAuth login and add GitHub OAuth

Release 3.3 (16-12-2014)

- Add Amazon AWS CloudWatch, Pingdom web hook integration
- Slack and HipChat plugins

Release 3.2 (11-10-2014)

- Major refactor and simplification of server architecture
- Add Google OAuth user logins
- API keys for controlling programatic access
- Add support for server-side custom plugins eg. Logstash, AWS SNS, AMQP
- Deprecated RabbitMQ as a dependency

Release 3.1 (9-5-2014)

- Extend API to support new dashboard
- Stability and performance enhancements

Release 3.0 (25-3-2014)

- Deploy server and dashboard as Python WSGI apps
- Add AWS Cloudwatch, PagerDuty and Solarwinds integrations
- Pinger module for host availablity checks
- Start development of version 3 console based on AngularJS

Release 2.0 (11-3-2013)

- Major refactoring into python modules and classes
- API rewrite based on Flask microframework
- Dashboard rewritten using Flask server-side templates
- Integrations for AWS SNS, Syslog, Dynect and URL monitoring

Release 1.0 (27-3-2012)

- CGI script receives alerts and pushes to ActiveMQ message bus
- Background daemon reads message bus, processes and stores to MongoDB
- HTML/JavaScript console displays alerts on web dashboard
- Integrations for AWS EC2, Ganglia, IRC, Kibana, Email and SNMP

4.2 About

Alerta started at The Guardian out of necessity as a replacement for a legacy monitoring tool but only after exhaustively evaluating all credible alternatives first.

Initially all we wanted was to be able to create alert thresholds against the hundreds of thousands of Ganglia metrics collected for the website and view the alerts in a web console ie. a Ganglia "alerter". Not having a proper name for this metrics and monitoring system the working name of "an alerter" stuck and a simple homophone was chosen to aid future Google searches.

In the end, the thresholding of metrics proved very difficult to scale so we eventually split the project in two and metric thresholding was given to Riemann (see riemann-config) and the alert correlation, de-duplication and visualisation became the "Alerta" project.

Over the years the project has evolved to meet the constantly changing needs of the Guardian developer teams as they moved to a more agile, dynamic, "swimlaned" architecture which has meant, for the operations team, a shift from static, self-hosted infrastructureto an internal OpenStack cloud to then finally an external cloud service.

In that time certain key features of Alerta have been deprecated as requirements changed (eg. the message bus, Ganglia, Riemann) and others added (eg. OAuth2 login, CloudWatch, Pingdom, PagerDuty integration). In the process it has been slimmed down to fewer core components making it easier to understand, deploy and manage.

As such, Alerta is now quite different to the somewhat "over engineered" initial solution but the core concepts of being a flexible, easy-to-use tool remain and it is now a "cloud-ready" solution adapted to the challenges of a fast changing environment.

CHAPTER 5

Indices and tables

- genindex
- search

Index

K

L

M

O

P

R

S