Moira

Release 1.0

September 23, 2016

Contents

CHAPTER 1

Contents

1.1 Overview

Moira is a real-time alerting tool, based on Graphite data.

1.1.1 Key Features

- **Graphite storage independence**

  Some Graphite queries are *very* ineffective. Tools like Seyren multiply this effect by making lots of ineffective queries every minute, overloading your cluster. Moira relies on the incoming metric stream, and has its own fast cache for recent data.

- **Support for all Graphite functions**

  Graphite function library is embedded directly into the source code of Moira. You can use any function and get predictable results, like in your Graphite dashboards.

- **Support for custom Python expressions**

  If simple warning/error threshold is not enough, you can write flexible Python expressions to calculate trigger state based on metric data.

- **Tags for triggers and subscriptions**

  When several teams/services share one monitoring tool, it is essential to provide some way of filtering triggers and subscriptions in the UI. Moira has a flexible tag system.

- **Extendable notification channels**

  Moira has support for email, Slack and Pushover notifications out-of-the-box. But you can always write your own plugin in Go and rebuild Moira Notifier microservice.

- **Alarm fatigue protection**

  Sometimes one of your triggers goes mad and switches back and forth between states, sending you hundreds of notifications. Sometimes you just ignore and delete all messages, accidentally also deleting one that is actually important. Moira tries to protect you with a feature called *throttling*. It's simple: if one of your triggers starts to send more than 10 messages over an hour, Moira limits this trigger to one message per 30 minutes. Alerts from this trigger are combined, and not lost - just packaged into a single message.

1.1.2 Limitations

By default, Moira stores metric history for one hour. This ensures performance under heavy load. You can tweak this in config file, but note that performance will degrade.

In order to reduce database load, Moira checks every single trigger at most once every 5 seconds. Probably, your metrics arrive once every minute, so you really won't notice this limitation. You can also tweak this in config file.

1.1.3 Microservices

In spirit of Graphite architecture, Moira consists of several loosely coupled microservices. You are welcome to replace or to add new ones.

Cache

Cache is a lightweight Go application responsible for receiving lots of metric data in Graphite format. Cache filters received data and saves only metrics that match any of user triggers. This reduces load on all other parts of Moira.

Checker

Checker is a Python application with embedded Graphite functions. Checker watches for incoming metric values and performs checks according to saved trigger settings. When state of any trigger changes, Checker generates an event.

Notifier

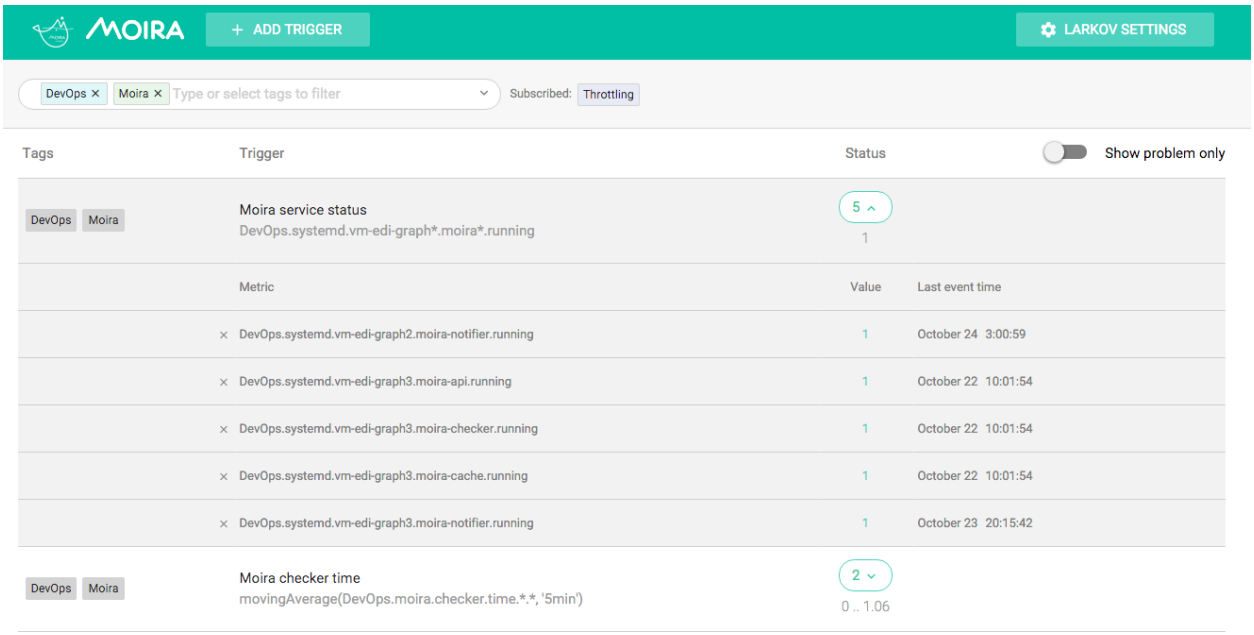
Notifier is a Go application that watches for generated events. Notifier is responsible for scheduling and sending notifications, observing quiet hours, retrying failed notifications, etc.

API

API is a Python application that serves as a backend for UI.

UI

UI is a frontend Angular application, it looks like this:



Database

All services communicate only through a Redis database, without any additional protocols or connections between each other.

1.2 Installation

1.2.1 Manual Installation

There are following components you need to install before running Moira microservices:

1. golang version 1.5 or higher
2. redis database version 2.8 or higher
3. python version 2.7
4. web server e.g. nginx

Install Moira Microservices

Cache and Notifier

```
git clone https://github.com/moira-alert/cache.git
cd cache
sudo make
```

```
git clone https://github.com/moira-alert/notifier.git
cd notifier
sudo make
```

Worker

```
git clone https://github.com/moira-alert/worker.git
cd worker
sudo make prepare
make pip
sudo pip install dist/moira_worker-*.tar.gz
```

Download Web UI Application

https://github.com/moira-alert/web/releases/latest

Configure

1. Place configuration file to the default location, `/etc/moira/config.yml`

You can dive into Configuration syntax on a separate page.

2. Place nginx configuration file to `/etc/nginx/conf.d/moira.conf`

```
server {
    listen 127.0.0.1:80;
    location / {
        root /var/local/www/moira;
        index index.html;
    }
    location /api/ {
        proxy_pass http://127.0.0.1:8081;
    }
}
```

3. Place UI config.json file to `/var/local/www/moira/config.json`

Run

1. Run nginx and redis-server
2. Run cache

```
$GOPATH/bin/cache --config=/etc/moira/config.yml
```

3. Run notifier

```
$GOPATH/bin/notifier --config=/etc/moira/config.yml
```

4. Run API

```
moira-api
```

5. Run checker

```
moira-checker
```

Now you need to feed your metrics to Moira (see Feeding Metrics to Moira) on port 2003 and to create alerts in UI (see User Guide).

1.2.2 Vagrant Box

Warning: Vagrant is a rapid development and testing tool. Do not use this Vagrant box in production.

Along with Docker Containers, Vagrant box is a quick way to evaluate Moira or setup a development environment.

Quickstart

```
git clone https://github.com/moira-alert/moira-vagrant-dev
vagrant up

start http://localhost:8888
```

Contributing

See README.

1.2.3 Docker Containers

Warning: This is a third-party installation container provided by community. Use at your own risk.

You can quickly test a local moira installationg using Docker containers provided by Toby Jackson.

```
git clone https://github.com/warmfusion/moira-docker.git
cd moira-docker
docker-compose up
```

Tip: If using Docker-Machine you can navigate to the Moira interface on `http://$(docker-machine ip dev):8080/`

This should setup a small cluster of Docker containers ready to accept data and handle notifications, with the interface on `http://localhost:8080` and accepts graphite metrics on `localhost:2003`

See Tobys documentation for more information on submitting data to your new containers, and the User Guide for everything else.

1.2.4 RPM and DEB Packages

All stable versions of Moira components are tagged on GitHub. For every tag, we automatically build RPM and DEB packages. You can download these packages on *releases* page of every repository:

1. Web
2. Notifier
3. Worker (RPM and DEB packages are not ready yet, but you can install with *pip*)

```
    wget https://github.com/moira-alert/worker/releases/download/v1.1.14/moira_worker-1.1.14.tar.gz
    sudo pip install moira_worker-1.1.14.tar.gz
```

4. Cache

We don't maintain strict version dependencies between packages (sorry), but you should be okay if you install/upgrade all of the packages at once.

1.2.5 Configuration

All Moira microservices can use a single configuration file in YAML format.

By default, microservices will look for `/etc/moira/config.yml`, but you can change this location in systemd configuration files (pass your path as a command-line parameter) and in `moira-alert/worker/moira/config.py`

```
redis:
  host: localhost
  port: 6379
  dbid: 0 # redis database index http://redis.io/commands/select

# Base log directory for Moira API and Moira Checker microservices
worker:
  log_dir: /var/log/moira/worker
  log_level: info

api:
  port: 8081
  listen: '127.0.0.1'

# Moira microservices will send their own metrics to this address
graphite:
  uri: graphite-relay:2003
  prefix: DevOps.moira
  interval: 30

checker:
  # every trigger will be periodically checked even if no metrics arrive
  nodata_check_interval: 60 # (in seconds)
  # every trigger will be checked at most once every interval
  check_interval: 5 # (in seconds)
  # metrics older than this age will be purged after use
  metrics_ttl: 3600 # (in seconds)
  # if master is not receiving any metrics it stops nodata checks after that interval
  stop_checking_interval: 30 # (in seconds)

# Moira frontend uri, used to make links in notification templates
front:
  uri: http://localhost

notifier:
  log_file: /var/log/moira/notifier/notifier.log
  log_level: debug
  log_color: true
  sender_timeout: 10s0ms
  resending_timeout: 24:00 # 24 hours
  senders:
    - type: mail
      smtp_host: smtp.gmail.com
      smtp_port: 587
      smtp_pass: # no auth if empty
```

```
      smtp_user: # be used mail_from value if empty
      mail_from: moira@gmail.com
      insecure_tls: false
    - type: slack
      api_token: # place your token here
    - type: pushover
      api_token: # place your token here
    - type: script
      name: MyCustomScript
      exec: /bin/bash /opt/myscripts/gsm-modem.sh
    - type: twilio sms
      api_asid: # place your api sid
      api_authtoken: # place your api secret
      api_fromphone: # place phone
    - type: twilio voice
      api_asid: # place your api sid
      api_authtoken: # place your api secret
      api_fromphone: # place phone
      voiceurl: # place your voice url


cache:
  log_file: /var/log/moira/cache/cache.log
  listen: ':2003' # listen on all interfaces
  retention-config: /etc/moira/cache/storage-schemas.conf
  pid: /var/run/moira-cache.pid
```

storage-schemas.conf is graphite carbon configuration file.

Example:

```
[default_1min_for_1day]
pattern = .*
retentions = 1m:1d
```

1.2.6 Feeding Metrics to Moira

Moira needs to keep its own local copy of your metric data to improve performance and reduce load on your existing graphite cluster. This means data needs to be duplicated from your existing stream and sent to your existing cluster and to your moira installation.

Unfortunatly, the Carbon-Relay included with Graphite does not support duplication of data to multiple backends, and so you need to use a more feature rich carbon relay such as carbon-c-relay.

The following shows a basic example configuration which defines two clusters and sends all metrics to both at once. One cluster is the Moira installation, and the other uses consistent hashing across a three node cluster of Carbon servers.

```
cluster moira
forward
    moira-host:2003
;

cluster graphite
 carbon_ch
     127.0.0.1:2006=a
     127.0.0.1:2007=b
     127.0.0.1:2008=c
```

```
 ;

match *
    send to
        moira
        graphite
;
```

Note: replace **moira-host** with your host name

1.2.7 Security

Typically, internal DevOps tools like Graphite are deployed in intranet without any external access, so you can skip authentication and leave everything accessible to everyone. But powerful Moira features, like separate subscriptions for tags, work best when you have some way to tell apart users.

Moira doesn't provide any authentication mechanism. It is hard to find one that fits all situations. Instead, Moira accepts `X-WebAuth-User` header with some user id, like login name. You are free to set up any reverse proxy and configure it to provide this header.

If you don't, Moira will assume that user id is "anonymous".

Warning: Even if you do provide authentication header, please note that most parts of Moira are read and write accessible to every user, and there is no meaningful way of authorization in Moira. This is by design, because Moira is an internal DevOps tool. Separating users is a convenience, not protection feature.

Example of Nginx configuration

Assuming that Moira UI static files are in `/var/www/moira-web` and API is running on port 8081

```
server {
  auth_basic "Moira";
  auth_basic_user_file /etc/nginx/htpasswd;

  listen 0.0.0.0:80 default_server;

  location / {
    root /var/www/moira-web;
    index index.html;
  }

  location /api/ {
    proxy_pass http://127.0.0.1:8081;
    proxy_set_header X-WebAuth-User $remote_user;
  }
}
```

Look at auth_basic_module if you need more details of Nginx basic authentication.

1.3 User Guide

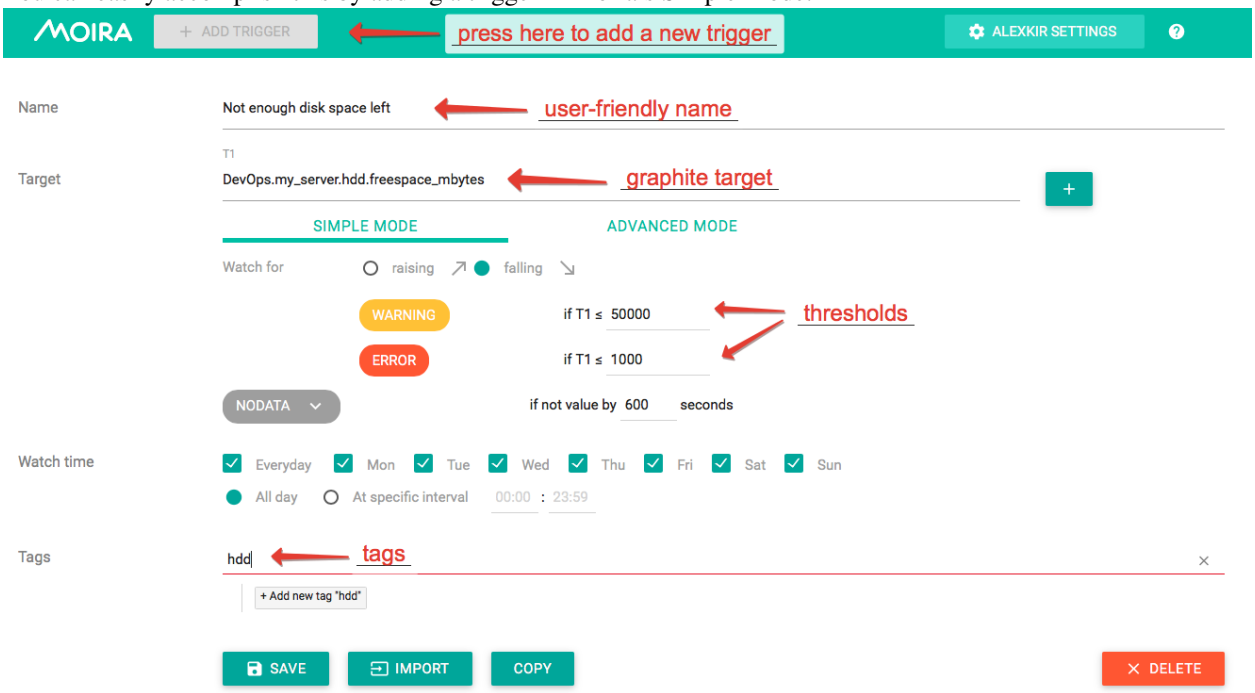
This user guide is based on a number of real-life scenarios, from simple and universal to complicated and specific. Also, we have screenshots.

Note: Click on any screenshot to see full-size version.

1.3.1 Simple Threshold Trigger

Let's say you measure how much free space is left on your HDD and store this value as `DevOps.my_server.hdd.freespace_mbytes` in Graphite. Maybe you want to get an email when you have less than 50 GB left (it's not a big problem), and a Pushover notification when you have less than 1 GB left (you really need to delete something asap).

You can easily accomplish this by adding a trigger in Moira's Simple Mode:



Graphite Target

You can specify a single metric like we did here: `DevOps.my_server.hdd.freespace_mbytes`.

You can also specify multiple metrics like `DevOps.*.hdd.freespace_mbytes`. All metrics will be monitored separately, and you will get separate notifications for each metric.

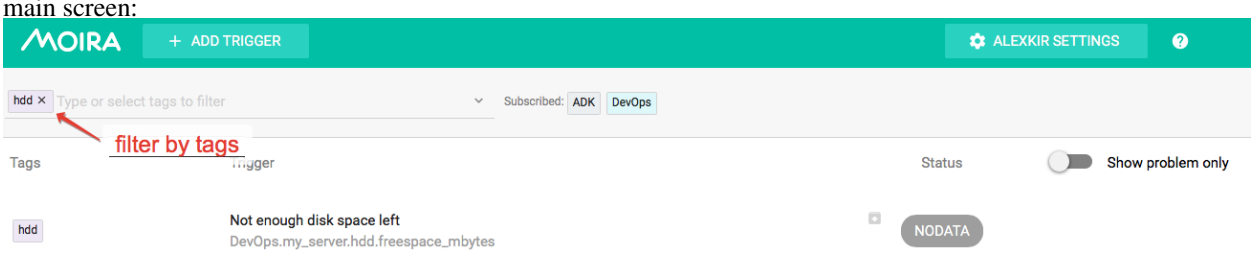
You can even use Graphite functions like `movingAverage(DevOps.my_server.hdd.freespace_mbytes, 10)`. Moira understands everything that Graphite itself understands. See appropriate documentation.

Thresholds

In simple mode you need to specify two threshold values: WARNING and ERROR. Note that Moira will automatically switch to raising or falling mode according to these settings. In our example, warning threshold is greater than error threshold. In this case, Moira will consider any value less than 50000 a warning and less than 1000 an error, which is what we want. In other cases, you may need to consider large values a problem - then you should make error threshold greater than warning.

Tags

In Moira, you cannot subscribe to a single trigger. Instead, you should categorize your triggers by tags and subscribe to a tag. It may look like an overkill here, but when you have dozens of triggers, you are much better off with tags, because you don't have to enter your contact information over and over again. Tags also help to filter information on main screen:



You can add as many tags as you want.

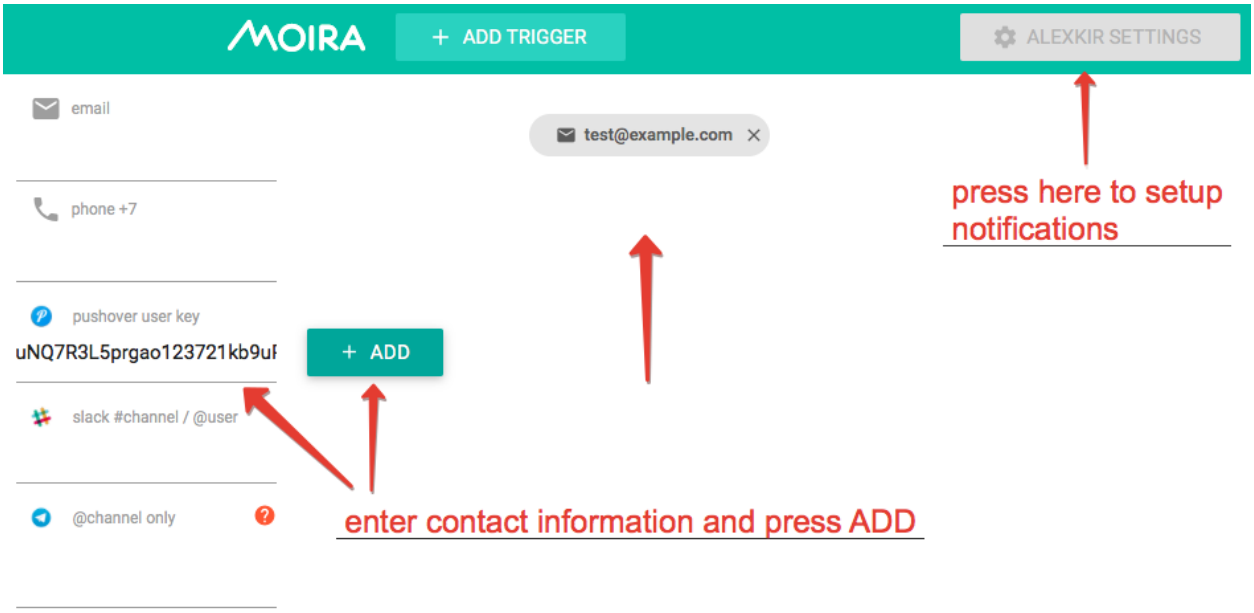
Subscriptions

Proceed to the Setting Up Your Subscriptions page to learn how to set up a subscription to your trigger.

1.3.2 Setting Up Your Subscriptions

By now you should have at least one trigger saved. If you don't, go back to the Simple Threshold Trigger page.

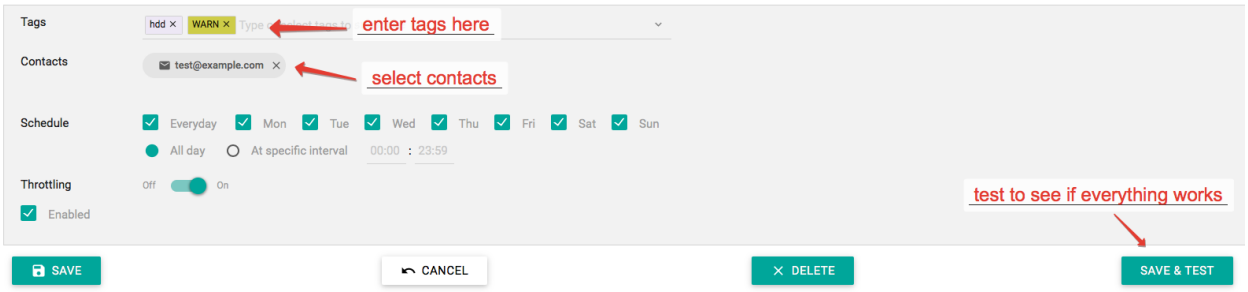
First, add some contacts:



If your Moira installation is configured with separate user accounts, you will see only your contacts and subscriptions on this page. Otherwise, every user will see the same "Anonymous settings" page with the same contacts and subscriptions.

Consult Security page for instructions on separating user accounts.

Once you have at least one contact, you can create a subscription. Press +NOTIFICATION button:



Tags

You will receive notifications from triggers with these tags. Note that a trigger must contain *all* of the subscription tags to match. For example, if you create a subscription for `tag1`, `tag2`, you will receive notifications from trigger with `tag1`, `tag2`, `tag3`, but not from trigger with `tag1` only.

`WARN` and `ERROR` are special pseudo-tags that match only events with corresponding level. For example,

you may wish to subscribe your Pushover account only to ERROR-level notifications.

`DEGRADATION` is a special pseudo-tag that match any event, where trigger's state degraded, e.g:

`OK` → `WARN`

`WARN` → `ERROR`

OK → NODATA

In addition to DEGRADATION a special HIGH DEGRADATION pseudo-tag will be added for the following events:

OK → ERROR
OK → NODATA
WARN → NODATA
ERROR → NODATA

Contacts

Select contacts for your notification by clicking on them. You can add as many contacts to one notification as you want.

Save & Test

You can just save your notification, but if you want to be 100% sure it works, you should immediately test it. Dummy notification message will arrive shortly.

1.3.3 Efficient Triggers

To use Moira efficiently, you should understand its underlying design decisions.

We often notice that when new users create their first triggers, they set thresholds at random, or by intuition. This is because when you configure your first 24/7/365 automated monitoring system, you don't really know how your system works. If you have at least hundreds of metrics, it's impossible to watch all of them with your eyes. What are the limits of your system? How often does your system reach critical resource consumption during a day? Should you immediately react when metric X reaches value N, or is it a fluctuation that passes by itself?

With time, when you learn to understand you system, you will need to tune your triggers. And that's when you need to understand Moira.

States

Unlike many other tools, that provide several distinct level systems like "priority" and "severity", Moira supports a single set of states. Every state has a well-defined meaning, and you should use these states accordingly.

OK

This is a basic state, in which all your metrics must spend most of their time. Just like you keep your autotests green, you should keep your metrics green.

WARN

This state means that you should do something to prevent having ERRORs in the future. Not immediately: maybe you should order more hardware from your vendor, or plan to optimize code in the next iteration. You can configure less intrusive notification channels here, like email.

Metrics can be in this state for days or even weeks.

ERROR

This is a critical condition that requires immediate intervention. Your datacenter is on fire. All application processes have shut down. There is no disk space left on your database server to process million-dollar transactions. These notifications are important enough to wake you up at night. You can still configure schedules to assign shifts to several engineers, though (see Schedules). You should configure more intrusive notification channels here, like Pushover.

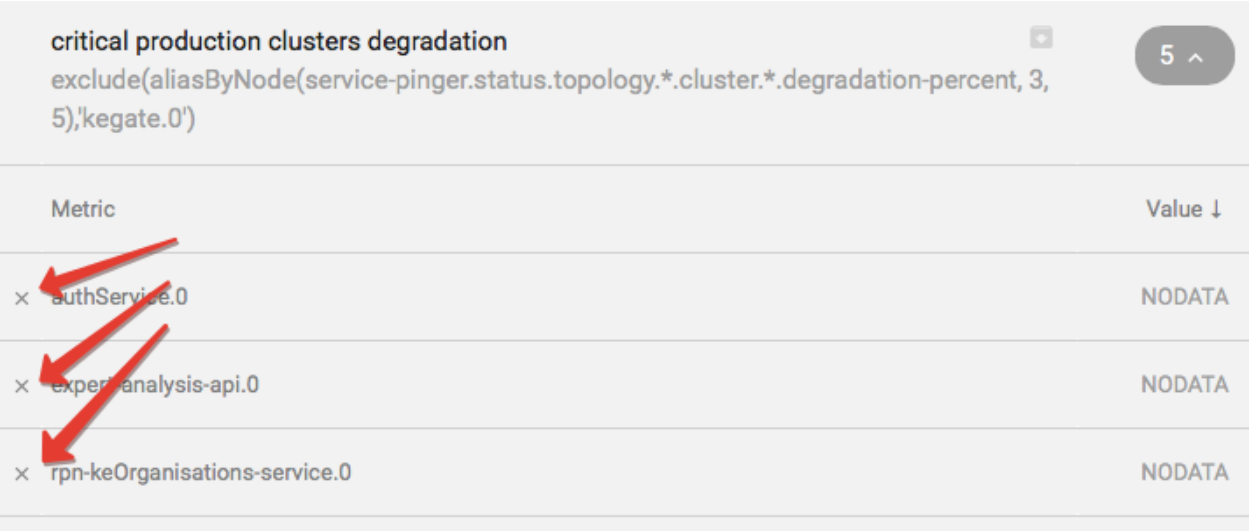
Metrics should not be in this state for more than several hours.

Moira will send you reminders every 24 hours if some of your metrics remain in this state.

If a notification channel supports high-priority messages (like Pushover does), Moira will try to use them for ERRORs.

NODATA

This state means that Moira hasn't been receiving data points for a metric for some time. See Dealing with NODATA for details. This state is considered as bad as an ERROR in Moira (because it can actually be an ERROR - we don't receive any data, so we don't know for sure). It may be even worse than an ERROR, because users tend to ignore metrics in this state and leave them hanging in the web interface, greatly increasing the chance to miss something actually important. You should delete old unused metrics from Moira when they stop providing data points:



Every metric is in this state at first. You will receive one NODATA -> OK notification when the first data point arrives.

Moira will send you reminders every 24 hours if some of your metrics remain in this state.

Moira will set NODATA state only for known metrics - i.e. for metrics that have sent at least one data point to Moira.

EXCEPTION

This is an error inside Moira. Unless you have bad syntax in your Advanced Mode Trigger trigger, this has nothing to do with your metric state. You should try to fix or update Moira, or contact Moira developers (see Contact Moira Developers).

Dealing With False Positives

Sometimes it's hard to maintain strict rule of keeping your metrics green, if your triggers switch OK->ERROR->OK->ERROR for short amounts of time several times a day. It can lead to alarm fatigue and missing actual failures.

There is no single recipe for eliminating false positives, but here are some tips.

Use Graphite Functions

Graphite provides tons of useful functions to process data, and Moira understands all of them. For example:

- If you are experiencing peaks on you graphs that lead to unnecessary state switches, you can alleviate these peaks with `movingAverage` or `movingMedian`.

- If you are interested in aggregate 10-minute values, not single minute values, use `summarize`.
- If you want zeroes instead of missing data points, use `transformNull`. Also, `keepLastValue` is useful when dealing with missing points.
- Avoid functions that show and hide metrics, like `averageAbove`. Moira does not consider hidden metrics to be in NODATA state. Instead, Moira retains last state that the metric had when it was visible.

Draw First, Monitor Later

Always draw a graph of target(s) you are planning to monitor. Use generic Graphite web interface or something like Grafana. Look for minumum and maximum values. Notice, how often and for how long the graph crosses your planned thresholds. Try to correlate the graph with previous system failures. Then, copy and paste corrected target to Moira.

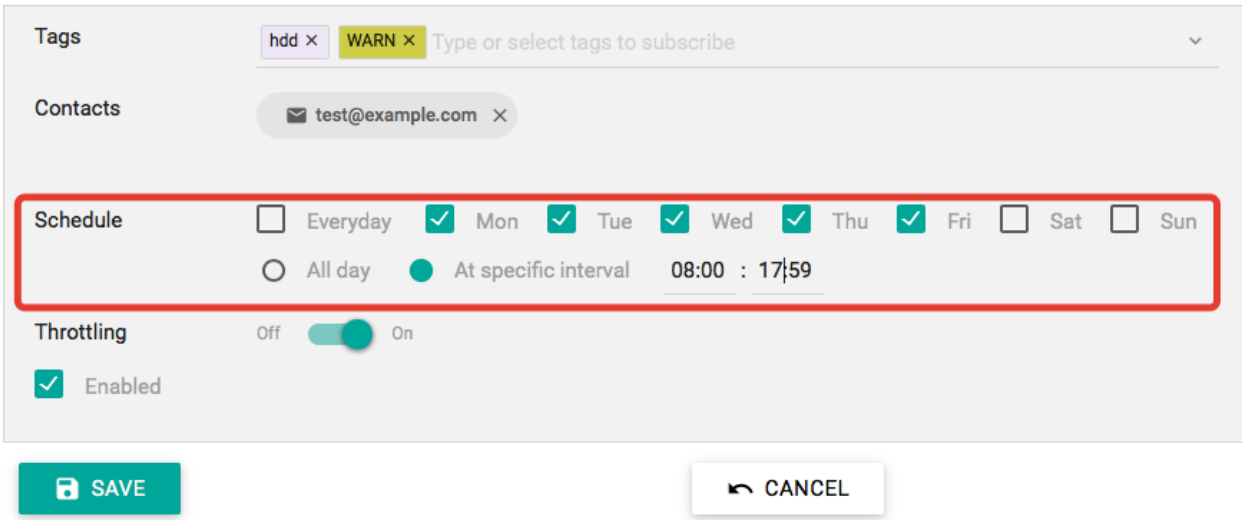
Of course, you can and should remove any functions that make no sense in Moira (like `sortByTotal`) and can generate unwanted side effects (like `averageAbove`).

1.3.4 Schedules

Moira provides two ways of defining allowed time intervals for notifications.

Subscription Schedule

If a metric is not that important to wake you up in the middle of the night, you can set a schedule for subscription:



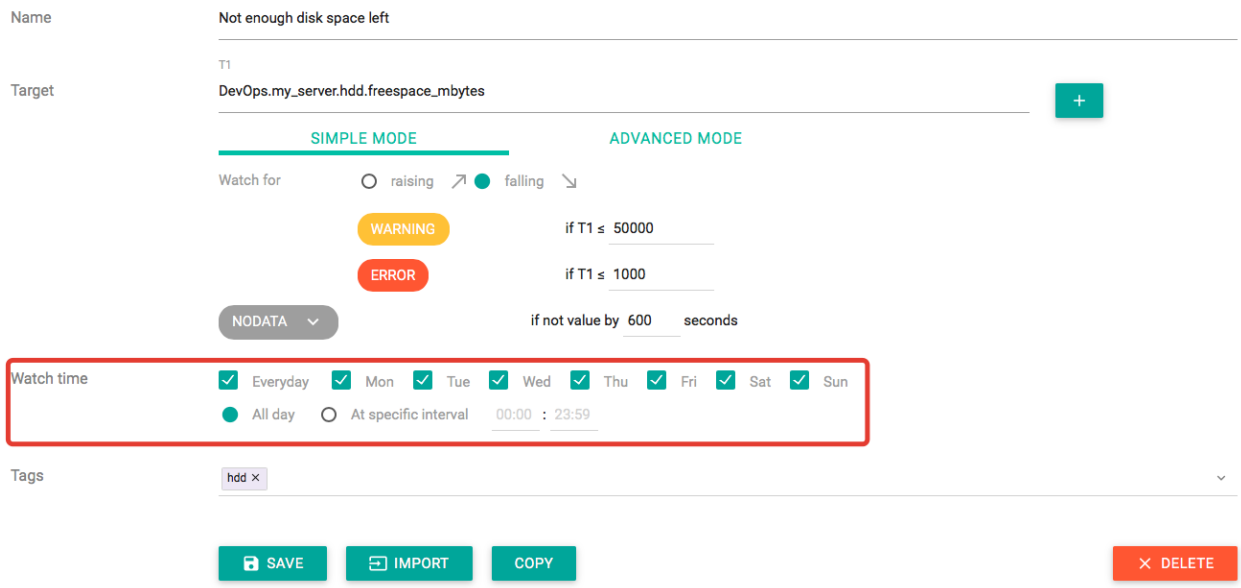
Notifications generated by this subscription will arrive only on weekdays, from 08:00 to 17:59 local time.

If an event happens on weekend, you will receive a notification at 08:00 on Monday. So notifications are not skipped, you just receive them later. Events will still appear on the event history page at the time when they happened (see Current State, Total State and Event History).

Trigger Watch Time

Sometimes you want to monitor a metric only part of the time. Let's say, you have a popular website, that serves over 1000 page views per second during the day. You can set up a trigger to notify you when you have less than 50 page views per second - obviously, something is wrong. You also need to disable this trigger for the night, because in the night all of your users are sleeping, and this metric is irrelevant.

Of course, you can set up a subscription schedule - but your history will become riddled with false night "events", and you will still receive notifications in the morning. In this case, you need to set up a trigger watch time:



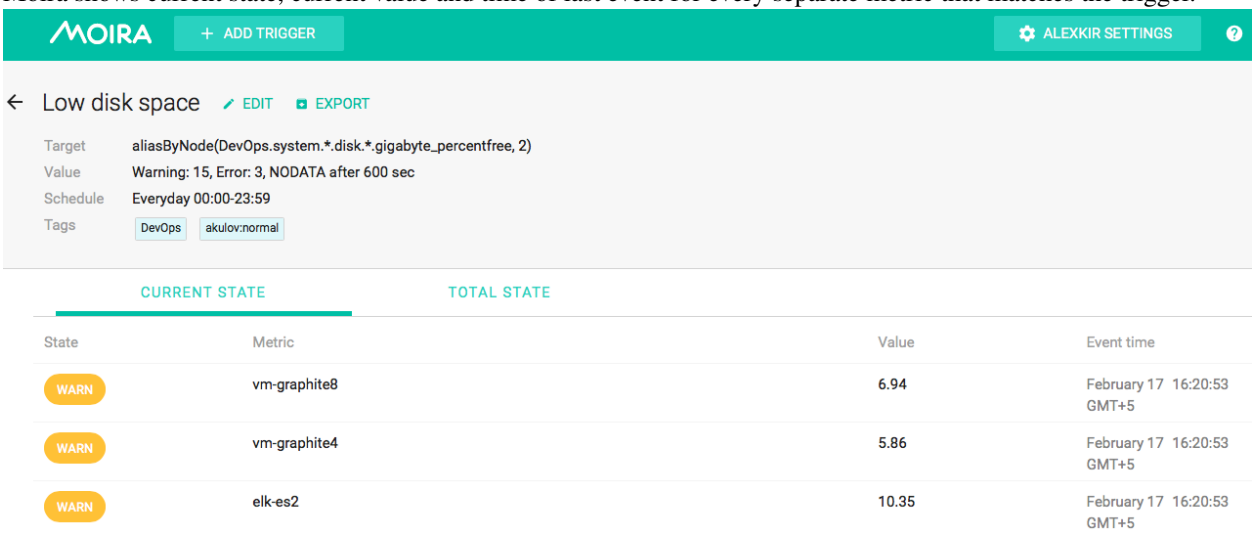
No events will be recorded for this trigger outside of watch time - you will receive no notifications, and the event history page will be empty (see Current State, Total State and Event History).

1.3.5 Current State, Total State and Event History

By clicking on a saved trigger, you can see current state, total state and event history of this trigger.

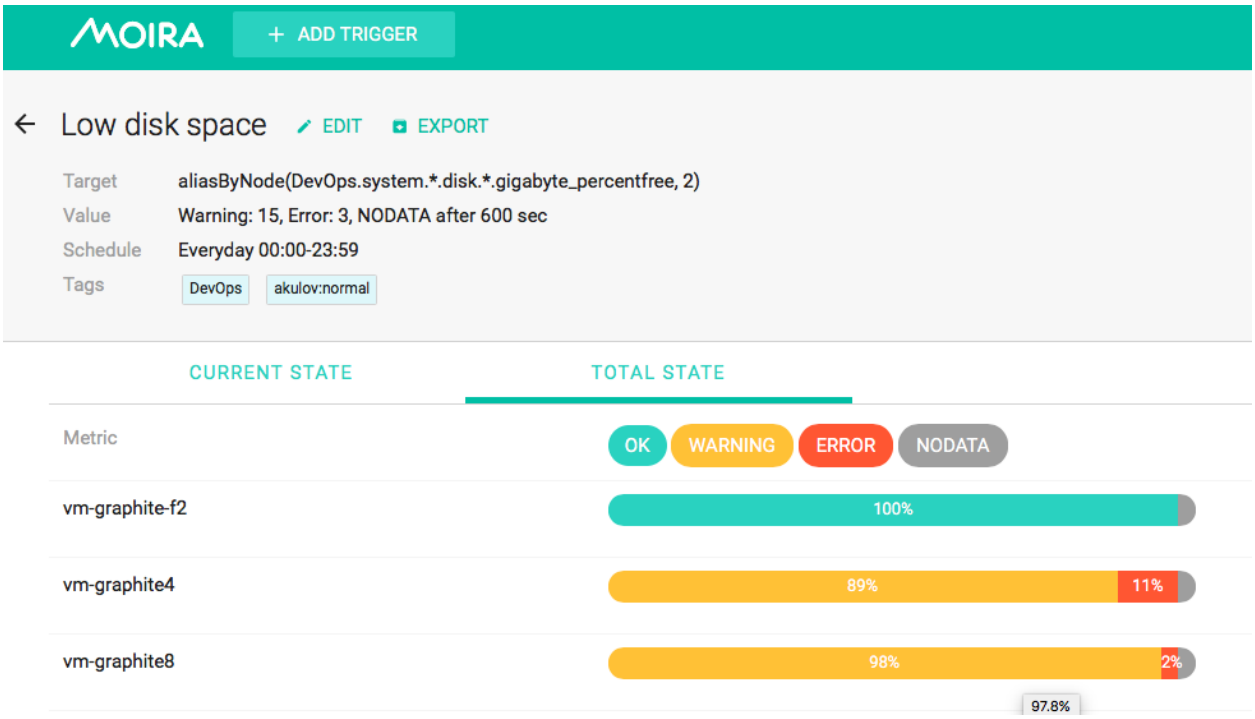
Current State

Moira shows current state, current value and time of last event for every separate metric that matches the trigger.



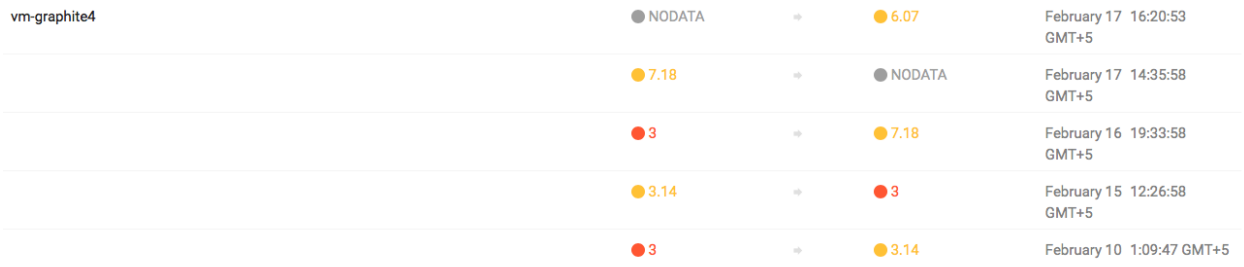
Total State

On this tab you can see total statistics for each separate metric.



Event History

Under the current state information you can see a chronologically sorted list of events for each separate metric. Each event includes time, old and new values. Please, note that the left (old) value is taken from the previous event, and does not represent metric value just before the event.



1.3.6 Throttling

Throttling is a distinctive and controversial feature of Moira. If you are experiencing a delay or any other strange behavior of notifications, chances are, it is because of throttling.

To understand throttling, imagine two triggers:

1. Send notification if CPU load on any of your servers is more than 75%.
2. Send notification if there is a fire in your server room.

It is a busy day, your servers are overloaded, and you are receiving a ton of notifications about CPU load. Probably, you already have several dozens of notifications in your inbox. You will likely delete all of them at once, and you probably won't notice that one of these hundreds of letters was about a fire in your server room.

So, the problem is: one misconfigured trigger spoils everything by spamming your inbox with irrelevant notifications. Moira provides a protection mechanism called throttling. Simple rules:

1. If a trigger sends more than 10 notifications per 1 hour, limit this trigger to 1 message per 30 minutes.
2. If a trigger sends more than 20 notifications per 3 hours, limit this trigger to 1 message per 1 hour.

It works like this:

- First notification is delivered immediately.
- Second notification is delivered immediately.
- ...
- Tenth notification is delivered immediately, and you get a warning: "Please, fix your system or tune this trigger to generate less events."
- Next notifications are delayed so that you receive one message per 30 minutes/1 hour. Nothing is lost, you just receive one message with pack of events. Every message contains a warning: "Please, fix your system or tune this trigger to generate less events."

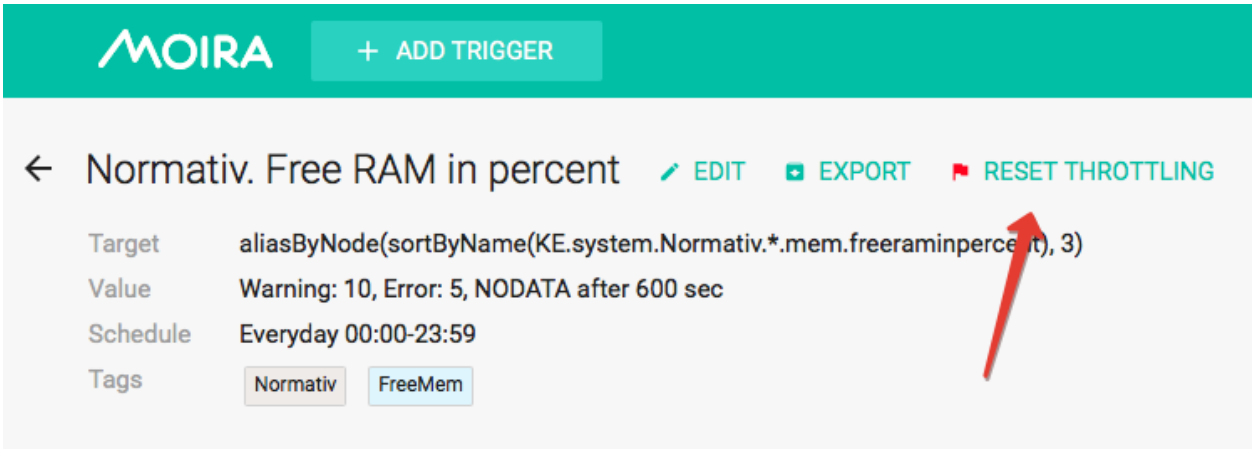
Moira will enable and disable throttling automatically based on frequency of events.

Disabling Throttling

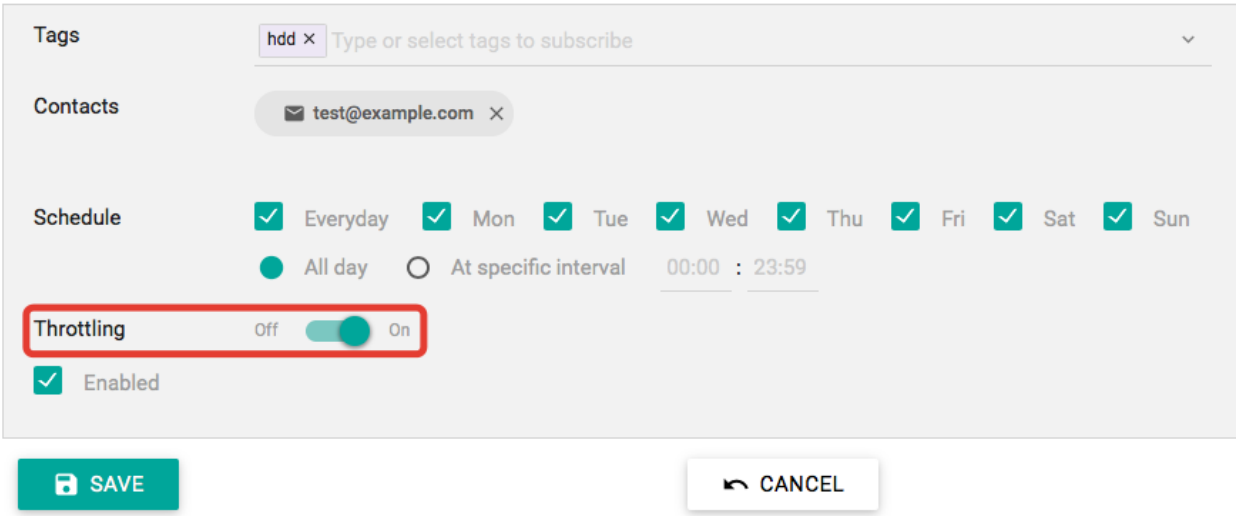
There are four ways to disable throttling for a specific trigger:

1. Listen to the warning message. That is, fix your system to generate less events. Or change trigger thresholds. Or use Graphite functions like `movingAverage` to remove spikes from your metric graph. This is the best method to deal with throttling.
2. Enable maintenance mode for some of your metrics. This will temporarily disable checking of a metric and give you time to fix the system:

3. Manually reset throttling for your trigger. This basically means that you've fixed the system and would like to resume operation normally. It won't help if your trigger is still spamming notifications:



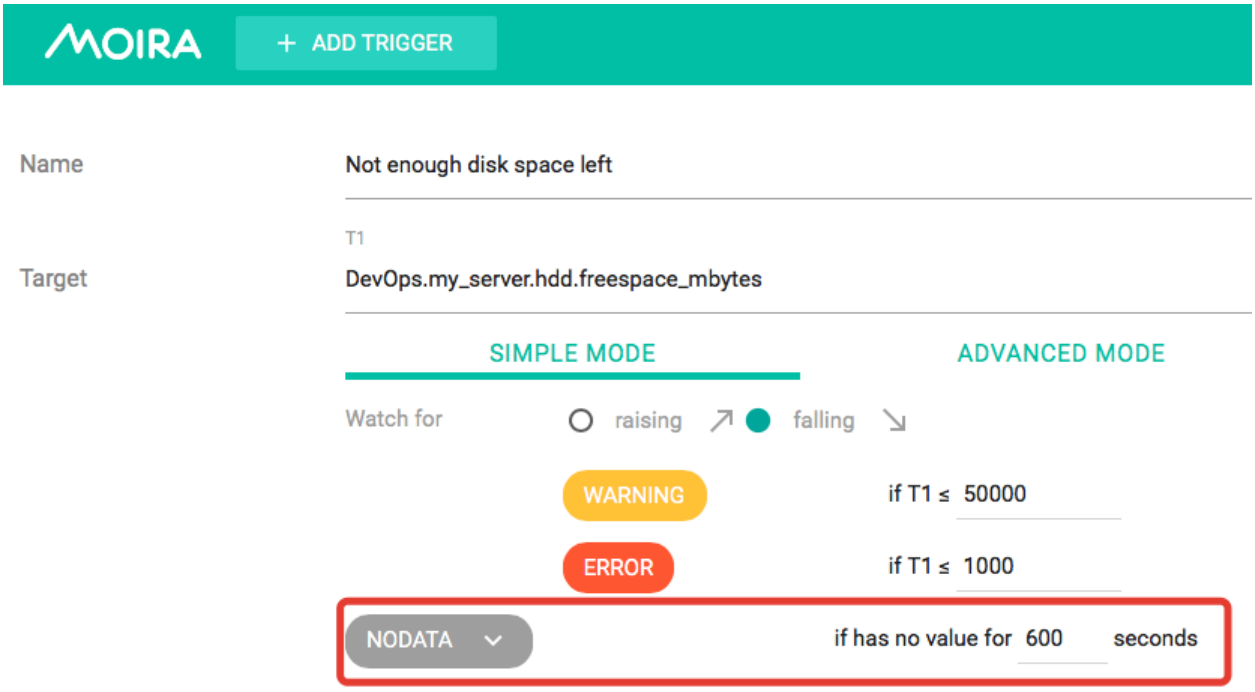
4. Disable throttling entirely for a trigger. *This is not recommended*, unless you really know what you are doing:



1.3.7 Dealing with NODATA

If you have a simple trigger (like the one described in Simple Threshold Trigger), you probably know what happens when a metric has a very high value or a very low value. Free disk space is too low? You get a notification.

But what if your metric has *no* value? Literally, what if Moira is not receiving any data for your metric? How can you know, whether you have enough disk space left or not? In this case, a trigger setting defines the behavior:



When Moira hasn't been receiving data for more than default 600 seconds, it will set a special NODATA state for this metric. You can set any other state or change time delay here. For example, if you have an error metric, and no data means no errors, you should set this to OK.

You can also select DEL here to automatically delete all metrics that no longer provide data. A simple use case is when you often rename metrics and Moira quickly becomes flooded with old irrelevant metric names.

> **Warning:** DEL is a dangerous setting, you can easily miss a real notification if your system stops sending metric data.

You will receive notifications when your metric goes in and out of NODATA state, just like any other state.

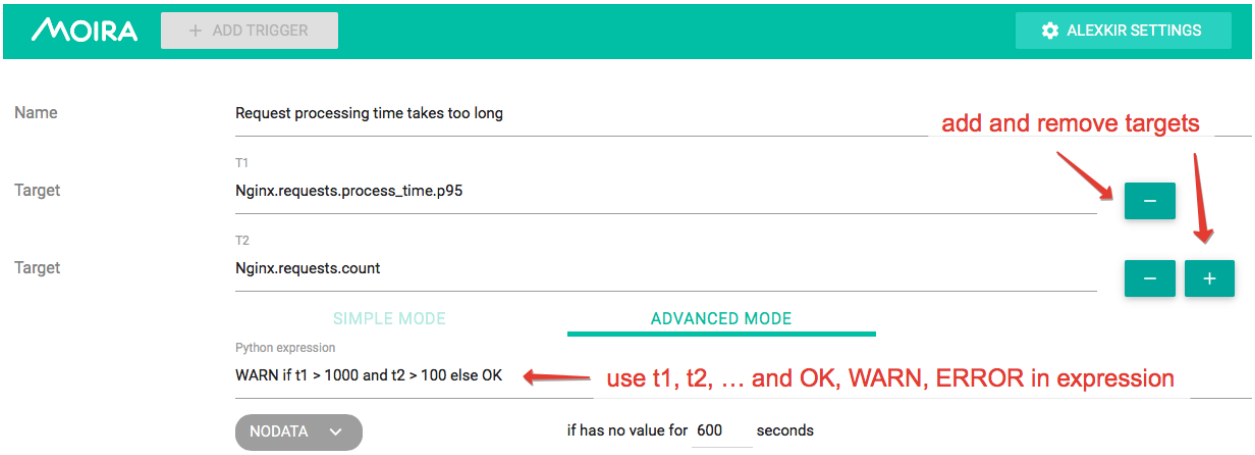
1.3.8 Advanced Mode Trigger

Sometimes a simple trigger (Simple Threshold Trigger) doesn't provide enough flexibility for your task.

For example, you may want to receive a notification when 5% of user requests take up more than a second to process, but only if there are more than 100 requests per minute. Usually, you will have two separate metrics for this:

1. `Nginx.requests.process_time.p95` - 95th percentile of request processing time in milliseconds
2. `Nginx.requests.count` - request count per minute

Maybe you can construct a monstrous Graphite expression to reflect this combination, but Moira's Advanced Mode is better:



You can use any Python expression with predefined constants here:

- `t1`, `t2`, ... are values from your targets
- `OK`, `WARN`, `ERROR`, `NODATA` are states that must be the result of evaluation
- `PREV_STATE` is equal to previously set state, and allows you to prevent frequent state changes

Note: Only T1 target can resolve into multiple metrics in Advanced Mode. T2, T3, ... must resolve to single metrics. Moira will calculate expression separately for every metric in T1.

Note: Expression evaluation is intended to be as safe as possible. You can't use any Python functions here.

Any incorrect expressions or bad syntax will result in EXCEPTION trigger state.

1.3.9 Hidden Pages

Some rarely used features of Moira are hidden on pages that are not linked from anywhere. This is a deliberate design decision to reduce visual clutter in the main UI.

You need to type an address of a hidden page manually, like this: `http://moira.example.com/#/hidden_page/`.

Notifications

If Moira encounters an error while sending a notification, it will try again every minute for the next 24 hours. After that period, the notification is considered lost. You can configure this via `resending_timeout` parameter.

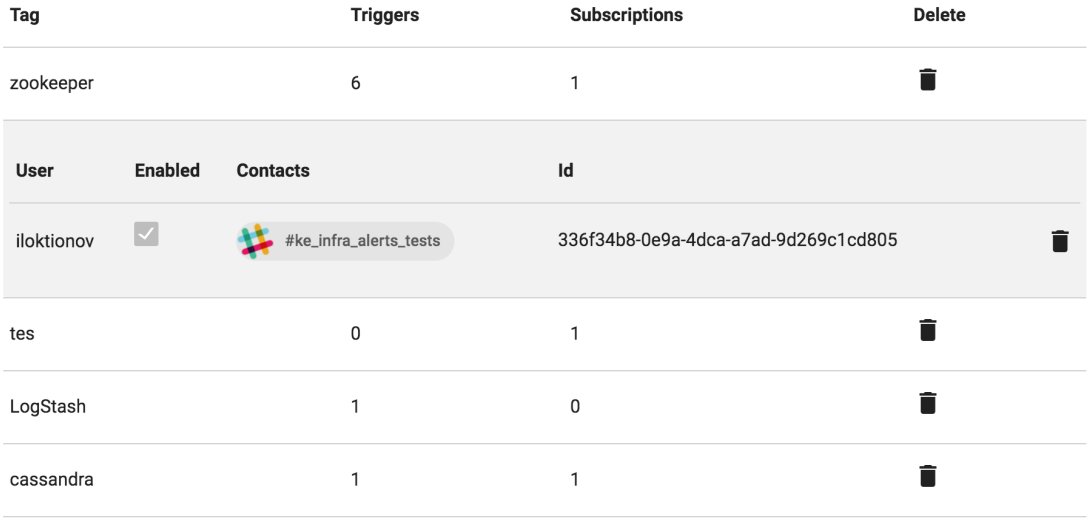
In some cases notifications will never be delivered, for example if a user specifies invalid contact.

If you need to interrupt this behavior, you can manually delete failing notifications at `/#/notifications/`.

Timestamp	Contact		Throttled	Fails	Remove
Today 11:07:51 PM	telegram	@asdasdasdaaa	false	1	

Tags

Deleting a tag is a rare and dangerous operation, but you still can do it at `/#/tags/`.



Tags list shows how many triggers and subscriptions use a tag.

You can not delete a tag if there is at least one trigger that uses it. You **can** delete a tag that is used in a subscription. Also, you can delete subscriptions of any user for a tag from this page.

1.4 Development

All services use Redis database to store and exchange data. Therefore, it is important to maintain an accurate description of data storage formats and conventions.

Following topics describe database structure, running tests, developing notification plugins and more.

1.4.1 Architecture

Terminology

Pattern

A Graphite pattern is a single dot-separated metric name, possibly containing one or more wildcards.

Examples:

```
server.web*.load
server.web{1,2,3}.load
server.web1.load
```

Target

A Graphite target is one or more patterns, possibly combined using Graphite functions.

Examples:

```
averageSeries(server.web*.load)
```

Metric

A metric is a single time-series that is a result of parsing some Graphite target.

Some targets produce a single metric, for example:

```
server.web1.load
highestCurrent(server.web*.load)
```

Some targets produce several metrics, for example:

```
movingAverage(server.web*.load, 10)
```

State

Moira stores separate state for every metric. Each metric can be in only one state at any moment:

Trigger

Trigger is a configuration that tells Moira which metrics to watch for. Triggers consist of:

- Name. This is just for convenience, user can enter anything here.
- One or more targets.
- WARN and ERROR value limits, or a Python expression to calculate state.
- One or more tags.
- TTL value. Metrics switch to NODATA state when new data doesn't arrive for TTL seconds.
- Check schedule. For example, a trigger can be set to check only during business hours.

Last Check

When Moira checks a trigger, it stores the following information on each metric:

- Current value.
- Current timestamp.
- Current state.

Trigger Event

When Moira checks a trigger, if any of the metric states change, Moira generates an event. Events consist of:

- Trigger ID.
- Metric name (as given by parsed target).
- New state.
- Previous state.

- Current timestamp.

Tags

Tags are simple string markers for grouping of triggers and configuring subscriptions.

Subscription

Moira generates notifications for an event only if trigger tags match any of the user-created subscriptions. Each subscription consists of:

- One or more tags.
- Contact information.
- Quiet time schedule.

Dataflow

Save and Filter Incoming Metrics

When user adds a new trigger, Moira parses patterns from targets and saves them to `moira-pattern-list` key in Redis. Cache rereads this list every second. When a metric value arrives, Cache checks metric name against the list of patterns. Matching metrics are saved to `moira-metric:<metricname>` keys in Redis. Redis pub/sub mechanism is used to inform Checker-master of incoming metric value that should be checked as soon as possible.

Checker-master reads triggers by pattern from `moira-pattern-triggers:<pattern>` key in Redis and adds triggers to check set at `moira-triggers-tocheck` Redis key. In case of no incoming data, all triggers are added to check once per `nodata_check_interval` setting.

Check Triggers

Checker-worker constantly reads `moira-triggers-tocheck` key in Redis and calculates trigger targets values. Target can contain one or multiple metrics, so results are written per metric.

`moira-metric-last-check:<trigger_id>` Redis key contains last check JSON with metric states.

When a metric changes its state, a new event is written to `moira-trigger-events` Redis key. This happens only if value timestamp falls inside time period allowed by trigger schedule.

If a metric has been in NODATA or ERROR state for a long period, every 24 hours Moira will issue an additional reminder event.

Trigger switches to EXCEPTION state, if any exception occurs during trigger checking.

Process Trigger Events

Notifier constantly pulls new events from `moira-trigger-events` Redis key and schedules notifications according to subscription schedule and throttling rules. If and only if a trigger contains *all* of the tags in a subscription, a notification is created for this subscription.

Subscription schedule delays notifications of occurred event to the beginning of next allowed time interval. Note that this is different from trigger schedule, which suppresses event generation entirely.

Throttling rules will delay notifications:

- If there are more than 10 events per hour, a notification will be sent at most once per 30 minutes.
- If there are more than 20 events per 3 hours, a notification will be sent at most once per hour.

Scheduled notifications are written to `moira-notifier-notifications` Redis key.

Process Notifications

Notifier constantly pulls scheduled notifications from `moira-notifier-notifications` Redis key. It calls sender for certain contact type and writes notification back to Redis in case of sender error.

1.4.2 API and Checker Microservices

Both microservices are built with Python Twisted framework and include a part of Graphite sources for Graphite function evaluation.

Note: Run `redis-server` before you run these microservices.

Install .

```
python setup.py install
```

Run API.

```
moira-api
```

Run Checker.

```
moira-checker
```

Run tests.

```
cd moira-alert/worker/tests
trial functional
```

1.4.3 UI Application

UI is a static web application built with TypeScript, AngularJS and webpack.

Install dependencies.

```
npm install
```

Run webpack dev server at http://localhost:8080.

```
npm start
```

Note: UI doesn't work without API microservice

Run tests.

```
karma start
```

1.4.4 Notifier

This microservice is written in Go. To run tests, first get all dependencies.

```
cd notifier/notifier
go get
```

Then, run Ginkgo tests.

```
cd notifier/tests
ginkgo
```

Writing Your Own Notification Sender

First, look at built-in senders:

- notifier/slack
- notifier/pushover
- notifier/mail

All of them implement interface `Sender` from `interfaces.go`. Please, note that scheduling and throttling require that senders support packing several events into one message.

You should include your new sender in `configureSenders` method of `notifier/main.go` with appropriate type.

Senders have access to their settings in common config, which is passed to the `Init` method.

1.4.5 Redis Data Layer

Redis database objects:

- KEY moira-metric-last-check:<trigger_id>
- SET moira-pattern-metrics:<pattern>
- SET moira-pattern-triggers:<pattern>
- KEY moira-trigger:<trigger_id>
- LIST moira-trigger-events
- LIST moira-trigger-events-ui
- SORTED SET moira-metric-data:<metric>
- KEY moira-metric-retention:<metric>
- SET moira-triggers-list
- SET moira-pattern-list
- SET moira-trigger-tags:<trigger_id>
- SET moira-tags
- SET moira-tag-triggers:<tag>

- SET moira-triggers-tocheck
- SET moira-user-subscriptions:<login>
- SET moira-tag-subscriptions:<tag>
- KEY moira-subscription:<subscription_id>
- KEY moira-contact:<contact_id>
- SET moira-user-contacts:<login>
- SORTED SET moira-trigger-events:<trigger_id>
- SET moira-notifier-throttling-beginning:<trigger_id>
- KEY moira-tag:<tag>
- KEY moira-metric-check-lock:trigger_id
- SORTED SET moira-triggers-checks
- SET moira-bad-state-triggers
- KEY moira-selfstate:checks-counter

class moira.db.**Db**

Redis database service class

accuireTriggerCheckLock(*self*, *trigger_id*, *timeout*)

Try to accuire lock for trigger check until timeout

Parameters

- **trigger_id** (*string*) – trigger identity
- **timeout** (*float*) – timeout in seconds

addPatternMetric(*self*, *pattern*, *metric*)

Add metric to set moira-pattern-metrics:<pattern>

Parameters

- **pattern** (*string*) – pattern of graphite that match multiple metric
- **metric** (*string*) – metric of graphite

addTriggerCheck(*self*, *trigger_id*)

Add *trigger_id* to set moira-triggers-tocheck

Parameters **trigger_id** (*string*) – trigger identity

addTriggerTag(*self*, *trigger_id*, *tag*)

Creates redis transaction for:

- Add *tag* to set moira-trigger-tags:<trigger_id>
- Add *trigger_id* to set moira-tag-triggers:<tag>
- Add *tag* to set moira-tags

Parameters **trigger_id** (*string*) – trigger identity

cleanupMetricValues(*self*, *metric*, *toTime*)

Remove metric values from sorted set moira-metric-data:<metric> until toTime

Parameters **startTime** (*long*) – unix epoch time

delMetric (*self*, *metric*)
Delete metric sorted set moira-metric-data:<metric>

Parameters **metric** (*string*) – metric of graphite

delPatternMetrics (*self*, *pattern*)
Delete whole set moira-pattern-metrics:<pattern> of metrics for given pattern

Parameters **pattern** (*string*) – pattern of graphite that match multiple metric

delTriggerCheckLock (*self*, *trigger_id*)
Delete lock for trigger check moira-metric-check-lock:<trigger_id>

Parameters **trigger_id** (*string*) – trigger identity

deleteTriggerThrottling (*self*, *trigger_id*)
Read all planning notifications from sorted set moira-notifier-notifications and rescheduling it delivery to now

Parameters **trigger_id** (*string*) – trigger identity

deleteUserContact (*self*, *contact_id*, *login*)

Creates redis transaction for:

- remove key moira-contact:<contact_id>
- remove *contact_id* from set moira-user-contacts:login

Parameters

- **contact_id** (*string*) – contact id
- **login** (*string*) – user login

getAllContacts (*self*, *login*)
Returns all contacts json

Return type array of strings

getContact (*self*, *contact_id*)
Returns contact by given id from key moira-contact:contact_id

Parameters **contact_id** (*string*) – contact id

Return type json dict

getEvents (*self*)
Returns all events for given trigger_id

Parameters **trigger_id** (*string*) – trigger identity

Return type list of dict

getFilteredTriggersChecksPage (*self*, *page*, *size*, *filter_ok*, *filter_tags*)

•Returns filtered triggers page

Parameters

- **start** (*integer*) – start position in range
- **start** – number of triggers
- **filter_ok** (*boolean*) – use triggers set in bad state
- **filter_tags** (*list of strings*) – use tag triggers set

Return type json

getMetricRetention (*self*, *metric*)
Returns metric retention in seconds from key moira-metric-retention:<metric> for given metric

Parameters **metric** (*string*) – metric of graphite

Return type integer

getMetricsValues (*self*, *metrics*, *startTime*, *endTime*)
Read multiple metric values from sorted set moira-metric-data:<metric> from startTime

Parameters

- **metrics** (*list of string*) – list of graphite metric path
- **startTime** (*long*) – unix epoch time
- **endTime** (*long*) – unix epoch time

Return type list of list of tuple ('value timestamp', long)

getNotifications (*self*, *start*, *end*)
Read all planning notifications from sorted set moira-notifier-notifications

Parameters

- **start** (*integer*) – range start
- **end** (*integer*) – range end

getPatternMetrics (*self*, *pattern*)
Read all metrics from set moira-pattern-metrics:<pattern> for given pattern

Parameters **pattern** (*string*) – pattern of graphite that match multiple metric

Return type set of strings

getPatternTriggers (*self*, *pattern*)
Returns all trigger identifiers from set moira-pattern-triggers:<pattern>

Parameters **pattern** (*string*) – pattern of graphite that match multiple metric

Return type set of strings

getPatterns (*self*)
Returns all patterns from SET moira-pattern-list

Return type set of strings

getSubscription (*self*, *sub_id*)
Returns subscription by given id from key moira-subscription:subscription_id

Parameters **sub_id** (*string*) – subscription id

Return type json dict

getTag (*self*, *tag*)
Returns tag data from key moira-tag:{0}

Return type json dict

getTagSubscriptions (*self*, *tag*)
Returns all subscriptions by given tag from set moira-tag-subscriptions:<tag>

Return type list of strings

getTagTriggers (*self*, *tag*)
Returns all trigger is for given tag from set moira-tag-triggers:<tag>

Return type set of strings

getTags (*self*)
Returns all tags from set moira-tags with tag data

Return type dict

getTrigger (*self*, *trigger_id*)

•Read trigger by key moira-trigger:<trigger_id>

•Unpack trigger json

Parameters

- **tags** (*boolean*) – get with tags
- **trigger_id** (*string*) – trigger identity

Return type tuple(json, trigger)

getTriggerLastCheck (*self*, *trigger_id*)
Returns last trigger check from key moira-metric-last-check:<trigger_id>

Parameters **trigger_id** (*string*) – trigger identity

Return type json dict

getTriggerThrottling (*self*, *trigger_id*)
Returns planning trigger notification timestamp in future or 0 from key moira-notifier-next:<trigger_id>

Parameters **trigger_id** (*string*) – trigger identity

Return type long

getTriggerToCheck (*self*)
Pop trigger id from set moira-triggers-tocheck

Return type string

getTriggers (*self*)
Returns all triggers id from set moira-triggers-list

Return type set of strings

getTriggersChecks (*self*)

•Returns all triggers with it check

Return type json

getTriggersChecksPage (*self*, *start*, *size*)

•Returns triggers range from sorted set moira-triggers-checks

Parameters

- **start** (*integer*) – start position in range
- **start** – number of triggers

Return type json

getUserContacts (*self*, *login*)
Returns contacts ids by given login from set moira-user-contacts:<login>

Parameters **login** (*string*) – user login

Return type set of strings

getUserSubscriptions (*self*, *login*)
Returns subscriptions ids by given login from set moira-user-subscriptions:<login>

Parameters **login** (*string*) – user login

Return type set of strings

pushEvent (*self*, *event*)

Creates redis transaction for:

- Add event to beginning of list moira-trigger-events as json string
- Trim list to 100 events

Parameters **event** (*dict*) – trigger state changing event

removeNotification (*self*, *json*)
Remove planning notification by given json from sorted set moira-notifier-notifications

Parameters **json** (*string*) – notification json

removePattern (*self*, *pattern*)
Remove pattern from set moira-pattern-list

Parameters **pattern** (*string*) – pattern of graphite that match multiple metric

removePatternTriggers (*self*, *pattern*)
Delete all trigger identifiers from set moira-pattern-triggers:<pattern>

Parameters **pattern** (*string*) – pattern of graphite that match multiple metric

removeTag (*self*, *tag*)

Creates redis transaction for:

- Remove *tag* from set moira-tags
- Delete key moira-tag-subscriptions:<tag>
- Delete key moira-tag-triggers:<tag>
- Delete key moira-tag:<tag>

removeTrigger (*self*, *trigger_id*)

Creates redis transaction for:

- Delete key moira-trigger:<trigger_id>
- Remove *trigger_id* from set moira-triggers-list
- Remove *trigger_id* from set moira-pattern-triggers:<pattern>

Parameters **trigger_id** (*string*) – trigger identity

removeTriggerLastCheck (*self*, *trigger_id*)
Delete trigger last check from key moira-metric-last-check:<trigger_id>

Parameters **trigger_id** (*string*) – trigger identity

removeTriggerTag (*self*, *trigger_id*, *tag*)

Creates redis transaction for:

- Remove *tag* from set moira-trigger-tags:<trigger_id>
- Remove *trigger_id* from set moira-tag-triggers:<tag>

Parameters **trigger_id** (*string*) – trigger identity

removeUserSubscription (*self*, *login*, *sub_id*)

Creates redis transaction for:

- delete key moira-subscription:subscription_id
- remove *sub_id* from set moira-user-subscriptions:login

Parameters

- **login** (*string*) – user login
- **sub_id** (*string*) – subscription id

saveTrigger (*self*, *trigger_id*, *trigger*)

Creates redis transaction for:

- Saving *trigger_json* to key moira-trigger:<trigger_id>
- Add *trigger_id* to set moira-triggers-list
- Update patterns set moira-pattern-list
- Update trigger patterns set moira-pattern-triggers:<pattern>

Parameters

- **trigger** (*dict*) – trigger json object
- **trigger_id** (*string*) – trigger identity

saveUserContact (*self*, *login*, *contact*)

Creates redis transaction for:

- save *contact* json to key moira-contact:<contact_id>
- add *contact_id* to set moira-user-contacts:login

Parameters

- **login** (*string*) – user login
- **contact** (*json dict*) – contact data

Return type json dict

saveUserSubscription (*self*, *login*, *sub*)

Creates redis transaction for:

- save *sub* json to key moira-subscription:<sub_id>
- add *sub_id* to set moira-user-subscriptions:<login>

Parameters

- **login** (*string*) – user login
- **sub** (*json dict*) – subscription data

Return type json dict

setTag(*self*, *tag*, *data*)
Save tag data to key moira-tag:{0}

Return type json dict

setTriggerCheckLock(*self*, *trigger_id*)
Try to accuire lock for trigger check moira-metric-check-lock:<trigger_id>

Parameters **trigger_id** (*string*) – trigger identity

setTriggerLastCheck(*self*, *trigger_id*, *check*)
Save trigger last check to key moira-metric-last-check:<trigger_id>

Parameters

- **trigger_id** (*string*) – trigger identity
- **check** (*json dict*) – trigger checking result

setTriggerMetricsMaintenance(*self*, *trigger_id*, *metrics*)
Atomic change of trigger last check and set metric maintenance

Parameters

- **trigger_id** (*string*) – trigger identity
- **metrics** (*dict*) – metrics maintenance flags

startService(*self*)
Creates redis connection pool

stopService(*self*)
Disconnect from redis connection pool

moira.db.**audit**(*f*)
Write json object changes to audit.log

1.5 Contact Moira Developers

The best way to contact us is to visit our Gitter chat. We usually reply within a day, but sometimes immediately :)

CHAPTER 2

Overview

Moira is a real-time alerting tool, based on Graphite data.

2.1 Key Features

- **Graphite storage independence**

  Some Graphite queries are *very* ineffective. Tools like Seyren multiply this effect by making lots of ineffective queries every minute, overloading your cluster. Moira relies on the incoming metric stream, and has its own fast cache for recent data.

- **Support for all Graphite functions**

  Graphite function library is embedded directly into the source code of Moira. You can use any function and get predictable results, like in your Graphite dashboards.

- **Support for custom Python expressions**

  If simple warning/error threshold is not enough, you can write flexible Python expressions to calculate trigger state based on metric data.

- **Tags for triggers and subscriptions**

  When several teams/services share one monitoring tool, it is essential to provide some way of filtering triggers and subscriptions in the UI. Moira has a flexible tag system.

- **Extendable notification channels**

  Moira has support for email, Slack and Pushover notifications out-of-the-box. But you can always write your own plugin in Go and rebuild Moira Notifier microservice.

- **Alarm fatigue protection**

  Sometimes one of your triggers goes mad and switches back and forth between states, sending you hundreds of notifications. Sometimes you just ignore and delete all messages, accidentally also deleting one that is actually important. Moira tries to protect you with a feature called *throttling*. It's simple: if one of your triggers starts to send more than 10 messages over an hour, Moira limits this trigger to one message per 30 minutes. Alerts from this trigger are combined, and not lost - just packaged into a single message.

2.2 Limitations

By default, Moira stores metric history for one hour. This ensures performance under heavy load. You can tweak this in config file, but note that performance will degrade.

In order to reduce database load, Moira checks every single trigger at most once every 5 seconds. Probably, your metrics arrive once every minute, so you really won't notice this limitation. You can also tweak this in config file.

2.3 Microservices

In spirit of Graphite architecture, Moira consists of several loosely coupled microservices. You are welcome to replace or to add new ones.

2.3.1 Cache

Cache is a lightweight Go application responsible for receiving lots of metric data in Graphite format. Cache filters received data and saves only metrics that match any of user triggers. This reduces load on all other parts of Moira.

2.3.2 Checker

Checker is a Python application with embedded Graphite functions. Checker watches for incoming metric values and performs checks according to saved trigger settings. When state of any trigger changes, Checker generates an event.

2.3.3 Notifier

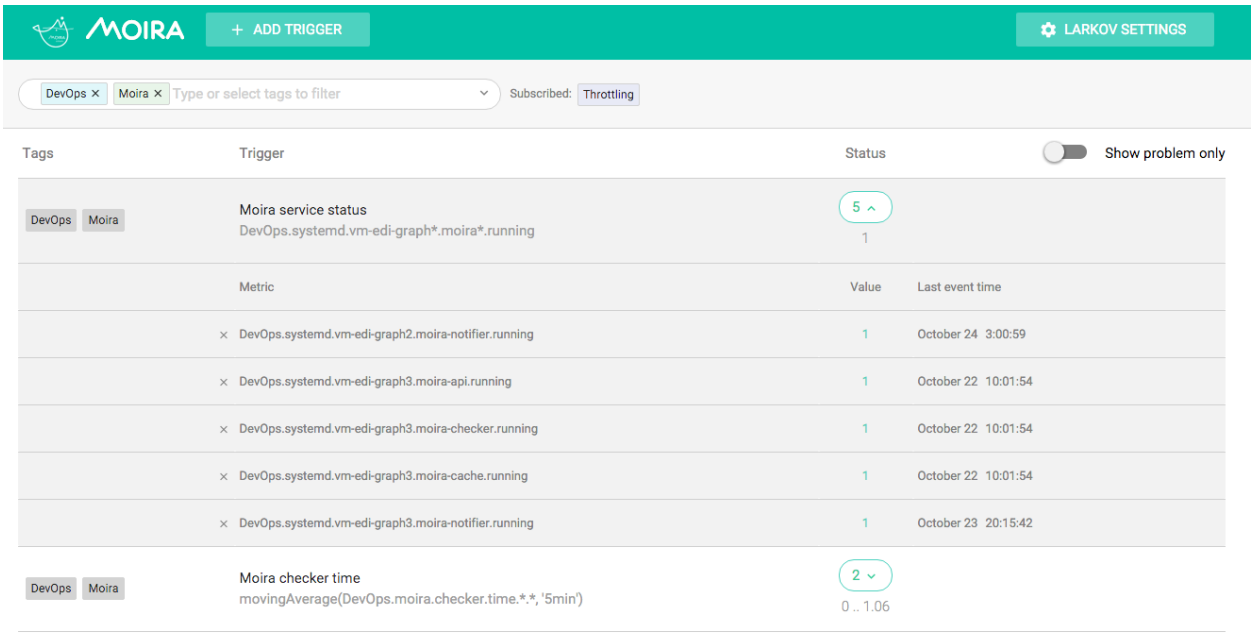
Notifier is a Go application that watches for generated events. Notifier is responsible for scheduling and sending notifications, observing quiet hours, retrying failed notifications, etc.

2.3.4 API

API is a Python application that serves as a backend for UI.

2.3.5 UI

UI is a frontend Angular application, it looks like this:



2.3.6 Database

All services communicate only through a Redis database, without any additional protocols or connections between each other.

m

A

C

D

G

M

P

R

S