
Moira

Release 2.2

Oct 29, 2018

Contents

1	Contents	1
1.1	Overview	1
1.2	Changelog	3
1.3	Installation	5
1.4	User Guide	11
1.5	Development	27
1.6	Contact Moira Developers	31
2	Overview	33
2.1	Key Features	33
2.2	Limitations	34
2.3	Microservices	34

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 every minute making lots of ineffective queries and overloading your cluster. Moira relies on the incoming metric stream, and has its own fast cache for recent data.

- **Support for (almost) all Graphite functions**

Graphite function library ([carbonapi](#)) is embedded directly into Moira source code. You can use any function and get predictable results, like in your Graphite or Grafana dashboards.

- **Support for custom expressions**

If simple warning/error threshold is not enough, you can write flexible [govaluate](#) 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 supports email, [Slack](#), [Pushover](#) and many other channels of notification 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 over 10 messages per 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.

Filter

Filter is a lightweight service responsible for receiving lots of metric data in Graphite format. It 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 an 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 an application that watches for generated events. Notifier is responsible for scheduling and sending notifications, observing quiet hours, retrying failed notifications, etc.

API

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

Web 2.0

Web 2.0 is a frontend React application, it looks like this:

MOIRA

Notifications
Help

DevOps
dbaas

Only Problems

Add Trigger

1
77

Cassandra GC metrics missing
exclude(aliasByNode((Alko,EDI,EDITest,KE,KE-cloud,KE-dev),Cassandra.*,*.GC.StopTheWorld.sum, 0, 2, 3), 'EDI.LegacyCluster')

normal
DevOps
dbaas
Cassandra

18
4
68

EDI Cassandra Data Disk Space Free
aliasByNode(exclude(exclude(EDI.system.*,disk_storage*.gigabyte_percentfree, 'elastic'), 'edi14'), 2, 4)

DevOps
dbaas
EDI
Cassandra
normal

2

EDI Test Cassandra Nodes Down
exclude(exclude(groupByNode(EDITest.Cassandra.*,*.DownEndpointCount.DownEndpointCount, 2, 'maxSeries'), 'CatalogueRtq BenchmarkCluster'), 'EdiRtqLoadCluster')

DevOps
dbaas
EDI
Cassandra
normal

Name	Last event	Value ↓		
EdiStagingCluster	December 4, 18:36:16	0	Maintenance	Del
EdiTestingCluster	December 4, 18:36:16	0	Maintenance	Del

1

KE Cassandras Read Latency
groupByNode(movingMin(KE.Cassandra.*,*.ClientRequest.Read.Latency.99thPercentile,5min),2,'maxSeries')

DevOps
dbaas
Cassandra
normal

30

Alko Cassandra Data Disk Space Free
aliasByNode(Alko.system.*,disk_storage*.gigabyte_percentfree, 2, 4)

DevOps
dbaas
Cassandra
normal
Alko

1.2 Changelog

1.2.1 2.2

- Add Redis Sentinel support.
- Increase new metric event processing speed by adding a cache on metric patterns.
- Update carbonapi (new functions: map, reduce, delay; updated: asPercent).
- Optimize reading metrics while checking trigger (removed unnecessary Redis transaction).
- Add domain autoresolving for self-metrics sending to Graphite.
- Fix concurrent read/write from expression cache.
- Re-enable Markdown in Slack sender.
- Optimize internal metric collection.
- Replace pseudotags with ordinary checkboxes in Web UI (but not on backend yet).
- Fix bug that allowed to create pseudotags (ERROR, etc.) as ordinary tags.
- Add metrics for each trigger handling time.
- Translate pagination.
- Make sorting by status the default option on trigger page.
- Hide tag list on trigger edit page.

- Sort tags alphabetically everywhere.
- Highlight metric row on mouse hover.
- Automatically add tags from search bar when creating new trigger.
- Add metric name to “Trigger has same timeseries names” error message.
- Update event names in case trigger name had changed.
- Fix bug in triggers with multiple targets. Metrics from targets T2, T3, ... were not deleted properly.
- Fix old-style configuration files in platform-specific packages.
- Fix bug that prevented non-integer timestamps from processing.
- Fix logo image background.
- Fix sorting on -s and 0s.
- Fix UI glitch while setting maintenance time.
- Fix retention scheme parsing for some rare cases with comments.

1.2.2 2.1

- Throw an exception if any target except the first one resolves in more than one metric.
- Fix Moira version detection in CI builds.
- Add user login information to API request logs.
- Fix long interval between creating a new trigger and getting data into that trigger.

1.2.3 2.0

Version 2.0 is fully rewritten in Go instead of Python. This implies lower CPU load in Checker and API microservices, but also changes the list of [supported Graphite functions](#).

We also introduce new UI based on React. It is not backwards-compatible with old API, but new API supports both old and new UI.

Breaking Changes

- New structure of *Configuration* files.
- New Advanced mode expression format. Moira 2.0 supports [govaluate](#) expressions instead of Python expressions. Use `moira-cli -convert-expressions` to convert.
- API methods URLs do not have trailing slashes anymore.
- API `/notification` method returns valid JSON list instead of plain text.
- `ttl` parameter in API calls is always a number instead of string.
- API `PUT` methods strictly separate create and update operations.
- There is no `tag maintenance` entity anymore.
- Error messages return valid JSON instead of plain text.
- Support for Graphite functions changed. See [carbonapi](#) compatibility list for details.

Other Improvements

- Internal Graphite metric names changed.
- Numerous bugs fixed. Some new were created :)

1.3 Installation

1.3.1 Manual Installation

Tip: To get Maira running quickly, try *Docker* version

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

1. `golang` version 1.9 or higher
2. `redis` database version 2.8 or higher
3. web server e.g. `nginx`

Build Maira Microservices

```
go get -u github.com/maira-alert/maira
cd $GOPATH/src/github.com/maira-alert/maira
make build
```

You will find binaries in `$GOPATH/src/github.com/maira-alert/maira/build`.

Download Web UI Application

<https://github.com/maira-alert/web2.0/releases/latest>

Download and unpack `.tar.gz` file into Nginx static files directory (e.g. `/var/local/www/maira`).

Configure

1. If you need to override default settings, place configuration files somewhere on your disk (e.g. `/etc/maira/`). You can dive into *Configuration* syntax on a separate page.
2. Place nginx configuration file to proper location (e.g. `/etc/nginx/conf.d/maira.conf`):

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

3. If you need to override UI settings, edit [web.json](#) file. You can find its location in [API configuration](#).

Run

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

```
$GOPATH/src/github.com/moira-alert/moira/build/cache
$GOPATH/src/github.com/moira-alert/moira/build/checker
$GOPATH/src/github.com/moira-alert/moira/build/notifier
$GOPATH/src/github.com/moira-alert/moira/build/api
```

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](#)).

1.3.2 Docker

You can quickly test a local Moira installation using Docker containers from [Docker Hub](#) and docker-compose file in documentation repository.

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

Containers are preconfigured to serve Web UI at localhost:8080 and accept graphite metrics at localhost:2003.

1.3.3 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 each repository release page:

1. <https://github.com/moira-alert/web2.0/releases>
2. <https://github.com/moira-alert/moira/releases>

1.3.4 Configuration

By default, microservices will look for `/etc/moira/<servicename>.yaml`, but you can change this location by passing your path as a command-line parameter `--config`.

On this page you can find examples of configuration files for Moira microservices.

Filter

```
# Use fields MasterName and SentinelAddrs to enable Redis Sentinel support,
# use Host and Port fields otherwise.
redis:
  # Sentinel cluster name
  master_name: ""
  # Sentinel address list, format: {host1_name:port};{ip:port}
```

```

sentinel_addrs: ""
# Node ip-address or host name
host: "moira-redis"
# Node port
port: "6379"
# Database id
dbid: 0
graphite:
# If true, graphite logger will be enabled
enabled: true
# Graphite relay URI
uri: "graphite-relay:2003"
# Moirra metrics prefix. Use 'prefix: {hostname}' to use hostname autoresolver.
prefix: DevOps.moirra
# Metrics sending interval
interval: 60s
filter:
# Metrics listener uri
listen: ":2003"
# Retentions config file path. Simply use your original storage-schemas.conf or
↪ create new if you're using Moirra without existing Graphite installation.
retention-config: /etc/moirra/storage-schemas.conf
log:
log_file: stdout
log_level: info

```

`storage-schemas.conf` is graphite carbon configuration file that should match similarly-named file in your Graphite installation.

Checker

```

redis:
  host: "moira-redis"
  port: "6379"
  dbid: 0
graphite:
  enabled: true
  uri: "graphite-relay:2003"
  prefix: DevOps.moirra
  interval: 60s
checker:
# Period for every trigger to perform forced check on
nodata_check_interval: 60s
# Min period to perform triggers re-check. Note: Reducing of this value leads to
↪ increasing of CPU and memory usage values
check_interval: 10s
# Time interval to store metrics. Note: Increasing of this value leads to
↪ increasing of Redis memory consumption value
metrics_ttl: 3h
# Period for every trigger to cancel forced check (greater than 'NoDataCheckInterval
↪ ') if no metrics were received
stop_checking_interval: 30s
# Equals to the number of processor cores found on Moirra host by default or when
↪ variable is defined as 0.
max_parallel_checks: 0
log:

```

```
log_file: stdout
log_level: info
```

Notifier

```
redis:
  host: "moira-redis"
  port: "6379"
  dbid: 0
graphite:
  enabled: true
  uri: "graphite-relay:2003"
  prefix: DevOps.moiras
  interval: 60s
notifier:
  # Soft timeout to start retrying to send notification after single failed attempt
  sender_timeout: 10s
  # Hard timeout to stop retrying to send notification after multiple failed attempts
  resending_timeout: "1:00"
  # Web-UI uri prefix for trigger links in notifications. For example: with 'http://
  ↪localhost' every notification will contain link like 'http://localhost/trigger/
  ↪triggerId'
  front_uri: "https://moira.example.com"
  # Timezone to use to convert ticks. Default is UTC. See https://golang.org/pkg/time/
  ↪#LoadLocation for more details.
  timezone: Europe/Moscow
  # List of senders, every element has "type" field (one of ["pushover", "slack",
  ↪"mail", "telegram", "twilio sms", "twilio voice", "script"])
  # Every type of sender has additional config fields
  senders:
    - type: pushover
      # Api token for your pushover channel, for more info see https://pushover.net/
      ↪api#registration
      api_token: ...
    - type: slack
      # Api token for your moira notifications slack user, for more info see https://
      ↪get.slack.help/hc/en-us/articles/215770388-Create-and-regenerate-API-tokens
      api_token: ...
    - type: telegram
      # Api token for your telegram bot, for more info about creating bot and get_
      ↪token see https://core.telegram.org/bots#3-how-do-i-create-a-bot
      api_token: ...
    - type: mail
      mail_from: ...
      smtp_host: ...
      smtp_port: ...
      # Skip SMTP server certificate chain validation if false
      insecure_tls: false
      # Uses "mail_from" if empty
      smtp_user: ...
      smtp_pass: ...
      # Email template file path (standard Go templates), if empty use default_
      ↪template
      template_file: ...
    - type: twilio sms
      api_asid: ...
```

```

    api_authtoken: ...
    api_fromphone: ...
    # URL that responds with TwiML config for voice message generation, see https://
    ↪www.twilio.com/docs/api/twiml/voice-overview
    voiceurl: ...
    append_message: true
  - type: twilio voice
    api_asid: ...
    api_authtoken: ...
    api_fromphone: ...
  - type: script
    name: ...
    # Executable path. File must exist on all machines where notifier is running.
    # You can use ${trigger_name} and ${contact_value} in command-line parameters,
    # they will be replaced with trigger name and contact (as specified in web_
    ↪interface).
    exec: ...
  # Self state monitor configuration section. Note: No inner subscriptions is_
  ↪required. It's own notification mechanism will be used.
  moira_selfstate:
    enabled: true
    # Max Redis disconnect delay to send alert when reached
    redis_disconnect_delay: 60s
    # Max Filter metrics receive delay to send alert when reached
    last_metric_received_delay: 120s
    # Max Checker checks perform delay to send alert when reached
    last_check_delay: 120s
    # Self state monitor alerting interval
    notice_interval: 300s
    # Contact list for Self state monitor alerts, use this like delivery channels in_
    ↪web-ui
    contacts:
      - type: mail
        value: devopsteam@example.com
  log:
    log_file: stdout
    log_level: info

```

API

```

redis:
  host: "moira-redis"
  port: "6379"
  dbid: 0
api:
  # Api local network address. Default is ':8081' so api will be available at http://
  ↪moira.company.com:8081/api
  listen: ":8081"
  # If true, CORS for cross-domain requests will be enabled. This option can be used_
  ↪only for debugging purposes.
  enable_cors: false
  # Web_UI config file path. If file not found, api will return 404 in response to
  ↪"api/config"
  web_config_path: "/etc/moira/web.json"
log:
  log_file: stdout

```

```
log_level: info
```

UI

```
{
  "contacts": [
    {"type": "pushover", "validation": ""},
    {"type": "slack", "validation": "^[@#].+$"},
    {"type": "telegram", "validation": "", "title": "#channel, @username, ↵
↵group", "help": "required to grant @ExampleMoiraBot admin privileges for channels, ↵
↵or /start command in groups and personal chats"},
  ],
  "supportEmail": "devops@example.com"
}
```

1.3.5 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.

Unfortunately, the Carbon-Relay 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 is a basic example configuration which defines two clusters and sends all metrics to both at once. One cluster is 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
;
```

1.3.6 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, Maira will assume that user id is "anonymous".

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

Example of Nginx configuration

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

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

    listen 0.0.0.0:80 default_server;

    location / {
        root /var/www/maira;
        index index.html;
        try_files $uri $uri/ /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.

1.4 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.

1.4.1 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 Maira's Simple Mode:

MOIRA

Notifications
Help

Add trigger

Name

Not enough disk space left

Description

You should first look at /var/log size: it is #1 reason of disk space shortage on this server.

Target

T1 DevOps.my_server.hdd.freespace_mbytes

+ Add one more

Simple mode

Advanced mode

☐ Watch for value raising:

WARN if T1 >=

ERROR if T1 >=

☒ Watch for value falling:

WARN if T1 <=

50000

ERROR if T1 <=

1000

☐ NODATA

if has no value for

600

Watch time

☒ Mon
☒ Tue
☒ Wed
☒ Thu
☒ Fri
☒ Sat
☒ Sun

☒ All day
☐ At specific interval

00:00 — 23:59

Tags

hdd

Search results

hdd

Add trigger

Cancel

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.

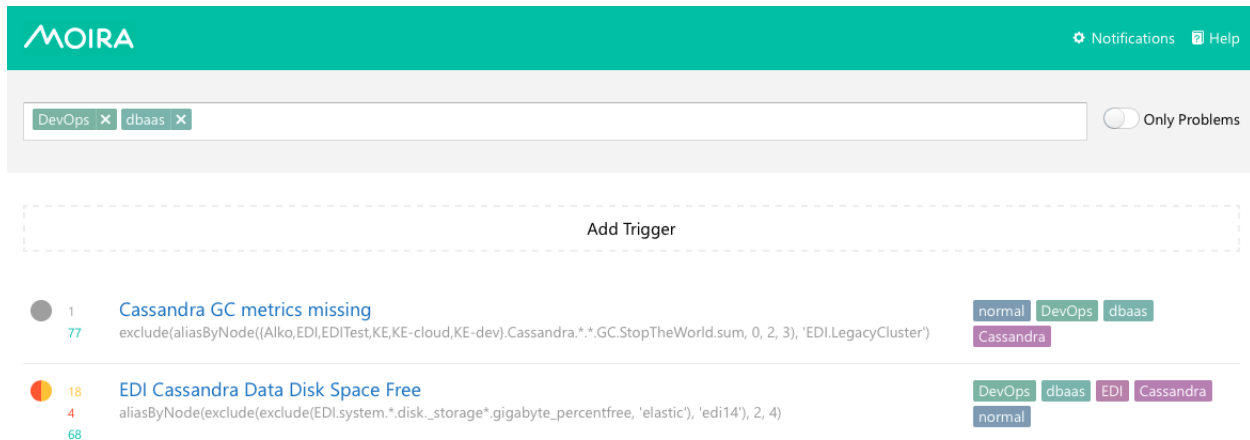
You can even use Graphite functions like `movingAverage(DevOps.my_server.hdd.freespace_mbytes, 10)`. Maira understands everything that Graphite itself understands. See [appropriate documentation](#).

Thresholds

In simple mode you need to specify two threshold values: WARNING and ERROR. In our example, lower values are bad, so we set warning threshold greater than error threshold. In this case, Maira 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 and select Watch for value raising option.

Tags

In Maira, 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.

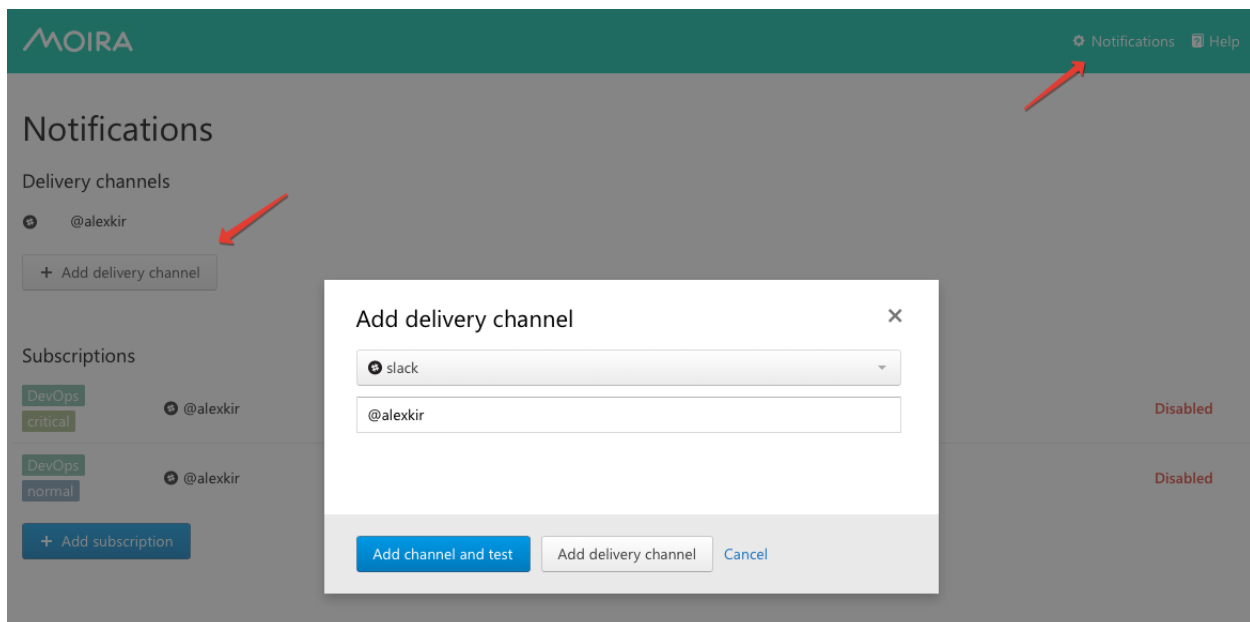
Subscriptions

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

1.4.2 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.

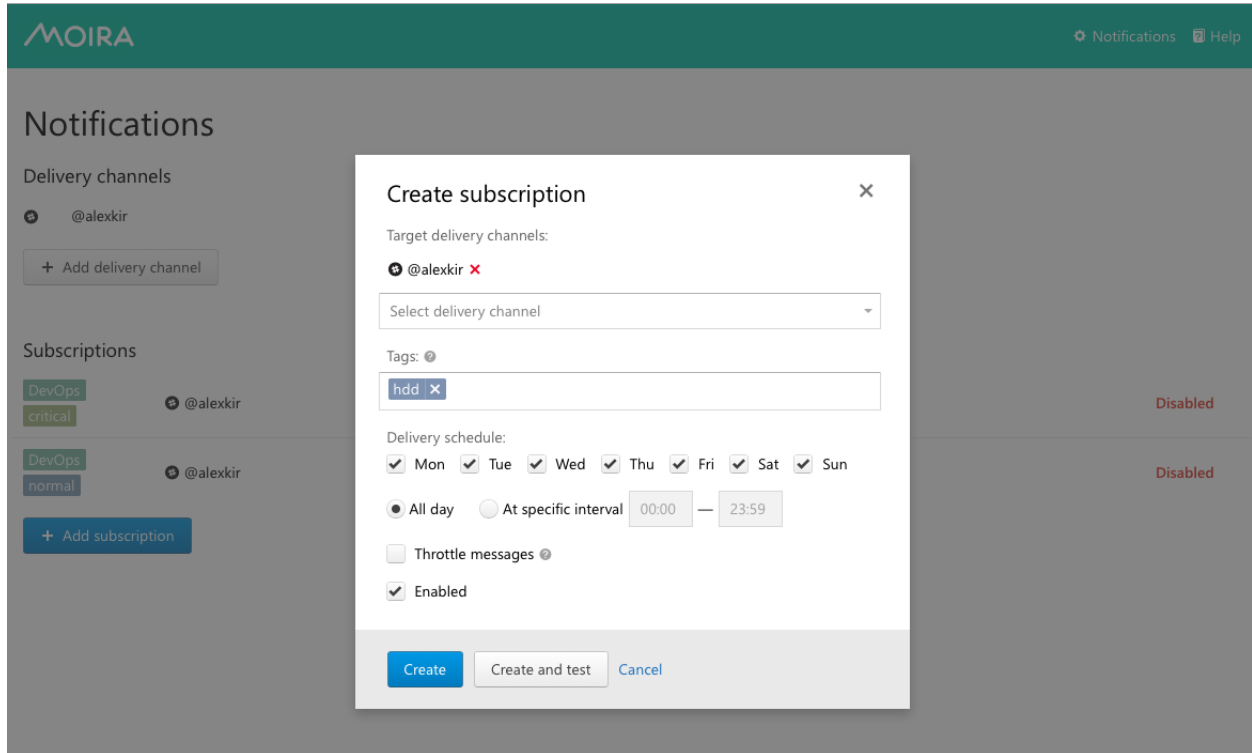
First, add some delivery channels:



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

Consult [Security](#) page for instructions on separating user accounts.

Once you have at least one channel, you can create a subscription. Press + Add subscription 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

Create and Test

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

1.4.3 Efficient Triggers

To use Maira 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. It happens 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?

Later, when you learn to understand your system, you will need to adjust your triggers. And that's when you need to understand Maira.

States

Unlike many other tools providing several distinct level systems like “priority” and “severity”, Maira 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 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 delivery 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 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 delivery channels here, like Pushover.

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

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

If a delivery channel supports high-priority messages (like Pushover does), Maira 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:



In the beginning every metric is in this state. 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 periods 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 zeros 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 minimum 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 may and should remove any functions that make no sense in Moira (like `sortByTotal`) and can generate unwanted side effects (like `averageAbove`).

1.4.4 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:

Edit subscription ✕

Target delivery channels:

 @alexkir ✕

Select delivery channel ▼

Tags: 


DevOps ✕ critical ✕

Delivery schedule:

☒ Mon ☒ Tue ☒ Wed ☒ Thu ☒ Fri ☐ Sat ☐ Sun

☐ All day ☒ At specific interval

08:00 — 17:59

☒ Throttle messages 

☐ Enabled

Update

Update and test

Delete

Cancel

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 and Event History*).

Trigger Watch Time

Let's say, you have a popular website, that serves over 1000 page views per second during a 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 at night all of your users sleep, 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:

Simple mode

Advanced mode

☐ Watch for value raising:

WARN if T1 >=

ERROR if T1 >=

☒ Watch for value falling:

WARN if T1 <=

ERROR if T1 <=

☒ NODATA if has no value for

Watch time

☒ Mon
 ☒ Tue
 ☒ Wed
 ☒ Thu
 ☒ Fri
 ☒ Sat
 ☒ Sun

☐ All day
 ☒ At specific interval

—

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 and Event History](#)).

1.4.5 Current State and Event History

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

Current State

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

MOIRA

NotificationsHelp

Low disk space

EditExport

Target

aliasByNode(movingMin(DevOps.system.*.disk.*.gigabyte_percentfree, '30min'), 2, 4)

Value

Warning: 11, Error: 3, Set NODATA if has no value for 600 seconds

Schedule

Everyday 00:00–23:59

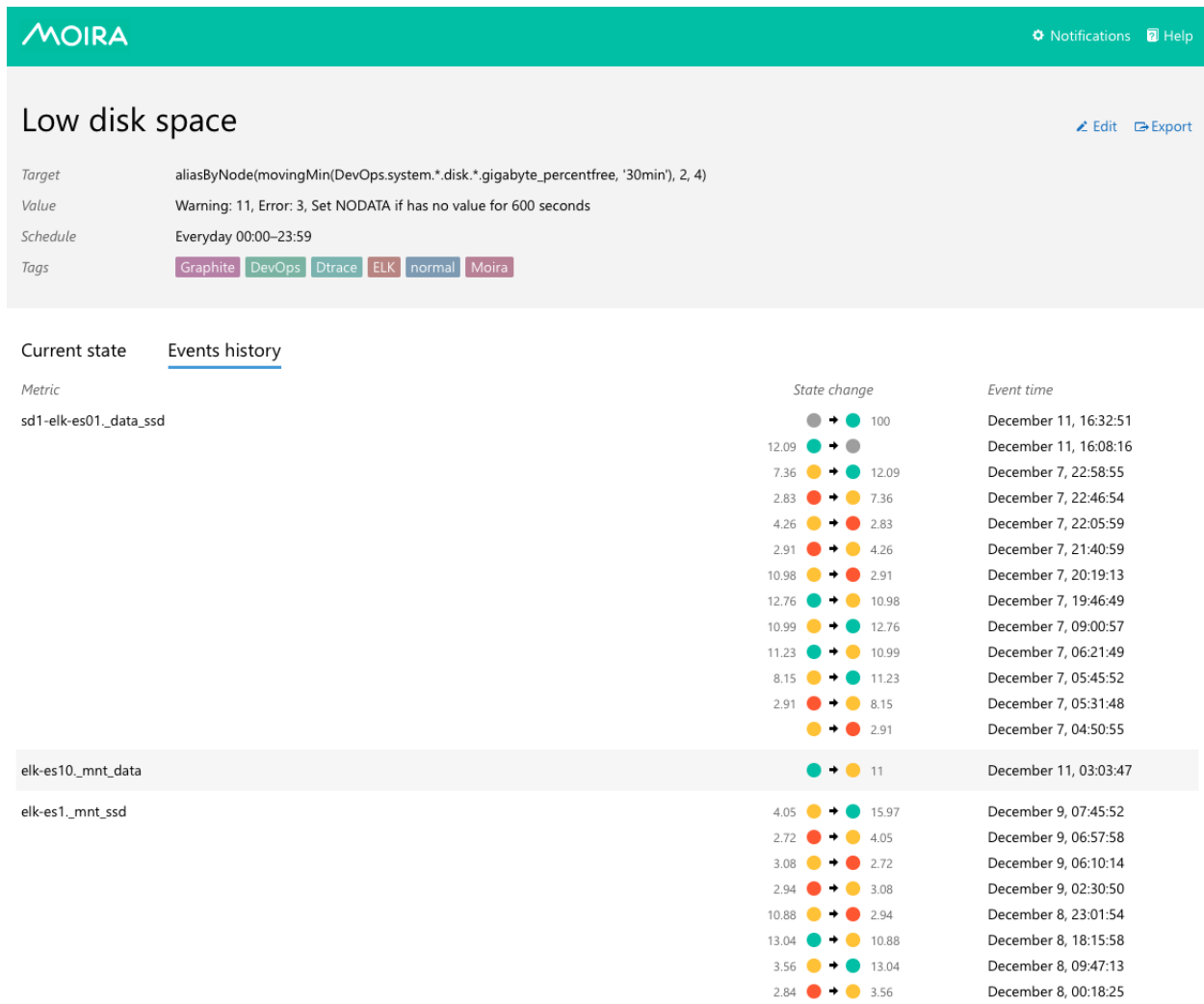
Tags

GraphiteDevOpsDtraceELKnormalMoira

Current state		Events history		
Name		Last event	Value	
●	elk-es9_mnt_data2	November 23, 22:04:01	4.23	Maintenance Del
	bst-elk-es03_data_hdd2	December 1, 09:11:08	4.38	Maintenance Del
	elk-es8_mnt_data3	November 26, 19:18:01	4.59	Maintenance Del
	elk-es1_mnt_data3	November 18, 18:23:23	4.67	Maintenance Del
	bst-elk-es02_data_hdd2	December 7, 13:27:42	4.71	Maintenance Del
	sd1-elk-es01_data_hdd1	November 20, 20:39:11	4.78	Maintenance Del
	elk-es1_mnt_data2	November 18, 19:44:24	4.81	Maintenance Del
	bst-elk-es03_data_hdd1	November 21, 06:26:01	4.87	Maintenance Del
	bst-elk-es02_data_hdd1	November 19, 10:21:08	4.91	Maintenance Del
	elk-es10_mnt_data3	November 26, 17:47:01	4.95	Maintenance Del
●	elk-es8_mnt_data2	November 20, 15:19:02	4.95	Maintenance Del
	sd1-elk-es01_data_hdd2	December 3, 12:35:08	4.97	Maintenance Del
	elk-es10_mnt_data2	November 21, 19:22:01	4.98	Maintenance Del
	elk-es9_mnt_data3	November 26, 12:47:08	4.99	Maintenance Del
	elk-es10_mnt_data	December 11, 03:03:47	10.6	Maintenance Del
	elk-es1_mnt_data	December 2, 14:15:08	11.47	Maintenance Del
	elk-es8_mnt_data	November 26, 13:51:08	13.52	Maintenance Del
	graphite01_mnt_data	December 1, 15:14:08	14.62	Maintenance Del
	graphite02_mnt_data	November 7, 19:06:49	15.7	Maintenance Del
	elk-es8.root	December 8, 20:19:59	23.8	Maintenance Del
●	bst-graphite01.root	November 7, 19:05:43	23.81	Maintenance Del
	elk-es10.root	December 9, 00:41:47	24.02	Maintenance Del

Event History

On this tab 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.



1.4.6 Throttling

Throttling is a distinctive and controversial feature of Maira. 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. Maira 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 a pack of events. Every message contains a warning: “Please, fix your system or tune this trigger to generate less events.”

Maira 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. Obey 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:

The screenshot shows the Maira interface with a 'Low disk space' warning. The warning is displayed as 'WARN' in red. Below the warning, there is a table of metrics. The table has three columns: 'Name', 'Last event', and 'Value'. The 'Name' column lists various disk space metrics for different Elasticsearch nodes. The 'Last event' column shows the date and time of the last event. The 'Value' column shows the percentage of free disk space. A dropdown menu is open for the 'Maintenance' status of the first metric, showing options like 'Off', '15 min', '1 hour', '3 hours', '6 hours', '1 day', and '1 week'.

Name	Last event	Value	Maintenance	Del
elk-es9_mnt_data2	November 23, 22:04:01	4.23	Maintenance	Del
bst-elk-es03_data_hdd2	December 1, 09:11:08	4.38	Off	Del
elk-es8_mnt_data3	November 26, 19:18:01	4.59	15 min	Del
elk-es1_mnt_data3	November 18, 18:23:23	4.67	1 hour	Del
bst-elk-es02_data_hdd2	December 7, 13:27:42	4.71	3 hours	Del
sd1-elk-es01_data_hdd1	November 20, 20:39:11	4.78	6 hours	Del
elk-es1_mnt_data2	November 18, 19:44:24	4.81	1 day	Del
bst-elk-es03_data_hdd1	November 21, 06:26:01	4.87	1 week	Del
sd1-elk-es01_data_hdd2	December 3, 12:35:08	4.87	Maintenance	Del
bst-elk-es02_data_hdd1	November 19, 10:21:08	4.91	Maintenance	Del
elk-es10_mnt_data3	November 26, 17:47:01	4.95	Maintenance	Del
elk-es8_mnt_data2	November 20, 15:19:02	4.95	Maintenance	Del
elk-es10_mnt_data2	November 21, 19:22:01	4.98	Maintenance	Del
elk-es9_mnt_data3	November 26, 12:47:08	4.99	Maintenance	Del
elk-es10_mnt_data	December 11, 03:03:47	10.6	Maintenance	Del

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:

 Notifications Help

BFrontV2 Errors

[✖ Disable throttling](#) [✎ Edit](#) [📄 Export](#)

TargetaliasByNode(movingAverage(transformNull(focus.bfront.*.log.ERROR,0),'5min'),2)

ValueWarning: 50, Error: 100, Set OK if has no value for 600 seconds

ScheduleEveryday 00:00~23:59

Tags

Focus

Current stateEvents history

Name

Last event

Value ↓

● FOCUS100

December 11, 17:44:20

165.4Maintenance ▾🗑 Del

● FOCUS200

December 11, 17:43:20

171.2Maintenance ▾🗑 Del

● focus300


December 11, 17:44:20

182.8Maintenance ▾🗑 Del


4. Entirely disable throttling for a subscription. *This is not recommended*, unless you really know what you are doing:

Edit subscription ✕

Target delivery channels:

 @alexkir ✕

Select delivery channel ▼

Tags: 



DevOps ✕ critical ✕

Delivery schedule:

☒ Mon ☒ Tue ☒ Wed ☒ Thu ☒ Fri ☒ Sat ☒ Sun

☒ All day ☐ At specific interval

00:00 — 23:59

☒ Throttle messages  

☐ Enabled

Update

Update and test

Delete

Cancel

1.4.7 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 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:

☒ Watch for value falling:

☐ WARN if T1 <=

☐ ERROR if T1 <=

☐ NODATA if has no value for

When Maira 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 Maira 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.

1.4.8 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 Maira's Advanced Mode is better:

MOIRA

Notifications
Help

Add trigger

Name

Request processing takes too long

Description

Target

T1

Nginx.requests.process_time.p95

—

T2

Nginx.requests.count

—

+ Add one more

Simple mode

Advanced mode

Expression

(t1 > 1000 && t2 > 100) ? WARN : OK

?

NODATA

if has no value for

600

Watch time

☒ Mon
☒ Tue
☒ Wed
☒ Thu
☒ Fri
☒ Sat
☒ Sun

☒ All day
☐ At specific interval

00:00 — 23:59

You can use any [govaluate](#) 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.

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

1.4.9 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 notifier yaml config.

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`.

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't delete a tag if there is at least one trigger that uses it. You **can** delete a tag that is used in a subscription.

Patterns

You can see a list of all Graphite patterns with links to corresponding triggers and list of all matching metrics at `/patterns`.

1.5 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, etc.

1.5.1 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.
- Description. Longer text that gets included in notification to delivery channels that support long texts.
- 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

Filter and Check Incoming Metrics

When user adds a new trigger, Moira parses patterns from targets and saves them to `moira-pattern-list` key in Redis. Filter rereads this list every second. When a metric value arrives, Filter 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 of incoming metric value that should be checked as soon as possible.

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

Trigger 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 a trigger contains *all* of the tags in a subscription, and only in this case, 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 differs 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.

1.5.2 UI

UI is a static web application built with [RetailUI](#) based on [React](#).

Install dependencies.

```
npm install
```

Run webpack dev server at <http://localhost:9000>.

```
npm start
```

Note: UI doesn't work without running API microservice.

1.5.3 Backend

Backend microservices are written in [Go](#). To run tests, first get all dependencies.

```
go get github.com/kardianos/govendor
govendor sync
```

Then, run [GoConvey](#) tests.

```
go get github.com/smartystreets/goconvey
goconvey
```

Writing Your Own Notification Sender

First, look at built-in senders:

- `senders/slack`
- `senders/pushover`
- `senders/mail`

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

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

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

1.6 Contact Moira Developers

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

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 every minute making lots of ineffective queries and overloading your cluster. Moira relies on the incoming metric stream, and has its own fast cache for recent data.

- **Support for (almost) all Graphite functions**

Graphite function library ([carbonapi](#)) is embedded directly into Moira source code. You can use any function and get predictable results, like in your Graphite or Grafana dashboards.

- **Support for custom expressions**

If simple warning/error threshold is not enough, you can write flexible [govaluate](#) 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 supports email, [Slack](#), [Pushover](#) and many other channels of notification 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 over 10 messages per 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 Filter

Filter is a lightweight service responsible for receiving lots of metric data in Graphite format. It 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 an 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 an 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 an application that serves as a backend for UI.

2.3.5 Web 2.0

Web 2.0 is a frontend React application, it looks like this:

MOIRA

[Notifications](#)
[Help](#)

DevOps

dbaas

Only Problems

Add Trigger

1

Cassandra GC metrics missing

normal

DevOps

dbaas

Cassandra

exclude(aliasByNode((Alko,EDI,EDITest,KE,KE-cloud,KE-dev),Cassandra.*,*.GC.StopTheWorld.sum, 0, 2, 3), 'EDI.LegacyCluster')

18

EDI Cassandra Data Disk Space Free

DevOps

dbaas

EDI

Cassandra

normal

aliasByNode(exclude(exclude(EDI.system.*,disk_storage*.gigabyte_percentfree, 'elastic'), 'edi14'), 2, 4)

2

EDI Test Cassandra Nodes Down

DevOps

dbaas

EDI

Cassandra

normal

exclude(exclude(groupByNode(EDITest.Cassandra.*,*.DownEndpointCount.DownEndpointCount, 2, 'maxSeries'), 'CatalogueRtqBenchmarkCluster'), 'EdiRtqLoadCluster')

Name	Last event	Value ↓		
EdiStagingCluster	December 4, 18:36:16	0	Maintenance	Del
EdiTestingCluster	December 4, 18:36:16	0	Maintenance	Del

1

KE Cassandras Read Latency

DevOps

dbaas

Cassandra

normal

groupByNode(movingMin(KE.Cassandra.*,*.ClientRequest.Read.Latency.99thPercentile,'5min'),2,'maxSeries')

30

Alko Cassandra Data Disk Space Free

DevOps

dbaas

Cassandra

normal

Alko

aliasByNode(Alko.system.*,disk_storage*.gigabyte_percentfree, 2, 4)