ZMON Documentation

Release 2.0

Henning Jacobs

Jan 22, 2018

Contents

CHAPTER 1

Introduction

ZMON is a flexible and extensible open-source platform monitoring tool developed at Zalando and is in production use since early 2014. It offers proven scaling with its distributed nature and fast storage with KairosDB on top of Cassandra. ZMON splits checking(data acquisition) from the alerting responsibilities and uses abstract entities to describe what's being monitored. Its checks and alerts rely on Python expressions, giving the user a lot of power and connectivity. Besides the UI it provides RESTful APIs to manage and configure most properties automatically.

Anyone can use ZMON, but offers particular advantages for technical organizations with many autonomous teams. Its front end (see Demo / Bootstrap / Kubernetes/ Vagrant) comes with Grafana3 "built-in," enabling teams to create and manage their own data-driven dashboards along side ZMON's own team/personal dashboards for alerts and custom widgets. Being able to inherit and clone alerts makes it easier for teams to reuse and share code. Alerts can trigger HipChat, Slack, and E-Mail notifications. iOS and Android clients are works in progress, but push notifications are already implemented.

ZMON also enables painless integration with CMDBs and deployment tools. It also supports service discovery via custom adapters or its built-in entity service's REST API. For an example, see zmon-aws-agent to learn how we connect AWS service discovery with our monitoring in the cloud.

Feel free to contact us via slack.zmon.io.

1.1 ZMON Components

A minimum ZMON setup requires these four components:

- zmon-controller: UI/Grafana/Oauth2 Login/Github Login
- zmon-scheduler: Scheduling check/alert evaluation
- zmon-worker: Doing the heavy lifting
- zmon-eventlog-service: History for state changes and modifications

Plus the storage covered in the *Requirements* section.

The following components are optional:

- zmon-cli: A command line client for managing entities/checks/alerts if needed
- zmon-aws-agent: Works with the AWS API to retrieve "known" applications
- zmon-data-service: API for multi DC federation: receiver for remote workers primarily
- zmon-metric-cache: Small scale special purpose metric store for API metrics in ZMON's cloud UI
- zmon-notification-service: Provides mobile API and push notification support for GCM to Android/iOS app
- zmon-android: An Android client for ZMON monitoring
- zmon-ios: An iOS client for ZMON monitoring

1.2 ZMON Origins

ZMON was born in late 2013 during Zalando's annual Hack Week, when a group of Zalando engineers aimed to develop a replacement for ICINGA. Scalability, manageability and flexibility were all critical, as Zalando's small teams needed to be able to monitor their services independent of each other. In early 2014, Zalando teams began migrating all checks to ZMON, which continues to serve Zalando Tech.

1.3 Entities

ZMON uses entities to describe your infrastructure or platform, and to bind check variables to fixed values.

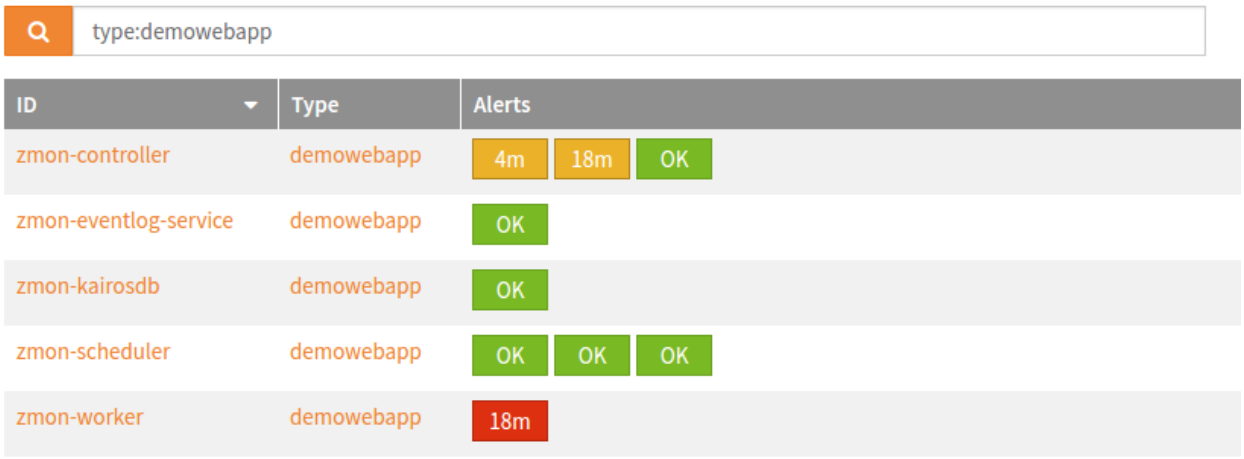
```
{
      "type":"host",
      "id":"cassandra01",
      "host":"cassandra01",
      "role":"cassandra-host",
      "ip":"192.168.1.17",
      "dc":"data-center-1"
}
```

Or more abstract objects:

```
{
      "type":"postgresql-cluster",
      "id":"article-cluster",
      "name":"article-cluster",
      "shards": {
              "shard1":"articledb01:5432/shard1",
              "shard2":"articledb02:5432/shard2"
      }
}
```

Entity properties are not defined in any schema, so you can add properties as you see fit. This enables finer-grained filtering or selection of entities later on. As an example, host entities can include a physical model to later select the proper hardware checks.

Below you see an exmple of the entity view with alerts per entity.



1.4 Checks

A check describes how data is acquired. Its key properties are: a command to execute and an entity filter. The filter selects a subset of entities by requiring an overlap on specified properties. An example:

```
{
  "type":"postgresql-cluster", "name":"article-cluster"
}
```

The check command itself is an executable Python expression. ZMON provides many custom wrappers that bind to the selected entity. The following example uses a PostgreSQL wrapper to execute a query on every shard defined above:

```
# sql() in this context is aware of the "shards" property

sql().execute('SELECT count(1) FROM articles "total"').result()
```

A check command always returns a value to the alert. This can be of any Python type.

Not familiar with Python's functional expressions? No worries: ZMON allows you to define a top-level function and define your command in an easier, less functional way:

```
def check():
  # sql() binds to the entity used and thus knows the connection URLs
  return sql().execute('SELECT count(1) FROM articles "total"').result()
```

1.5 Alerts

A basic alert consists of an alert condition, an entity filter, and a team. An alert has only two states: up or down. An alert is up if it yields anything but False; this also includes exceptions thrown during evaluation of the check or alert, e.g. in the event of connection problems. ZMON does not support levels of criticality, or something like "unknown", but you have a color option to customize sort and style on your dashboard (red, orange, yellow).

Let's revisit the above PostgreSQL check again. The alert below would either popup if there are no articles found or if we get an exception connecting to the PostgreSQL database.

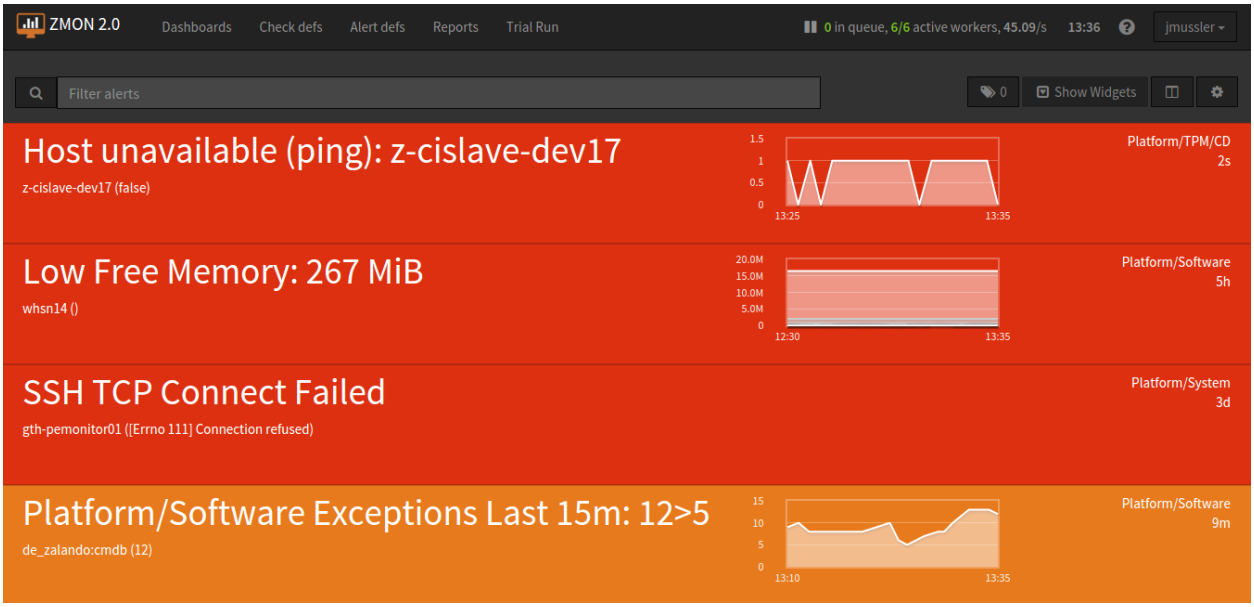
```
team: database
entities:
  - type: postgresql-cluster
alert_condition: |
  value <= 0
```

Alerts raised by exceptions are marked in the dashboard with a "!".

Via ZMON's UI, alerts support parameters to the alert condition. This makes it easy for teams/users to implement different thresholds, and — with the priority field defining the dashboard color — render their dashboards to reflect their priorities.

1.6 Dashboards

Dashboards include a widget area where you can render important data with charts, gauges, or plain text. Another section features rendering of all active alerts for the team filter, defined at the dashboard level. Using the team filter, select the alerts you want your dashboard to include. Specify multiple teams, if necessary. TAGs are supported to subselect topics.



1.7 REST API and CLI

To make your life easier, ZMON's REST API manages all the essential moving parts to support your daily work — creating and updating entities to allow for sync-up with your existing infrastructure. When you create and modify checks and alerts, the scheduler will quickly pick up these changes so you won't have to restart or deploy anything.

And ZMON's command line client - a slim wrapper around the REST API - also adds usability by making it simpler to work with YAML files or push collections of entities.

1.8 Development Status

The team behind ZMON continues to improve performance and functionality. Please let us know via GitHub's issues tracker if you find any bugs or issues.

CHAPTER 2

Getting Started

To quickly get started with ZMON, use the preconfigured Vagrant box featured on the main ZMON repository. Make sure you've installed Vagrant *(at least 1.7.4)* and a Vagrant provider like VirtualBox on your machine. Clone the repository with Git:

```
$ git clone https://github.com/zalando/zmon.git
$ cd zmon/
```

From within the cloned repository, run:

```
$ vagrant up
```

Bootstrapping the image for the first time will take a bit of time. You might want to grab some coffee while you wait. :)

When it's finally up, Vagrant will report on how to reach the ZMON web interface:

```
==> default: ZMON installation is done!
==> default: Goto: https://localhost:8443
==> default: Login with your GitHub credentials
```

2.1 Creating Your First Alert

2.1.1 Log In

Open your web browser and navigate to the URL reported by Vagrant: e.g. https://localhost:8443/. Click on *Sign In*. This will redirect you to Github where you sign in and authorize the ZMON app. Then it takes you back and you are logged in.

Note: For your own deployment create your own app in Github with your redirect URL. In ZMON you can then limit users allowed access to your Github organization.

2.1.2 Checks and Alerts

An alert shown on ZMON's dashboard typically consists of two parts: the *check-definition*, which is responsible for fetching the underlying data; and the *alert-definition*, which defines the condition under which the alert will trigger. Multiple alerts with different alert conditions can operate on the same check, fetching data only once.

Let's explore this concept now by creating a simple check and defining some alerts on it.

2.1.3 Create a new Check

One way to create a new check from scratch is via the *Using the CLI*. A more convenient way, however, is to use the "Trial Run" feature. It enables you to develop checks and alerts, execute them immediately, and inspect the result. Once you are happy with your check command and filter, you can save it from the Trial Run directly. Some users prefer to download the YAML definition from there to store and maintain it in Git.

2.1.4 Create an Alert

In the top navigation of ZMON's web interface, select Check defs from the list and click on *Website HTTP status*. Then click *"Add New Alert Definition"* to create a new alert for this particular check. Fill out the form (see example values below), and hit *"Save"*:

Name	Oops . . . website is gone!
Description	Website was not reachable.
Priority	Priority 1 (red)
Alert Condition	value != 200
Team	Team 1
Responsible Team	Team 1
Status	ACTIVE

After you hit save, it will take a few seconds until it is picked up and executed.

2.1.5 View Dashboard

If the alerts condition evaluates to anything but `False` the alert will appear on the dashboard. This means not only for `True`, but also e.g. in case of exceptions triggered, e.g. due to timeouts or failure to connect. Currently there's only one dashboard, and it is configured to show all present alerts. To view the dashboard, select Dashboards from the main menu and click on *Example Dashboard*.

To see the alert, you must simulate the error condition; try modifying its condition or the check-definition to return an error code). You do this, set the URL in the check command to http://httpstat.us/500. (The number in the URL represents the HTTP error code you will get.)

To see the actual error code in the alert, you might want to create/modify it like this:

Name	Website gone with status {code}
Description	Website was not reachable.
Priority	Priority 1 (red)
Alert Condition	capture(code=value)!=200
Team	Team 1
Responsible Team	Team 1
Status	ACTIVE

2.2 Using the CLI

The ZMON Vagrant box comes preinstalled with *zmon-cli*. To use the CLI, log in to the running Vagrant box with:

```
$ vagrant ssh
```

The Vagrant box also contains some sample yaml files for creating entities, checks and alerts. You can find these in */vagrant/examples*.

As an example of using ZMON's CLI, let's create a check to verify that google.com is reachable. *cd* to */vagrant/examples/check-definitions* and, using zmon-cli, create a new check-definition:

```
$ cd /vagrant/examples/check-definitions
$ zmon check-definitions init website-availability.yaml
$ vim website-availability.yaml
```

Edit the newly created *website-availability.yaml* to contain the following code. (type `i` for insert-mode)

```
name: "Website HTTP status"
owning_team: "Team 1"
command: http("http://httpstat.us/200", timeout=5).code()
description: "Returns current http status code for Website"
interval: 60
entities:
 - type: GLOBAL
status: ACTIVE
```

Type `ESC :wq RETURN` to save the file.

To push the updated check definition to ZMON, run:

```
$ zmon check-definitions update website-availability.yaml
Updating check definition... http://localhost:8080/#/check-definitions/view/2
```

Find more detailed information here: *Command Line Client*.

CHAPTER 3

Entities

Entities describe what you want to monitor in your infrastructure. This can be as basic as a host, with its attributes hostname and IP; or something more complex, like a PostgreSQL sharded cluster with its identifier and set of connection strings.

ZMON gives you two options for autonomation in/integration with your platform: storing entities via zmon-controller's entity service, or discovering them via the adapters in zmon-scheduler. At Zalando we use both, connecting ZMON to tools like our CMDB but also pushing entities via REST API.

ZMON's entity service describes entities with a single JSON document. Any entity must contain an ID that is unique within your ZMON deployment. We often use a pattern like <hostname>(:<port>) to create uniqueness at the host and application levels, but this is up to you.

At the check execution we bind entity properties as default values to the functions executed, e.g. the IP gets used for relative `http()` requests.

3.1 Format

Generally, ZMON entity is a set of properties that can be represented as a multi-level dictionary. For example:

```
{
    "id":"arbitrary_entity_id",
    "type":"some_type",
    "oneMoreProperty":"foo",
    "nestedProperty": {
        "subProperty1": "foo",
        "subProperty2": "bar",
    }
}
```

2 notes here to keep in mind:

1. `id` and `type` properties are **mandatory**.
2. ZMON filtering (e.g. in ZMON UI) **does not support nested properties**.

3.2 Examples

In working with the Vagrant Box, you can use the scheduler instance entity like this:

```
{
    "id":"localhost:3421",
    "type":"instance",
    "host":"localhost",
    "project":"zmon-scheduler-ng",
    "ports": {"3421":3421}
}
```

Here, you can use the "ports" dictionary to also describe additional open ports. As with Spring Boot, a second port is usually added, exposing management features.

Now let's look at an example of the PostgreSQL instance:

```
{
    "id":"localhost:5432",
    "type":"database",
    "name":"zmon-cluster",
    "shards": {"zmon":"localhost:5432/local_zmon_db"}
}
```

Usage of the property "shards" is given by how ZMON's worker exposes PostgreSQL clusters to the sql() function.

View more examples here.

If you'd like to create an entity by yourself, check ZMON CLI tool

CHAPTER 4

Check Definitions

Checks are ZMON's way of gathering data from arbitrary entities, e.g. databases, micro services, hosts and more. Create them as describe below using either the UI or the CLI.

4.1 Key properties

4.1.1 Command

The command is being executed by the worker and is considered the data gathering part. It is executed once per selected entity and its result made available to all attached alerts. You have different wrappers at hand and the `entity` variable is also available for access.

4.1.2 Entity Filter

Select the entities you want the check to execute against in general, often only a type filter is applied, sometimes more specific. The alert allows you to do more fine grained filtering. This proves useful to allow checks to be easily reused.

4.1.3 Interval

Specify the interval in seconds at which you want the check to be executed.

4.1.4 Owning team

This is the team originally creating the check, right now this has little effect.

4.2 Creating new checks

4.2.1 Using trial run

4.2.2 Using the CLI

```
$ zmon check init new-check.yaml
$ zmon check update new-check.yaml
```

CHAPTER 5

Alert Definitions

Alert definitions specify when (condition, time period) and who (team) to notify for a desired monitoring event. Alert definitions can be defined in the ZMON web frontend and via the *ZMON CLI*.

The following fields exist for alert definitions:

name The alert's display name on the dashboard. This field can contain curly-brace variables like `{mycapture}` that are replaced by capture's value when the alert is triggered. It's also possible to format decimal precision (e.g. "My alert `{mycapture:.2f}`" would show as "My alert `123.45`" if mycapture is `123.456789`). To include a comma separated list of entities as part of the alert's name, just use the special placeholder `{entities}`.

description Meaningful text for people trying to handle the alert, e.g. incident support.

priority The alert's dashboard priority. This defines color and sort order on the dashboard.

condition Valid Python expression to return true when alert should be triggered.

parameters You may apply parameters your alert condition using variables. More details *here*

entities filter Additional filter to apply the alert definition only to a subset of entities.

notifications List of notification commands, e.g. to send out emails.

time_period Notification time period.

team Team dashboard to show alert on.

responsible_team Additional team field to allow delegating alert monitoring to other teams. The responsible team's name will be shown on the dashboard.

status Alerts will only be triggered if status is "ACTIVE".

template A template is an alert definition that is not evaluated and can only be used for extension. More details *here*

5.1 Condition

Simple expressions can start directly with an operator. To trigger an alert if the check result value is larger than zero:

```
> 0
```

You can use the `value` variable to create more complex conditions:

```
value >= 10 and value <= 100
```

Some more examples of valid conditions:

```
== 'OK'
!= False
value in ('banana', 'apple')
```

If the value already is a dictionary (hash map), we can apply all the Python magic to it:

```
['mykey'] > 100                                    # check a specific dict value
'error-message' in value                           # trigger alert if key is␣
↪present
not empty([ k for k, v in value.items() if v > 100 ]) # trigger alert if some dict␣
↪value is > 100
```

5.2 Captures

You can capture intermediate results in alert conditions by using the `capture` function. This allows easier debugging of complex alert conditions.

```
capture(value["a"]/value["b"]) > 0
capture(myval=value["a"]/value["b"]) > 0
any([capture(foo=FOO) > 10, capture(bar=BAR) > 10])
```

Please refer to Recipes section in *Python Tutorial* for some Python tricks you may use.

Named captures can be used to customize the alert display on the *dashboard* by using template substitution in the alert name.

If you call your capture *dashboard*, it will be used on dashboard next to entity name instead of entity value. For example, if you have a host-based alert that fails on z-host1 and z-host2, you would normally see something like that

ALERT TITLE (N) z-host1 (value1), z-host2 (value2)

Once you introduce capture called *dashboard*, you will get something like

ALERT TITLE (N) z-host1 (capturevalue1), z-host2 (capturevalue2)

where capturevalue1 is value of "dashboard" capture evaluated against z-host1.

Example alert condition (based on PF/System check for diskspace)

```
"ERROR" not in value
and
capture(dashboard=(lambda d: '{}:{}'.format(d.keys()[0], d[d.keys()[0]]['percentage_
↪space_used']) if d else d)(dict((k, v) for k,v in value.iteritems() if v.get(
↪'percentage_space_used', 0) >= 90))))
```

5.3 Entity (Exclude) Filter

The *check definition* already defines on what entities the checks should run. Usually the check definition's `entities` are broader than you want. A diskspace check might be defined for all hosts, but you want to trigger alerts only for hosts you are interested in. The alert definition's `entities` field allows to filter entities by their attributes.

See *Entities* for details on supported entities and their attributes.

Note: The entity name can be included in the alert message by using a special placeholder *{entities}* on the alert name.

5.4 Notifications

ZMON notifications lets you know when you have a new alert without check the web UI. This section will explain how to use the different options available to notify about changes in alert states. We support E-Mail, HipChat, Slack and one SMS provider that we have been using.

The notifications field is a list of function calls (see below for examples), calling one of the following methods of notification:

send_email (*email**[, *subject*, *message*, *repeat*])

send_sms (*number**[, *message*, *repeat*])

send_push ([*message*, *repeat*, *url*, *key*])

send_slack ([*channel*, *message*, *repeat*, *token*])

send_hipchat ([*room*, *message*, *color='red'*, *repeat*, *token*, *notify=False*])

If the alert has the top priority and should be handled immediately, you can specify the repeat interval for each notification. In this case, you will be notified periodically, according to the specified interval, while the alert persists. The interval is specified in seconds.

To receive push notifications you need one of the ZMON mobile apps (configured for your deployment) and subscribe to alert ids, before you can receive notifications.

In addition, you may use notification-groups to configure groups of people with associated **emails** and/or **phone numbers** and use these groups in notifications like this:

Example JSON email and SMS configuration using groups:

```
[
   "send_sms('active:2nd-database')",
   "send_email('group:2nd-database')"
]
```

In the above example you send SMS to **active** member of **2nd-database** group and send email to **all members** of the group.

Example JSON email configuration:

```
[
   "send_mail('a@example.org', 'b@example.org')",
   "send_mail('a@example.com', 'b@example.com', subject='Critical Alert please do␣
↪something!')",
   "send_mail('c@example.com', repeat=60)"
]
```

Example JSON Slack configuration:

```
[
   "send_slack()",
   "send_slack(channel='#incidents')",
   "send_slack(channel='#incidents', token='your-token')"
]
```

Example JSON HipChat configuration:

```
[
   "send_hipchat()",
   "send_hipchat(room='#incidents', color='red')",
   "send_hipchat(room='#incidents', token='your-token')",
   "send_hipchat(room='#incidents', token='your-token', notify=True)"
]
```

Example JSON Push configuration:

```
[
   "send_push()"
]
```

Example JSON SMS configuration:

```
[
   "send_sms('0049123555555', '0123111111')",
   "send_sms('0049123555555', '0123111111', message='Critical Alert please do␣
↪something!')",
   "send_sms('0029123555556', repeat=300)"
]
```

Example email:

```
From: ZMON <zmon@example.com>
Date: 2014-05-28 18:37 GMT+01:00
Subject: NEW ALERT: Low Orders/m: 84.9% of last weeks on GLOBAL
To: Undisclosed Recipients <zmon@example.com>

New alert on GLOBAL: Low Orders/m: {percentage_wow:.1f}% of last weeks


Current value: {'2w_ago': 188.8, 'now': 180.8, '1w_ago': 186.6, '3w_ago': 196.4, '4w_
↪ago': 208.8}


Captures:

percentage_wow: 184.9185496584

last_weeks_avg: 195.15



Alert Definition
Name (ID):      Low Orders/m: {percentage_wow:.1f}% of last weeks (ID: 190)
Priority:       1
Check ID:       203
Condition       capture(percentage_wow=100. * value['now']/capture(last_weeks_
↪avg=(value['1w_ago'] + value['2w_ago'] + value['3w_ago'] + value['4w_ago'])/4. )) <␣
↪85
```

```
Team:          Platform/Software
Resp. Team:    Platform/Software
Notifications: [u"send_mail('example@example.com')"]

Entity

id: GLOBAL

type: GLOBAL

percentage_wow: 184.9185496584

last_weeks_avg: 195.15
```

Example SMS:

```
Message details:
   Type: Text Message
   From: zmon2
Message text:
   NEW ALERT: DB instances test alert on all shards on customer-integration-master
```

5.5 Time periods

ZMON 2.0 allows specifying time periods in alert definitions. When specified, user will be notified about the alert only when it occurs during given period. Examples below cover most common use cases of time periods' definitions.

To specify a time period from Monday through Friday, 9:00 to 17:00, use a period such as

> wd {Mon-Fri} hr {9-16}

When specifying a range by using -, it is best to think of - as meaning through. It is 9:00 through 16:00, which is just before 17:00 (16:59:59).

To specify a time period from Monday through Friday, 9:00 to 17:00 on Monday, Wednesday, and Friday, and 9:00 to 15:00 on Tuesday and Thursday, use a period such as

> wd {Mon Wed Fri} hr {9-16}, wd{Tue Thu} hr {9-14}

To specify a time period that extends Mon-Fri 9-16, but alternates weeks in a month, use a period such as

> wk {1 3 5} wd {Mon Wed Fri} hr {9-16}

A period that specifies winter in the northern hemisphere:

> mo {Nov-Feb}

This is equivalent to the previous example:

> mo {Jan-Feb Nov-Dec}

As is

> mo {jan feb nov dec}

And this is too:

> mo {Jan Feb}, mo {Nov Dec}

To specify a period that describes every other half-hour, use something like:

minute { 0-29 }

To specify the morning, use

hour { 0-11 }

Remember, 11 is not 11:00:00, but rather 11:00:00 - 11:59:59.

5 second blocks:

sec {0-4 10-14 20-24 30-34 40-44 50-54}

To specify every first half-hour on alternating week days, and the second half-hour the rest of the week, use the period

wd {1 3 5 7} min {0-29}, wd {2 4 6} min {30-59}

For more examples and syntax reference, please refer to this documentation, note that suffixes like *am* or *pm* for hours are **not** supported, only integers between 0 and 23. In doubt, try calling with python with your period definition like

```
from timeperiod import in_period
in_period('hr { 0 - 23 }')
```

This should not throw an exception. The timeperiod module in use is timeperiod2. The *in_period* function accepts a second parameter which is a datetime like

```
from datetime import datetime
from timeperiod import in_period
in_period('hr { 7 - 23 }', datetime(2018, 1, 8, 2, 15)) # check 2018-01-08 02:15:00
```

5.6 Alert Definition Inheritance

Alert definition *inheritance* allows one to create an alert definition based on another alert whereby a child reuses attributes from the parent. Each alert definition can only inherit from a single alert definition (`single inheritance`).

5.6.1 Template

A Template is basically an alert definition with a subset of attributes that **is not evaluated and can only be used for extension**.

To create a template:

1. Select the check definition
2. click **Add New Alert Definition**
3. Set attributes to reuse and activate checkbox `template`

5.6.2 Extending

In general one can inherit from any alert definition/template. One should open the alert definition details and click `inherit` on the top right corner. To override a field, just type in a new value. An icon should appear on the left side, meaning that the field will be overridden. To rollback the change and keep the value defined on the parent, one should click in `override` icon.

5.6.3 Overriding

By default the child alert retains all attributes of the parent alert with the exception of the following mandatory attributes:

- team
- responsible team
- status

These attributes are used for `authorization` (see permissions for details) therefore, they cannot be reused. If one changes these attributes on the parent alert definition, child alerts are not affected and you don't loose access rights. All the remaining attributes can be overridden, replacing the parent alert definition with its own values.

5.7 Alert Definition Parameters

Alert definition *parameters* allows one to decouple alert condition from constants that are used inside it.

5.7.1 Use Case: Technical alert condition

If your alert condition is highly technical with a lot of Python code in it, it is often makes sense to split actual calculation from threshold values and move such constant values into parameters.

The same may apply in certain cases to alert definitions created by technical staff, which later need to be adjusted by non-technical people - if you split calculation from variable definition, you may let non-technical people just change values without touching calculation logic.

5.7.2 Use Case: Same alert, different priorities

Another use case where we recommend to use parameters is when you need to have the same alert come up with a different priority depending on threshold values.

In such case, refer to *alert inheritance* for configuring inherited alerts.

Proposed structure would look like:

- Base alert "A" with alert condition and parameters, check *template* box
- Alert "B1" inherits from "A" specifying *priority* RED and associated parameter values
- Alert "B2" inherits from "A" specifying *priority* YELLOW and associated parameter values

5.7.3 An example: Setting a simple parameter in trial run

In the zmon2 web interface click on the trial run button.

1. In the **Check Command** text box enter:

```
normalvariate(50, 20)
```

This is a simple normal probability function that produce a float number 50% of the time over 50.0, so it's good to test things.

2. In the **Alert Condition** enter:

```
value>capture(threshold=threshold) + len(capture(params=params))
```

3. In the **Parameters** selector enter two values (by clicking the plus sign):

Name	Value	Type
threshold	50.0	Float
anything	Kartoffel	String

4. In the **Entity Filter** text box enter:

```
[
    {
        "type": "GLOBAL"
    }
]
```

5. In the **Interval** enter: 10

If you run this Trial you can get an Alert or an 'OK', but the interesting thing will be in the **Captures** column. See how the parameters that you entered are evaluated in the alert condition with the value that you provided. Notice also that there is a special parameter called **params** that holds a dict with all the parameters that you entered, this is done so the user can iterate over all the parameters and take conditional decisions, providing a kind of introspection capability, but this is only for advanced users.

Last but not least: *Most of the time you don't need to capture the parameter values*, we did it like this so you can visually see that the parameters are evaluated, this means that you can run exactly the same check with this **Alert Condition**:

```
value>threshold + len(params)
```

5.8 Downtimes

This functionality allows the user to acknowledge an existing alert or create a downtime schedule for an anticipated service interruption. When acknowleding an existing alert, the user has to provide the predicted duration, and when creating a scheduled downtime - start and end date. If the downtime is currently active, meaning an alert occured within the downtime period, the alert notification won't be shown in the dashboard and it'll be greyed out in alert details page. Please note that the downtime will not be evaluated immediately after creation, meaning that the alert might appear as active until it's evaluated again by the worker. E.g. if the user defined a downtime for an alert which is evaluated every minute and the last evaluation was 5 seconds ago, it would take approximately one more minute for the alert to appear in "downtime state".

To acknowledge an alert or to schedule a new downtime, the user has to go to the specific alert details page and click on a downtime button next to the desired alert.

5.9 Comments

Comments are useful in providing additional information to other members of your team (or other teams) about your alerts. Those with ADMIN, LEAD and USER roles can add comments to an alert, but VIEWERS can not. ADMINs can delete either their own or other people's comments. USERs can delete only their own comments.

5.9.1 Adding Comments

Follow these steps:

- Open the alert definition where you want to add your comment.
- Either click on the top-right link *Comments* to add a **general** comment (for all entities), or click on the balloon on the left side of the entity name to add a comment on a **specific** entity.
- In the comments window, type your comment. Use as many lines as you need.
- Click the *Post comment* button and save your comment. Done!

5.9.2 Seeing Existing Comments

It's easy: Just open the alert definition, then click on *Comments* (top-right link).

5.9.3 Deleting Comments

Deleting is also easy: Open the alert definition, click on the top right-link *Comments*, click on the cross above the comment, and delete.

CHAPTER 6

Dashboards

ZMON's customizable dashboards enable you to configure widgets and choose which alerts to show. Dashboards have the following fields:

name The dashboard's name. This is mainly used to identify the dashboard.

default view The dashboard default view. Here you can specify the default rendering behavior when you open the dashboard. There are two options available:

- Full: Provides detailed information about the alert. Useful when using big screens.
- Compact: Only displays the alert message. Useful for small screens.

Note: You can toggle the view in the dashboard by clicking on the top right button of the alert container.

edit mode Here you can specify who can modify your dashboard. There are three options available:

- Private: Only you (and the admin) can edit the dashboard
- Team: All members of your team(s) can edit the dashboard
- Public: Everyone can edit the dashboard

widget configuration The widget configuration defines the different widgets that the current dashboard has. An example of a valid widget configuration is the following:

```
[
    {
        "checkDefinitionId": 1,
        "entityId": "GLOBAL",
        "type": "gauge",
        "title": "Order Failure %",
        "options": {
            "max": 35
        }
    },
    {
        "checkDefinitionId": 4,
        "entityId": "GLOBAL",
        "type": "gauge",
        "title": "Random",
        "options": {
            "max": 100
        }
    },
    {
        "checkDefinitionId": 5,
        "entityId": "my_db_name-live",
        "type": "value",
        "title": "My database value"
    }
]
```

Supported widget types are:

- gauge
- chart
- value
- networkmap
- iframe

In order to edit a specific dashboard, go to the dashboard tab, and click the edit button. To set it as active, just click on its name.

In order to be able to create or edit a new dashboard, user should be logged in. Unless you have the admin role, you will only be able to edit the dashboards you created.

Widgets will automatically spread out across the whole width, i.e. if you define two widgets both will take about 50% of screen width.

alert teams Here you can specify a list of patterns to filter alerts by team or responsible team you want to display (wildcards using * are allowed)

Example: All incident alerts (including sub-teams)

```
[
 "Incident*"
]
```

6.1 value, gauge, chart, trend

The value widget will show the check value with a big font. The gauge will show a gauge from "min" to "max". The chart will show the history of check values. The trend will show a trend arrow (going up or down).

These widgets expect a "checkDefinitionId", "entityId" and "title" properties:

- "checkDefinitionId" - self-explanatory. Data in widget is based on check results
- "entityId" - if your check is based on GLOBAL, leave "GLOBAL", otherwise specify name of entity (as it appears in alert details) that you will use to get the data from (as check returns one result for each entity).
- "title" - text displayed in the top part of the widget.

For chart widgets, instead of using "checkDefinitionId" + "entityId", you can also define the data to be shown *using a KairosDB query*.

They'll share the full screen width unless you set the "width" property, ranging from 12 (full width, calculated in "columns", see Bootstrap) to 2 (smallest meaningful) or even 1.

Configuration options can be defined inside an "options" property. Each widget accepts a different set of options.

Value widgets accept "fontSize", "color" and "format" properties. Additionally you can set a specific JSON value of the check result to be displayed by using the "jsonPath" property, in case the result is a JSON object instead of a string / number.

A font size can be specified with the "fontSize" property, with numbers (in pixels) for the desired size.

A color for the font can be specified with the "color" property.

A formatting string can be also specified to make python-like string interpolation and floating point precision rounding, by defining a "format" property in the options object. Syntax of the format string is mostly same as in python.

Options example for all widgets to specify which value from the check result to be displayed using "jsonPath":

```
"options": {
    "fontSize": 120,     # set font size to 120px,
    "color": "red",      # set color to red (also accepts #FF0000).
    "format": ".3f"      # show value with 3 places of floating point precision
},
"jsonPath": ".cpu.load1"
```

Check the documentation of JSONPath for more info on how to use the jsonPath property. Please note that you don't need to use the $ symbol, as it's prepended automatically on parsing.

Charts can be configured by defining an "options" property. All options available to Flot charts can be overridden here, plus some extra options like stacked mode. The following shows an example of a stacked area chart with customized colors.

Series of data can be filtered, so that Charts show only the customized data you want to see. To specify which data series you want visible, define the 'series' property as an array of names of the data series as showed below.

```
{
    "type": "chart",
    "title": "Orders+Failures/m",
    "checkDefinitionId": 131,
    "entityId": "GLOBAL",
    "options": {
        "series": {
            "stack": true
        },
        "colors": [
            "#ff3333",
            "#33ff33"
        ]
    "series": [ "Mean", "Peak" ]
}
```

See the Flot documentation for more details.

6.1.1 Data from KairosDB-queries

As detailed in the *Grafana3 and KairosDB section*, all ZMON check data is saved into KairosDB, and can be queried from there. For chart widgets, you can directly use a KairosDB query in the `options` section of a widget to specify the data series to be used. The query consists of the key `metrics` (which indicates the data series to use) and a time specifier, for our purposes usually `start_relative`. In addition you can use `cache_time` (in seconds) to indicate that a previous result can be reused.

Here is an example which shows the values of check 1 for just three of its entities.

```
{
    "options": {
        "lines": {},
        "legend": {
            "backgroundOpacity": 0.1,
            "show": true,
            "position": "ne"
        },
        "series": {
            "stack": false
        },
        "start_relative": {
            "unit": "minutes",
            "value": "30"
        },
        "metrics": [
            {
                "tags": {
                    "entity": [
                        "website-zalando.de",
                        "website-zalando.ch",
                        "website-zalando.at"
                    ],
                    "key": []
                },
                "name": "zmon.check.1",
```

```
                "group_by": [
                    {
                        "name": "tag",
                        "tags": [
                            "entity",
                            "key"
                        ]
                    }
                ]
            }
        ],
        "cache_time": 0,
        "colors": [
            "#F00",
            "#0F0",
            "#00F"
        ]
    },
    "type": "chart",
    "title": "Response time (just de/at/ch)"
}
```

An easy way to compose the KairosDB queries (specially the value for `metrics`) is to create a new Grafana Dashboard in the built-in Grafana and then copy the query from the requests sent by the browser (Developer Tools → Network in Chromium).

6.2 IFRAME

The Iframe widget is a simple widget that allows you to embed a thrid party page in a widget container.

For browser security reasons, only same-domain source urls can be used.

Style property is used to set scale and size of iframe inside the widget container. Normally widths and heights bigger than 100% will be used, and scales around 0.5 are also common.

Reload after a given amount of miliseconds can be done by setting the 'refresh' property.

Sample iframe widget:

```
{
    "type": "iframe",
    "src": "http://example.com",
    "style": {
        "width": "180%",     // Width to be occupied by iframe (px or %).
        "height": "180%",    // Height to be occupied by iframe (px or %).
        "scale: 0.54         // Scaling ratio
    },
    "refresh": 60000         // time in miliseconds after which the iframe content␣
↪will be reloaded.
}
```

6.3 Alert Age

In the rightmost column of each alert block on the dashboard, the age of that alert is shown. An entry of "28m", for example, indicates that the alert is 28 minutes old.

If an alert is raised for multiple entities, the alert age is based on the entity for which the alert has been raised first. Entities in downtime are ignored for determining alert age, but when an entity leaves downtime, the length of time it spent in downtime is taken into account.

An example:

time	event	entity A	entity B	alert age
00:00	alert is raised for entity A	raised for 0h	not raised	0h
01:00	alert is raised for entity B	raised for 1h	raised for 0h	1h (from entity A)
02:00	alert enters downtime for entity A	raised for 2h, in downtime	raised for 1h	1h (from entity B)
03:00	alert leaves downtime for entity A	raised for 3h	raised for 2h	3h (from entity A)
04:00	alert is cleared for entity A	not raised	raised for 3h	3h (from entity B)
05:00	alert enters downtime for entity A	not raised, in downtime	raised for 4h	4h (from entity B)
06:00	alert is raised for entity A	raised for 0h, in downtime	raised for 5h	5h (from entity B)
07:00	alert leaves downtime for entity A	raised for 1h	raised for 6h	6h (from entity B)
08:00	alert is cleared for entity B	raised for 2h	not raised	2h (from entity A)

6.4 Widgets styling and effects based on active alerts

You can change the styling or add a blinking effect to widgets in case one or more alerts are active at the moment. This is done by using the "alertStyles" option, like the sample below:

```
{
    "type": "gauge",
    // Some widget configuration here...
    "alertStyles": {
        "blink": [1, 4, 20],
        "red": [9]
    }
}
```

On the sample below the gauge widget will blink if alert 1, 4 or 20 is active, and make the background red if alert 9 is active. At the moment the following effects are defined:

- blink: will blink the whole widget (opacity 0 to 100%, 1 second interval)
- shake: will start shaking the widget
- red: set the background to red
- orange: set the background to orange
- yellow: set the background to yellow
- green: set the background to green
- blue: set the background to blue

Please note that you can mix different styles and alerts, as shown on the previous sample. If alerts 1 and 9 are active, it will blink with a red background. If you define different styles with the same alert ID it will always give priority to the last one.

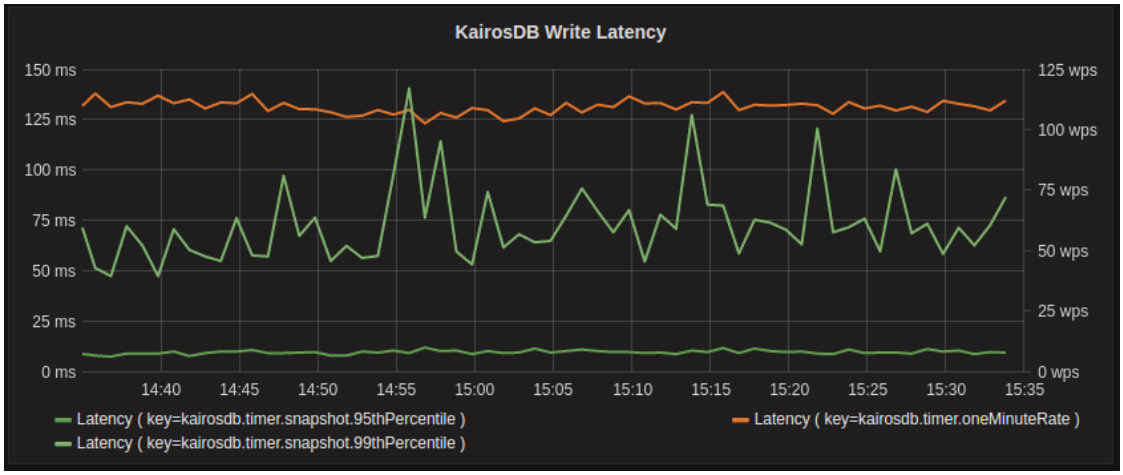
CHAPTER 7

Grafana3 and KairosDB

Grafana is a powerful open-source tool for creating dashboards to visualize metric data. ZMON deploys Grafana 3.x along with the new KairosDB plugin to read metric data from KairosDB. Grafana is served directly from the ZMON controller. Read requests are proxied through the controller so as not to expose the write/delete API from KairosDB. Dashboards are also saved via the controller, so there's no need for any additional data store.

http://grafana.org

Example of latency and requests charted via Grafana:



7.1 Check data

Workers will send all their data to KairosDB. Depending on the KairosDB setting, data is stored forever or you may set a TTL in KairosDB. ZMON will not clean up or roll up any data.

7.1.1 Serialization

For checks retrieving only numeric values data storage in KairosDB is easy. But the worker will also flatten more complex result types and persist them in KairosDB. At Zalando checks that yield only single numeric values have become quite rare.

If the check returns dictionary the worker will try to flatten its structure and persist all entries with a numeric value.

```
{
    "load": {"1min":1,"5min":3,"15min":2}
    "memory_free": 16000
}
```

Will be flattened to an equivalent of

```
{
    "load.1min": 1,
    "load.5min": 3,
    "load.15min": 2,
    "memory_free": 16000
}
```

Always yielding a simple dictionary with (key, value) pairs.

7.1.2 Tagging

KairosDB creates timer series with a name and allows us to tag data points with additional (tagname, tagvalue) pairs.

ZMON stores all data to a single check in a time series named: “zmon.check.<checkid>”.

Single data points are then tagged as follows to describe their contents:

- entity: containing the entity id (some character replace rules are applied)
- key: containing the dict key after serialization of check value (see above)
- metric: contains the last segment of “key” split by “.” (making selection easier in tooling)
- hg: host group(hg) will contain a substring of the entity id, to try to group e.g. cassandra01 and cassandra02 into hg=cassandra

For a certrain set of metrics additional tags may be deployed(REST metrics/actuator)

- sc: HTTP status code
- sg: first digit of HTTP status code

Some of the tagging may seem strange, but as KairosDB does not allow real operations on tags they are basically precreated to allow easier filtering in the tools/charts. This is also fine from a storage/performance point of view during writes, as KairosDB’s Cassandra implementation creates a new row for each unique tuple (time series name, set of tags) thus this is only stored once.

CHAPTER 8

"Read Only" Display Login

The ZMON front end requires users to login. However a very common way of deploying dashboards is on TV screens running across office spaces to e.g. render Grafana or ZMON dashboards. For this ZMON provides you with a way to login a read only authenticated user via one-time tokens. Those tokens can be created by any real user via the ZMON CLI.

8.1 Getting a token

```
zmon onetime-token get

Retrieving new one-time token ...
https://zmon.example.org/tv/AocciOWf/
OK
```

A token can be used exactly **once** to login. However the session created will last up to 365 days.

8.2 Login with token

Use the above URL in the target browser to login directly. This will create a read-only session.

```
https://<your zmon url>/tv/<your token>
```

Note: Please make sure you access the generated URL in order to login. Appending the <token> to any other ZMON location won't work.

8.3 How does it work

First time a valid one time token is used to login we associate a random UUID with it and the device IP. Both are registered within ZMON to create a persisted session, thus this will continue to work after the frontend gets deployed.

So once the token is used, it can no longer be reused and you need to create another one if you want to run additional displays.

CHAPTER 9

Check Command Reference

To give an overview of available commands, we divided them into several categories.

9.1 AppDynamics

Enable AppDynamics Healthrule violations check and *optionally* query underlying Elasticsearch cluster raw logs.

appdynamics (*url=None*, *username=None*, *password=None*, *es_url=None*, *index_prefix=''*)

Initialize AppDynamics wrapper.

Parameters

- **url** (str) – Appdynamics url.
- **username** (str) – Appdynamics username.
- **password** (str) – Appdynamics password.
- **es_url** (str) – Appdynamics Elasticsearch cluster url.
- **index_prefix** (str) – Appdynamics Elasticsearch cluster logs index prefix.

Note: If `username` and `password` are not supplied, then OAUTH2 will be used.

If `appdynamics()` is initialized with no args, then plugin configuration values will be used.

9.1.1 Methods of AppDynamics

healthrule_violations (*application*, *time_range_type=BEFORE_NOW*, *duration_in_mins=5*, *start_time=None*, *end_time=None*, *severity=None*)

Return Healthrule violations for AppDynamics application.

Parameters

- **application** (str) – Application name or ID
- **time_range_type** (str) – Valid time range type. Valid range types are BEFORE_NOW, BEFORE_TIME, AFTER_TIME and BETWEEN_TIMES. Default is BEFORE_NOW.
- **duration_in_mins** (int) – Time duration in mins. Required for BEFORE_NOW, AFTER_TIME, BEFORE_TIME range types. Default is 5 mins.
- **start_time** (int) – Start time (in milliseconds) from which the metric data is returned. Default is 5 mins ago.
- **end_time** (int) – End time (in milliseconds) until which the metric data is returned. Default is now.
- **severity** (str) – Filter results based on severity. Valid values are CRITICAL or WARNING.

Returns List of healthrule violations

Return type *list*

Example query:

```
appdynamics('https://appdynamics/controller/rest').healthrule_violations('49',␣
↪time_range_type='BEFORE_NOW', duration_in_mins=5)

[
    {
        affectedEntityDefinition: {
            entityId: 408,
            entityType: "BUSINESS_TRANSACTION",
            name: "/error"
        },
        detectedTimeInMillis: 0,
        endTimeInMillis: 0,
        id: 39637,
        incidentStatus: "OPEN",
        name: "Backend errrors (percentage)",
        severity: "CRITICAL",
        startTimeInMillis: 1462244635000,
    }
]
```

metric_data(*application*, *metric_path*, *time_range_type=BEFORE_NOW*, *duration_in_mins=5*, *start_time=None*, *end_time=None*, *rollup=True*)

AppDynamics's metric-data API

Parameters

- **application** (str) – Application name or ID
- **metric_path** (str) – The path to the metric in the metric hierarchy
- **time_range_type** (str) – Valid time range type. Valid range types are BEFORE_NOW, BEFORE_TIME, AFTER_TIME and BETWEEN_TIMES. Default is BEFORE_NOW.
- **duration_in_mins** (int) – Time duration in mins. Required for BEFORE_NOW, AFTER_TIME, BEFORE_TIME range types.
- **start_time** (int) – Start time (in milliseconds) from which the metric data is returned. Default is 5 mins ago.

- **end_time** (int) – End time (in milliseconds) until which the metric data is returned. Default is now.
- **rollup** (bool) – By default, the values of the returned metrics are rolled up into a single data point (rollup=True). To get separate results for all values within the time range, set the `rollup` parameter to `False`.

Returns metric values for a metric

Return type *list*

query_logs(*q=''*, *body=None*, *size=100*, *source_type=SOURCE_TYPE_APPLICATION_LOG*, *duration_in_mins=5*)

Perform search query on AppDynamics ES logs.

Parameters

- **q** (str) – Query string used in search.
- **body** (dict) – (dict) holding an ES query DSL.
- **size** (int) – Number of hits to return. Default is 100.
- **source_type** (str) – `sourceType` field filtering. Default to `application-log`, and will be part of `q`.
- **duration_in_mins** (int) – Duration in mins before current time. Default is 5 mins.

Returns ES query result `hits`.

Return type *list*

count_logs(*q=''*, *body=None*, *source_type=SOURCE_TYPE_APPLICATION_LOG*, *duration_in_mins=5*)

Perform count query on AppDynamics ES logs.

Parameters

- **q** (str) – Query string used in search. Will be ingnored if `body` is not None.
- **body** (dict) – (dict) holding an ES query DSL.
- **source_type** (str) – `sourceType` field filtering. Default to `application-log`, and will be part of `q`.
- **duration_in_mins** (int) – Duration in mins before current time. Default is 5 mins. Will be ignored if `body` is not None.

Returns Query match count.

Return type *int*

Note: In case of passing an ES query DSL in `body`, then all filter parameters should be explicitly added in the query body (e.g. `eventTimestamp`, `application_id`, `sourceType`).

9.2 Cassandra

Provides access to a Cassandra cluster via `cassandra()` wrapper object.

cassandra(*node*, *keyspace*, *username=None*, *password=None*, *port=9042*, *connect_timeout=1*, *protocol_version=3*)

Initialize cassandra wrapper.

Parameters

- **node** (str) – Cassandra host.
- **keyspace** (str) – Cassandra keyspace used during the session.
- **username** (str) – Username used in connection. It is recommended to use unprivileged user for cassandra checks.
- **password** (str) – Password used in connection.
- **port** (int) – Cassandra host port. Default is 9042.
- **connect_timeout** (int) – Connection timeout.
- **protocol_version** (str) – Protocol version used in connection. Default is 3.

Note: You should always use an unprivileged user to access your databases. Use `plugin.cassandra.user` and `plugin.cassandra.pass` to configure credentials for the zmon-worker.

---

execute(*stmt*)

Execute a CQL statement against the specified keyspace.

Parameters **stmt** (str) – CQL statement

Returns CQL result

Return type *list*

9.3 CloudWatch

If running on AWS you can use `cloudwatch()` to access AWS metrics easily.

cloudwatch(*region=None*, *assume_role_arn=None*)

Initialize CloudWatch wrapper.

Parameters

- **region** (str) – AWS region for CloudWatch queries. Will be auto-detected if not supplied.
- **assume_role_arn** (str) – AWS IAM role ARN to be assumed. This can be useful in cross-account CloudWatch queries.

9.3.1 Methods of Cloudwatch

query_one(*dimensions*, *metric_name*, *statistics*, *namespace*, *period=60*, *minutes=5*, *start=None*, *end=None*, *extended_statistics=None*)

Query a single AWS CloudWatch metric and return a single scalar value (float). Metric will be aggregated over the last five minutes using the provided aggregation type.

This method is a more low-level variant of the `query` method: all parameters, including all dimensions need to be known.

Parameters

- **dimensions** (dict) – Cloudwatch dimensions. Example `{'LoadBalancerName': 'my-elb-name'}`
- **metric_name** (list) – Cloudwatch metric. Example `'Latency'`.

- **statistics** (list) – Cloudwatch metric statistics. Example 'Sum'
- **namespace** (str) – Cloudwatch namespace. Example 'AWS/ELB'
- **period** (int) – Cloudwatch statistics granularity in seconds. Default is 60.
- **minutes** (int) – Used to determine start time of the Cloudwatch query. Default is 5. Ignored if start is supplied.
- **start** (int) – Cloudwatch start timestamp. Default is None.
- **end** (int) – Cloudwatch end timestamp. Default is None. If not supplied, then end time is now.
- **extended_statistics** (list) – Cloudwatch ExtendedStatistics for percentiles query. Example ['p95', 'p99'].

Returns Return a float if single value, dict otherwise.

Return type *float*, *dict*

Example query with percentiles for AWS ALB:

```
cloudwatch().query_one({'LoadBalancer': 'app/my-alb/1234'}, 'TargetResponseTime',
↪'Average', 'AWS/ApplicationELB', extended_statistics=['p95', 'p99'])
{
    'TargetResponseTime': 0.224,
    'p95': 0.245,
    'p99': 0.300
}
```

Note: In very rare cases, e.g. for ELB metrics, you may see only 1/2 or 1-2/3 of the value in ZMON due to a race condition of what data is already present in cloud watch. To fix this click "evaluate" on the alert, this will trigger the check and move its execution time to a new start time.

---

query (*dimensions*, *metric_name*, *statistics='Sum'*, *namespace=None*, *period=60*, *minutes=5*)

Query AWS CloudWatch for metrics. Metrics will be aggregated over the last five minutes using the provided aggregation type (default "Sum").

dimensions is a dictionary to filter the metrics to query. See the list_metrics boto documentation. You can provide the special value "NOT_SET" for a dimension to only query metrics where the given key is not set. This makes sense e.g. for ELB metrics as they are available both per AZ ("AvailabilityZone" has a value) and aggregated over all AZs ("AvailabilityZone" not set). Additionally you can include the special "*" character in a dimension value to do fuzzy (shell globbing) matching.

metric_name is the name of the metric to filter against (e.g. "RequestCount").

namespace is an optional namespace filter (e.g. "AWS/EC2).

To query an ELB for requests per second:

```
# both using special "NOT_SET" and "*" in dimensions here:
val = cloudwatch().query({'AvailabilityZone': 'NOT_SET', 'LoadBalancerName':
↪'pierone-*'}, 'RequestCount', 'Sum')['RequestCount']
requests_per_second = val / 60
```

You can find existing metrics with the AWS CLI tools:

```
$ aws cloudwatch list-metrics --namespace "AWS/EC2"
```

Use the "dimensions" argument to select on what dimension(s) to aggregate over:

```
$ aws cloudwatch list-metrics --namespace "AWS/EC2" --dimensions␣
↪Name=AutoScalingGroupName,Value=my-asg-FEYBCZF
```

The desired metric can now be queried in ZMON:

```
cloudwatch().query({'AutoScalingGroupName': 'my-asg-*'}, 'DiskReadBytes', 'Sum')
```

alarms (*alarm_names=None*, *alarm_name_prefix=None*, *state_value=STATE_ALARM*, *action_prefix=None*, *max_records=50*)

Retrieve cloudwatch alarms filtered by state value.

See describe_alarms boto documentation for more details.

Parameters

- **alarm_names** (list) – List of alarm names.
- **alarm_name_prefix** (str) – Prefix of alarms. Cannot be specified if alarm_names is specified.
- **state_value** (str) – State value used in alarm filtering. Available values are OK, ALARM (default) and INSUFFICIENT_DATA.
- **action_prefix** (str) – Action name prefix. Example arn:aws:autoscaling: to filter results for all autoscaling related alarms.
- **max_records** (int) – Maximum records to be returned. Default is 50.

Returns List of MetricAlarms.

Return type *list*

```
cloudwatch().alarms(state_value='ALARM')[0]
{
    'ActionsEnabled': True,
    'AlarmActions': ['arn:aws:autoscaling:...'],
    'AlarmArn': 'arn:aws:cloudwatch:...',
    'AlarmConfigurationUpdatedTimestamp': datetime.datetime(2016, 5, 12, 10, 44, 15,␣
↪707000, tzinfo=tzutc()),
    'AlarmDescription': 'Scale-down if CPU < 50% for 10.0 minutes (Average)',
    'AlarmName': 'metric-alarm-for-service-x',
    'ComparisonOperator': 'LessThanThreshold',
    'Dimensions': [
        {
            'Name': 'AutoScalingGroupName',
            'Value': 'service-x-asg'
        }
    ],
    'EvaluationPeriods': 2,
    'InsufficientDataActions': [],
    'MetricName': 'CPUUtilization',
    'Namespace': 'AWS/EC2',
    'OKActions': [],
    'Period': 300,
    'StateReason': 'Threshold Crossed: 1 datapoint (36.1) was less than the threshold␣
↪(50.0).',
    'StateReasonData': '{...}',
    'StateUpdatedTimestamp': datetime.datetime(2016, 5, 12, 10, 44, 16, 294000,␣
↪tzinfo=tzutc()),
    'StateValue': 'ALARM',
```

```
    'Statistic': 'Average',
    'Threshold': 50.0
}
```

9.4 Counter

The `counter()` function allows you to get increment rates of increasing counter values. Main use case for using `counter()` is to get rates per second of JMX counter beans (e.g. "Tomcat Requests"). The counter function requires one parameter `key` to identify the counter.

per_second(*value*)

```
counter('requests').per_second(get_total_requests())
```

Returns the value's increment rate per second. Value must be a float or integer.

per_minute(*value*)

```
counter('requests').per_minute(get_total_requests())
```

Convenience method to return the value's increment rate per minute (same as result of `per_second()` divided by 60).

Internally counter values and timestamps are stored in Redis.

9.5 DNS

The `dns()` function provide a way to resolve hosts.

dns(*host=None*)

9.5.1 Methods of DNS

resolve(*host=None*)

Return IP address of host. If host is `None`, then will resolve host used in initialization. If both are `None` then exception will be raised.

Returns IP address

Return type *str*

Example query:

```
dns('google.de').resolve()
'173.194.65.94'

dns().resolve('google.de')
'173.194.65.94'
```

9.6 EBS

Allows to describe EBS objects (currently, only Snapshots are supported).

ebs()

9.6.1 Methods of EBS

list_snapshots(*account_id*, *max_items*)

List the EBS Snapshots owned by the given account_id. By default, listing is possible for up to 1000 items, so we use pagination internally to overcome this.

Parameters

- **account_id** – AWS account id number (as a string). Defaults to the AWS account id where the check is running.
- **max_items** – the maximum number of snapshots to list. Defaults to 100.

Returns an `EBSSnapshotsList` object

class EBSSnapshotsList

items()

Returns a list of dicts like

```
{
    "id": "snap-12345",
    "description": "Snapshot description...",
    "size": 123,
    "start_time": datetime.datetime(2017, 7, 16, 1, 1, 21,␣
↪tzinfo=tzutc()),
    "state": "completed"
}
```

Example usage:

```
ebs().list_snapshots().items()

snapshots = ebs().list_snapshots(max_items=1000).items()  # for listing more than␣
↪the default of 100 snapshots
start_time = snapshots[0]["start_time"].isoformat()  # returns a string that can␣
↪be passed to time()
age = time() - time(start_time)
```

9.7 Elasticsearch

Provides search queries and health check against an Elasticsearch cluster.

elasticsearch(*url=None*, *timeout=10*, *oauth2=False*)

Note: If `url` is **None**, then the plugin will use the default Elasticsearch cluster set in worker configuration.

9.7.1 Methods of Elasticsearch

search(*indices=None*, *q=''*, *body=None*, *source=True*, *size=DEFAULT_SIZE*)
Search ES cluster using URI or Request body search. If `body` is None then GET request will be used.

Parameters

- **indices** (`list`) – List of indices to search. Limited to only 10 indices. ['_all'] will search all available indices, which effectively leads to same results as *None*. Indices can accept wildcard form.
- **q** (`str`) – Search query string. Will be ignored if `body` is not None.
- **body** (`dict`) – Dict holding an ES query DSL.
- **source** (`bool`) – Whether to include *_source* field in query response.
- **size** (`int`) – Number of hits to return. Maximum value is 1000. Set to 0 if interested in hits count only.

Returns ES query result.

Return type *dict*

Example query:

```
elasticsearch('http://es-cluster').search(indices=['logstash-*'], q='client:192.
↪168.20.* AND http_status:500', size=0, source=False)

{
    "_shards": {
        "failed": 0,
        "successful": 5,
        "total": 5
    },
    "hits": {
        "hits": [],
        "max_score": 0.0,
        "total": 1
    },
    "timed_out": false,
    "took": 2
}
```

count(*indices=None*, *q=''*, *body=None*)
Return ES count of matching query.

Parameters

- **indices** (`list`) – List of indices to search. Limited to only 10 indices. ['_all'] will search all available indices, which effectively leads to same results as *None*. Indices can accept wildcard form.
- **q** (`str`) – Search query string. Will be ignored if `body` is not None.
- **body** (`dict`) – Dict holding an ES query DSL.

Returns ES query result.

Return type *dict*

Example query:

```
elasticsearch('http://es-cluster').count(indices=['logstash-*'], q='client:192.
↪168.20.* AND http_status:500')

{
    "_shards": {
        "failed": 0,
        "successful": 16,
        "total": 16
    },
    "count": 12
}
```

health()
Return ES cluster health.

Returns Cluster health result.

Return type *dict*

```
elasticsearch('http://es-cluster').health()

{
    "active_primary_shards": 11,
    "active_shards": 11,
    "active_shards_percent_as_number": 50.0,
    "cluster_name": "big-logs-cluster",
    "delayed_unassigned_shards": 0,
    "initializing_shards": 0,
    "number_of_data_nodes": 1,
    "number_of_in_flight_fetch": 0,
    "number_of_nodes": 1,
    "number_of_pending_tasks": 0,
    "relocating_shards": 0,
    "status": "yellow",
    "task_max_waiting_in_queue_millis": 0,
    "timed_out": false,
    "unassigned_shards": 11
}
```

9.8 Entities

Provides access to ZMON entities.

entities(*service_url*, *infrastructure_account*, *verify=True*, *oauth2=False*)
Initialize entities wrapper.

Parameters

- **service_url** (str) – Entities service url.
- **infrastructure_account** (str) – Infrastructure account used to filter entities.
- **verify** – Verify SSL connection. Default is True.
- **oauth2** (bool) – Use OAUTH for authentication. Default is False.

Note: If *service_url* or *infrastructure_account* were not supplied, their corresponding values in worker plugin config will be used.

9.8.1 Methods of Entities

search_local(***kwargs*)

Search entities in local infrastructure account. If *infrastructure_account* is not supplied in kwargs, then should search entities "local" to your filtered entities by using the same *infrastructure_account* as a default filter.

Parameters **kwargs** (str) – Filtering kwargs

Returns Entities

Return type *list*

Example searching all `instance` entities in local account:

```
entities().search_local(type='instance')
```

search_all(***kwargs*)

Search all entities.

Parameters **kwargs** (str) – Filtering kwargs

Returns Entities

Return type *list*

alert_coverage(***kwargs*)

Return alert coverage for infrastructure_account.

Parameters **kwargs** (str) – Filtering kwargs

Returns Alert coverage result.

Return type *list*

```
entities().alert_coverage(type='instance', infrastructure_account='1052643')

[
    {
        'alerts': [],
        'entities': [
            {'id': 'app-1-instance', 'type': 'instance'}
        ]
    }
]
```

9.9 EventLog

The `eventlog()` function allows you to conveniently count *EventLog* events by type and time.

count(*event_type_ids*, *time_from*[, *time_to=None*][, *group_by=None*])

Return event counts for given parameters.

event_type_ids is either a single integer (use hex notation, e.g. `0x96001`) or a list of integers.

time_from is a string time specification (`'-5m'` means 5 minutes ago, `'-1h'` means 1 hour ago).

time_to is a string time specification and defaults to *now* if not given.

group_by can specify an EventLog field name to group counts by

```
eventlog().count(0x96001, time_from='-1m')                          # returns a␣
↪single number
eventlog().count([0x96001, 0x63005], time_from='-1m')               # returns dict
↪{'96001': 123, '63005': 456}
eventlog().count(0x96001, time_from='-1m', group_by='appDomainId') # returns dict
↪{'1': 123, '5': 456, ..}
```

The `count()` method internally requests the EventLog Viewer's "count" JSON endpoint.

9.10 History

Wrapper for KairosDB to access history data about checks.

history(*url=None*, *check_id=''*, *entities=None*, *oauth2=False*)

9.10.1 Methods of History

result(*time_from=ONE_WEEK_AND_5MIN*, *time_to=ONE_WEEK*)
Return query result.

Parameters

- **time_from** – Relative time from in seconds. Default is `ONE_WEEK_AND_5MIN`.
- **time_to** – Relative time to in seconds. Default is `ONE_WEEK`.

Returns Json result

Return type *dict*

get_one(*time_from=ONE_WEEK_AND_5MIN*, *time_to=ONE_WEEK*)
Return first result values.

Parameters

- **time_from** – Relative time from in seconds. Default is `ONE_WEEK_AND_5MIN`.
- **time_to** – Relative time to in seconds. Default is `ONE_WEEK`.

Returns List of values

Return type *list*

get_aggregated(*key*, *aggregator*, *time_from=ONE_WEEK_AND_5MIN*, *time_to=ONE_WEEK*)
Return first result values. If no `key` filtering matches, empty list is returned.

Parameters

- **key** (`str`) – Tag key used in filtering the results.
- **aggregator** (`str`) – Aggregator used in query. (e.g 'avg')
- **time_from** – Relative time from in seconds. Default is `ONE_WEEK_AND_5MIN`.
- **time_to** – Relative time to in seconds. Default is `ONE_WEEK`.

Returns List of values

Return type *list*

get_avg (*key*, *time_from=ONE_WEEK_AND_5MIN*, *time_to=ONE_WEEK*)
Return aggregated average.

Parameters

- **key** (str) – Tag key used in filtering the results.
- **time_from** – Relative time from in seconds. Default is ONE_WEEK_AND_5MIN.
- **time_to** – Relative time to in seconds. Default is ONE_WEEK.

Returns List of values

Return type *list*

get_std_dev (*key*, *time_from=ONE_WEEK_AND_5MIN*, *time_to=ONE_WEEK*)
Return aggregated standard deviation.

Parameters

- **key** (str) – Tag key used in filtering the results.
- **time_from** – Relative time from in seconds. Default is ONE_WEEK_AND_5MIN.
- **time_to** – Relative time to in seconds. Default is ONE_WEEK.

Returns List of values

Return type *list*

distance (*self*, *weeks=4*, *snap_to_bin=True*, *bin_size='1h'*, *dict_extractor_path=''*)
For detailed docs on distance function please see *History distance functionality* .

9.11 HTTP

Access to HTTP (and HTTPS) endpoints is provided by the `http()` function.

http (*url*[, *method='GET'*][, *timeout=10*][, *max_retries=0*][, *verify=True*][, *oauth2=False*][, *allow_redirects=None*][, *headers=None*])

Parameters

- **url** (str) – The URL that is to be queried. See below for details.
- **method** (str) – The HTTP request method. Allowed values are GET or HEAD.
- **timeout** (float) – The timeout for the HTTP request, in seconds. Defaults to 10.
- **max_retries** (int) – The number of times the HTTP request should be retried if it fails. Defaults to 0.
- **verify** (bool) – Can be set to False to disable SSL certificate verification.
- **oauth2** (bool) – Can be set to True to inject a OAuth 2 Bearer access token in the outgoing request
- **oauth2_token_name** (str) – The name of the OAuth 2 token. Default is uid.
- **allow_redirects** (bool) – Follow request redirects. If None then it will be set to True in case of GET and False in case of HEAD request.
- **headers** (dict) – The headers to be used in the HTTP request.

Returns

An object encapsulating the response from the server. See below.

For checks on entities that define the attributes `url` or `host`, the given URL may be relative. In that case, the URL `http://<value><url>` is queried, where *<value>* is the value of that attribute, and *<url>* is the URL passed to this function. If an entity defines both `url` and `host`, the former is used.

This function cannot query URLs using a scheme other than HTTP or HTTPS; URLs that do not start with `http://` or `https://` are considered to be relative.

Example:

```
http('http://www.example.org/data?fetch=json').json()

# avoid raising error in case the response error status (e.g. 500 or 503)
# but you are interested in the response json
http('http://www.example.org/data?fetch=json').json(raise_error=False)
```

9.11.1 HTTP Responses

The object returned by the `http()` function provides methods: `json()`, `text()`, `headers()`, `cookies()`, `content_size()`, `time()` and `code()`.

`json`(*raise_error=True*)
This method returns an object representing the content of the JSON response from the queried endpoint. Usually, this will be a map (represented by a Python *dict*), but, depending on the endpoint, it may also be a list, string, set, integer, floating-point number, or Boolean.

`text`(*raise_error=True*)
Returns the text response from queried endpoint:

```
http("/heartbeat.jsp", timeout=5).text().strip()=='OK: JVM is running'
```

Since we're using a relative url, this check has to be defined for specific entities (e.g. type=zomcat will run it on all zomcat instances). The strip function removes all leading and trailing whitespace.

`headers`(*raise_error=True*)
Returns the response headers in a case-insensitive dict-like object:

```
http("/api/json", timeout=5).headers()['content-type']=='application/json'
```

`cookies`(*raise_error=True*)
Returns the response cookies in a dict like object:

```
http("/heartbeat.jsp", timeout=5).cookies()['my_custom_cookie'] == 'custom_cookie_
↪value'
```

`content_size`(*raise_error=True*)
Returns the length of the response content:

```
http("/heartbeat.jsp", timeout=5).content_size() > 1024
```

`time`(*raise_error=True*)
Returns the elapsed time in seconds until response was received:

```
http("/heartbeat.jsp", timeout=5).time() > 1.5
```

code()
Return HTTP status code from the queried endpoint.:

```
http("/heartbeat.jsp", timeout=5).code()
```

actuator_metrics(*prefix='zmon.response.'*, *raise_error=True*)
Parses the json result of a metrics endpoint into a map ep->method->status->metric

http("/metrics", timeout=5).actuator_metrics()

prometheus()
Parse the resulting text result according to the Prometheus specs using their prometheus_client.

http("/metrics", timeout=5).prometheus()

9.12 JMX

To use JMXQuery, run "jmxquery" (this is not yet released)

Queries beans' attributes on hosts specified in entities filter:

```
jmx().query('java.lang:type=Memory', 'HeapMemoryUsage', 'NonHeapMemoryUsage').
↪results()
```

Another example:

```
jmx().query('java.lang:type=Threading', 'ThreadCount', 'DaemonThreadCount',
↪'PeakThreadCount').results()
```

This would return a dict like:

```
{
    "DaemonThreadCount": 524,
    "PeakThreadCount": 583,
    "ThreadCount": 575
}
```

9.13 KairosDB

Provides read access to the target KairosDB

kairosdb(*url*, *oauth2=False*)

9.13.1 Methods of KairosDB

query(*name*, *group_by = None*, *tags = None*, *start = -5*, *end = 0*, *time_unit='seconds'*, *aggregators = None*, *start_absolute = None*, *end_absolute = None*)
Query kairosdb.

Parameters

- **name** (str) – Metric name.

- **group_by** (list) – List of fields to group by.
- **tags** (dict) – Filtering tags.
- **start** (int) – Relative start time. Default is 5. Should be greater than or equal 1.
- **end** (int) – End time. Default is 0. If not 0, then it should be greater than or equal to 1.
- **time_unit** (*str.*) – Time unit ('seconds', 'minutes', 'hours'). Default is 'minutes'.
- **aggregators** (list) – List of aggregators.
- **start_absolute** (long) – Absolute start time in milliseconds, overrides the start parameter which is relative
- **end_absolute** (long) – Absolute end time in milliseconds, overrides the end parameter which is relative

Returns Result queries.

Return type *dict*

9.14 Kubernetes

Provides a wrapper for querying Kubernetes cluster resources.

kubernetes (*namespace='default'*)

If `namespace` is `None` then **all** namespaces will be queried. This however will increase the number of calls to Kubernetes API server.

Note:

- Kubernetes wrapper will authenticate using service account, which assumes the worker is running in a Kubernetes cluster.
- All Kubernetes wrapper calls are scoped to the Kubernetes cluster hosting the worker. It is not intended to be used in querying multiple clusters.

9.14.1 Label Selectors

Kubernetes API provides a way to filter resources using labelSelector. Kubernetes wrapper provides a friendly syntax for filtering.

The following examples show different usage of the Kubernetes wrapper utilizing label filtering:

```
# Get all pods with label ``application`` equal to ``zmon-worker``
kubernetes().pods(application='zmon-worker')
kubernetes().pods(application__eq='zmon-worker')


# Get all pods with label ``application`` **not equal to** ``zmon-worker``
kubernetes().pods(application__neq='zmon-worker')


# Get all pods with label ``application`` **any of** ``zmon-worker`` or ``zmon-agent``
kubernetes().pods(application__in=['zmon-worker', 'zmon-agent'])
```

```
# Get all pods with label ``application`` **not any of** ``zmon-worker`` or ``zmon-
↪agent``
kubernetes().pods(application__notin=['zmon-worker', 'zmon-agent'])
```

9.14.2 Methods of Kubernetes

pods (*name=None*, *phase=None*, *ready=None*, ***kwargs*)
Return list of Pods.

Parameters

- **name** (str) – Pod name.
- **phase** (str) – Pod status phase. Valid values are: Pending, Running, Failed, Succeeded or Unknown.
- **ready** (bool) – Pod readiness status. If None then all pods are returned.
- ****kwargs** – Pod labelSelectors filters.

Returns List of pods. Typical pod has "metadata", "status" and "spec" fields.

Return type *list*

nodes (*name=None*, ***kwargs*)
Return list of Nodes. Namespace does not apply.

Parameters

- **name** (str) – Node name.
- ****kwargs** – Node labelSelectors filters.

Returns List of nodes. Typical pod has "metadata", "status" and "spec" fields.

Return type *list*

services (*name=None*, ***kwargs*)
Return list of Services.

Parameters

- **name** (str) – Service name.
- ****kwargs** – Service labelSelectors filters.

Returns List of services. Typical service has "metadata", "status" and "spec" fields.

Return type *list*

endpoints (*name=None*, ***kwargs*)
Return list of Endpoints.

Parameters

- **name** (str) – Endpoint name.
- ****kwargs** – Endpoint labelSelectors filters.

Returns List of Endpoints. Typical Endpoint has "metadata", and "subsets" fields.

Return type *list*

ingresses (*name=None*, ***kwargs*)
Return list of Ingresses.

Parameters

- **name** (str) – Ingress name.
- ****kwargs** – Ingress labelSelectors filters.

Returns List of Ingresses. Typical Ingress has "metadata", "spec" and "status" fields.

Return type *list*

statefulsets(*name=None*, *replicas=None*, ***kwargs*)
Return list of Statefulsets.

Parameters

- **name** (str) – Statefulset name.
- **replicas** (int) – Statefulset replicas.
- ****kwargs** – Statefulset labelSelectors filters.

Returns List of Statefulsets. Typical Statefulset has "metadata", "status" and "spec" fields.

Return type *list*

daemonsets(*name=None*, ***kwargs*)
Return list of Daemonsets.

Parameters

- **name** (str) – Daemonset name.
- ****kwargs** – Daemonset labelSelectors filters.

Returns List of Daemonsets. Typical Daemonset has "metadata", "status" and "spec" fields.

Return type *list*

replicasets(*name=None*, *replicas=None*, ***kwargs*)
Return list of ReplicaSets.

Parameters

- **name** (str) – ReplicaSet name.
- **replicas** (int) – ReplicaSet replicas.
- ****kwargs** – ReplicaSet labelSelectors filters.

Returns List of ReplicaSets. Typical ReplicaSet has "metadata", "status" and "spec" fields.

Return type *list*

deployments(*name=None*, *replicas=None*, *ready=None*, ***kwargs*)
Return list of Deployments.

Parameters

- **name** (str) – Deployment name.
- **replicas** (int) – Deployment replicas.
- **ready** (bool) – Deployment readiness status.
- ****kwargs** – Deployment labelSelectors filters.

Returns List of Deployments. Typical Deployment has "metadata", "status" and "spec" fields.

Return type *list*

configmaps (*name=None*, ***kwargs*)
Return list of ConfigMaps.

Parameters

- **name** (str) – ConfigMap name.
- ****kwargs** – ConfigMap labelSelectors filters.

Returns List of ConfigMaps. Typical ConfigMap has "metadata" and "data".

Return type *list*

persistentvolumeclaims (*name=None*, *phase=None*, ***kwargs*)
Return list of PersistentVolumeClaims.

Parameters

- **name** (str) – PersistentVolumeClaim name.
- **phase** (str) – Volume phase.
- ****kwargs** – PersistentVolumeClaim labelSelectors filters.

Returns List of PersistentVolumeClaims. Typical PersistentVolumeClaim has "metadata", "status" and "spec" fields.

Return type *list*

persistentvolumes (*name=None*, *phase=None*, ***kwargs*)
Return list of PersistentVolumes.

Parameters

- **name** (str) – PersistentVolume name.
- **phase** (str) – Volume phase.
- ****kwargs** – PersistentVolume labelSelectors filters.

Returns List of PersistentVolumes. Typical PersistentVolume has "metadata", "status" and "spec" fields.

Return type *list*

metrics ()
Return API server metrics in prometheus format.

Returns Cluster metrics.

Return type *dict*

9.15 LDAP

Retrieve OpenLDAP statistics (needs "cn=Monitor" database installed in LDAP server).

```
ldap().statistics()
```

This would return a dict like:

```
{
    "connections_current": 77,
    "connections_per_sec": 27.86,
    "entries": 359369,
```

```
    "max_file_descriptors": 65536,
    "operations_add_per_sec": 0.0,
    "operations_bind_per_sec": 27.99,
    "operations_delete_per_sec": 0.0,
    "operations_extended_per_sec": 0.23,
    "operations_modify_per_sec": 0.09,
    "operations_search_per_sec": 24.09,
    "operations_unbind_per_sec": 27.82,
    "waiters_read": 76,
    "waiters_write": 0
}
```

All information is based on the cn=Monitor OpenLDAP tree. You can get more information in the OpenLDAP Administrator's Guide. The meaning of the different fields is as follows:

connections_current Number of currently established TCP connections.

connections_per_sec Increase of connections per second.

entries Number of LDAP records.

operations_*_per_sec Number of operations per second per operation type (add, bind, search, ..).

waiters_read Number of waiters for read (whatever that means, OpenLDAP documentation does not say anything).

9.16 Memcached

Read-only access to memcached servers is provided by the `memcached()` function.

memcached([*host=some.host*][, *port=11211*])
Returns a connection to the Memcached server at *<host>*:*<port>*, where *<host>* is the value of the current entity's `host` attribute, and *<port>* is the given port (default `11211`). See below for a list of methods provided by the returned connection object.

9.16.1 Methods of the Memcached Connection

The object returned by the `memcached()` function provides the following methods:

get(*key*)
Returns the string stored at *key*. If *key* does not exist an error is raised.

```
memcached().get("example_memcached_key")
```

json(*key*)
Returns the data of the key as unserialized JSON data. I.e. you can store a JSON object as value of the key and get a dict back

```
memcached().json("example_memcached_key")
```

stats([*extra_keys=[STR, STR*])
Returns a `dict` with general Memcached statistics such as memory usage and operations/s. All values are extracted using the Memcached STATS command.

The *extra_keys* may be retrieved as returned as well from the memcached server's *stats* command, e.g. *version* or *uptime*.

Example result:


```
{
    "incr_hits_per_sec": 0,
    "incr_misses_per_sec": 0,
    "touch_misses_per_sec": 0,
    "decr_misses_per_sec": 0,
    "touch_hits_per_sec": 0,
    "get_expired_per_sec": 0,
    "get_hits_per_sec": 100.01,
    "cmd_get_per_sec": 119.98,
    "cas_hits_per_sec": 0,
    "cas_badval_per_sec": 0,
    "delete_misses_per_sec": 0,
    "bytes_read_per_sec": 6571.76,
    "auth_errors_per_sec": 0,
    "cmd_set_per_sec": 19.97,
    "bytes_written_per_sec": 6309.17,
    "get_flushed_per_sec": 0,
    "delete_hits_per_sec": 0,
    "cmd_flush_per_sec": 0,
    "curr_items": 37217768,
    "decr_hits_per_sec": 0,
    "connections_per_sec": 0.02,
    "cas_misses_per_sec": 0,
    "cmd_touch_per_sec": 0,
    "bytes": 3902170728,
    "evictions_per_sec": 0,
    "auth_cmds_per_sec": 0,
    "get_misses_per_sec": 19.97
}
```


9.17 MongoDB

Provides access to a MongoDB cluster

mongodb(*host*, *port=27017*)

9.17.1 Methods of MongoDB

find(*database*, *collection*, *query*)

9.18 Nagios

This function provides a wrapper for Nagios plugins.

check_load()

```
nagios().nrpe('check_load')
```

Example check result as JSON:

```
{
    "load1": 2.86,
    "load15": 3.13,
    "load5": 3.23
}
```

check_list_timeout()

```
nagios().nrpe('check_list_timeout',  path="/data/production/", timeout=10)
```

This command will run "timeout 10 ls /data/production/" on the target host via nrpe.

Example check result as JSON:

```
{

    "exit":0,
    "timeout":0
}
```

Exit is the exitcode from nrpe 0 for OK, 2 for ERROR. Timeout should not be used, yet.

check_diff_reverse()

```
nagios().nrpe('check_diff_reverse')
```

Example check result as JSON:

```
{
    "CommitLimit-Committed_AS": 16022524
}
```

check_mailq_postfix()

```
nagios().nrpe('check_mailq_postfix')
```

Example check result as JSON:

```
{
    "unsent": 0
}
```

check_memcachestatus()

```
nagios().nrpe('check_memcachestatus', port=11211)
```

Example check result as JSON:

```
{
    "curr_connections": 0.0,
    "cmd_get": 3569.09,
    "bytes_written": 66552.9,
    "get_hits": 1593.9,
    "cmd_set": 0.04,
```

```
    "curr_items": 0.0,
    "get_misses": 1975.19,
    "bytes_read": 83077.28
}
```

check_findfiles()

Find-file analyzer plugin for Nagios. This plugin checks for newer files within a directory and checks their access time, modification time and count.

```
nagios().nrpe('check_findfiles', directory='/data/example/error/', epoch=1)
```

Example check result as JSON:

```
{
    "ftotal": 0,
    "faccess": 0,
    "fmodify": 0
}
```

check_findolderfiles()

Find-file analyzer plugin for Nagios. This plugin checks for files within a directory older than 2 given times in minutes.

```
nagios().nrpe('check_findolderfiles', directory='/data/stuff,/mnt/other',␣
↪time01=480, time02=600)
```

Example check result as JSON:

```
{
    "total files": 831,
    "files older than time01": 112,
    "files older than time02": 0
}
```

check_findfiles_names()

Find-file analyzer plugin for Nagios. This plugin checks for newer files within a directory, optionally matching a filename pattern, and checks their access time, modification time and count.

```
nagios().nrpe('check_findfiles_names', directory='/mnt/storage/error/', epoch=1,␣
↪name='app*')
```

Example check result as JSON:

```
{
    "ftotal": 0,
    "faccess": 0,
    "fmodify": 0
}
```

check_findfiles_names_exclude()

Find-file analyzer plugin for Nagios. This plugin checks for newer files within a directory, optionally matching a filename pattern(in this command the files are excluded), and checks their access time, modification time and count.

```
nagios().nrpe('check_findfiles_names_exclude', directory='/mnt/storage/error/',␣
↪epoch=1, name='app*')
```

Example check result as JSON:

```
{
    "ftotal": 0,
    "faccess": 0,
    "fmodify": 0
}
```

check_logwatch()

```
nagios().nrpe('check_logwatch', logfile='/var/logs/example/p{}/catalina.out'.
↪format(entity['instance']), pattern='Full.GC')
```

Example check result as JSON:

```
{
    "last": 0,
    "total": 0
}
```

check_ntp_time()

```
nagios().nrpe('check_ntp_time')
```

Example check result as JSON:

```
{
    "offset": 0.003063
}
```

check_iostat()

```
nagios().nrpe('check_iostat', disk='sda')
```

Example check result as JSON:

```
{
    "tps": 944.7,
    "iowrite": 6858.4,
    "ioread": 6268.4
}
```

check_hpacucli()

```
nagios().nrpe('check_hpacucli')
```

Example check result as JSON:

```
{
    "logicaldrive_1": "OK",
    "logicaldrive_2": "OK",
    "logicaldrive_3": "OK",
    "physicaldrive_2I:1:6": "OK",
    "physicaldrive_2I:1:5": "OK",
```

```
    "physicaldrive_1I:1:3": "OK",
    "physicaldrive_1I:1:2": "OK",
    "physicaldrive_1I:1:1": "OK",
    "physicaldrive_1I:1:4": "OK"
}
```

check_hpasm_fix_power_supply()

```
nagios().nrpe('check_hpasm_fix_power_supply')
```

Example check result as JSON:

```
{
    "status": "OK",
    "message": "System: 'proliant dl360 g6', S/N: 'CZJ947016M', ROM: 'P64 05/05/
↪2011', hardware working fine, da: 3 logical drives, 6 physical drives cpu_0=ok␣
↪cpu_1=ok ps_2=ok fan_1=46% fan_2=46% fan_3=46% fan_4=46% temp_1=21 temp_2=40␣
↪temp_3=40 temp_4=36 temp_5=35 temp_6=37 temp_7=32 temp_8=36 temp_9=32 temp_
↪10=36 temp_11=32 temp_12=33 temp_13=48 temp_14=29 temp_15=32 temp_16=30 temp_
↪17=29 temp_18=39 temp_19=37 temp_20=38 temp_21=45 temp_22=42 temp_23=39 temp_
↪24=48 temp_25=35 temp_26=46 temp_27=35 temp_28=71 | fan_1=46%;0;0 fan_2=46%;0;0␣
↪fan_3=46%;0;0 fan_4=46%;0;0 'temp_1_ambient'=21;42;42 'temp_2_cpu#1'=40;82;82
↪'temp_3_cpu#2'=40;82;82 'temp_4_memory_bd'=36;87;87 'temp_5_memory_bd'=35;78;78
↪'temp_6_memory_bd'=37;87;87 'temp_7_memory_bd'=32;78;78 'temp_8_memory_bd'=36;
↪87;87 'temp_9_memory_bd'=32;78;78 'temp_10_memory_bd'=36;87;87 'temp_11_memory_
↪bd'=32;78;78 'temp_12_power_supply_bay'=33;59;59 'temp_13_power_supply_bay'=48;
↪73;73 'temp_14_memory_bd'=29;60;60 'temp_15_processor_zone'=32;60;60 'temp_16_
↪processor_zone'=3"
}
```

check_hpasm_gen8()

```
nagios().nrpe('check_hpasm_gen8')
```

Example check result as JSON:

```
{
    "status": "OK",
    "message": "ignoring 16 dimms with status 'n/a' , System: 'proliant dl360p␣
↪gen8', S/N: 'CZJ2340R6C', ROM: 'P71 08/20/2012', hardware working fine, da: 1␣
↪logical drives, 4 physical drives"
}
```

check_openmanage()

```
nagios().nrpe('check_openmanage')
```

Example check result as JSON:

```
{
    "status": "OK",
    "message": "System: 'PowerEdge R720', SN: 'GN2J8X1', 256 GB ram (16 dimms), 5␣
↪logical drives, 10 physical drives|T0_System_Board_Inlet=21C;42;47 T1_System_
↪Board_Exhaust=36C;70;75 T2_CPU1=59C;95;100 T3_CPU2=52C;95;100 W2_System_Board_
↪Pwr_Consumption=168W;896;980 A0_PS1_Current_1=0.8A;0;0 A1_PS2_Current_2=0.2A;0;
↪0 V25_PS1_Voltage_1=230V;0;0 V26_PS2_Voltage_2=232V;0;0 F0_System_Board_
↪Fan1=1680rpm;0;0 F1_System_Board_Fan2=1800rpm;0;0 F2_System_Board_Fan3=1680rpm;
↪0;0 F3_System_Board_Fan4=2280rpm;0;0 F4_System_Board_Fan5=2400rpm;0;0 F5_System_
↪Board_Fan6=2400rpm;0;0"
```

```
}
```

check_ping()

```
nagios().local('check_ping')
```

Example check result as JSON:

```
{
    "rta": 1.899,
    "pl": 0.0
}
```

check_apachestatus_uri()

```
nagios().nrpe('check_apachestatus_uri', url='http://127.0.0.1/server-status?auto
↪') or nagios().nrpe('check_apachestatus_uri', url='http://127.0.0.1:10083/
↪server-status?auto')
```

Example check result as JSON:

```
{
    "idle": 60.0,
    "busy": 15.0,
    "hits": 24.256,
    "kBytes": 379.692
}
```

check_check_command_procs()

```
nagios().nrpe('check_command_procs', process='httpd')
```

Example check result as JSON:

```
{
    "procs": 33
}
```

check_http_expect_port_header()

```
nagios().nrpe('check_http_expect_port_header', ip='localhost', url= '/', redirect=
↪'warning', size='9000:90000', expect='200', port='88', hostname='www.example.com
↪')
```

Example check result as JSON:

```
{
    "size": 33633.0,
    "time": 0.080755
}
```

NOTE: if the nrpe check returns an 'expect'result(return code is not the expected) , the check returns a NagiosError

check_mysql_processes()

```
nagios().nrpe('check_mysql_processes', host='localhost', port='/var/lib/mysql/
↪mysql.sock', user='myuser', password='mypas')
```

Example check result as JSON:

```
{
    "avg": 0,
    "threads": 1
}
```

check_mysqlperformance()

```
nagios().nrpe('check_mysqlperformance', host='localhost', port='/var/lib/mysql/
↪mysql.sock', user='myuser', password='mypass')
```

Example check result as JSON:

```
{
    "Com_select": 15.27,
    "Table_locks_waited": 0.01,
    "Select_scan": 2.25,
    "Com_change_db": 0.0,
    "Com_insert": 382.26,
    "Com_replace": 8.09,
    "Com_update": 335.7,
    "Com_delete": 0.02,
    "Qcache_hits": 16.57,
    "Questions": 768.14,
    "Qcache_not_cached": 1.8,
    "Created_tmp_tables": 2.43,
    "Created_tmp_disk_tables": 2.25,
    "Aborted_clients": 0.3
}
```

check_mysql_slave()

```
nagios().nrpe('check_mysql_slave', host='localhost', port='/var/lib/mysql/mysql.
↪sock', database='mydb', user='myusr', password='mypwd')
```

Example check result as JSON:

```
{
    "Uptime": 6215760.0,
    "Open tables": 3953.0,
    "Slave IO": "Yes",
    "Queries per second avg": 967.106,
    "Slow queries": 1047406.0,
    "Seconds Behind Master": 0.0,
    "Threads": 1262.0,
    "Questions": 6011300666.0,
    "Slave SQL": "Yes",
    "Flush tables": 1.0,
```

```
    "Opens": 59466.0
}
```

check_ssl_cert()

```
nagios().nrpe('check_ssl_cert', host_ip='91.240.34.73', domain_name='www.example.
↪com') or nagios().local('check_ssl_cert', host_ip='91.240.34.73', domain_name=
↪'www.example.com')
```

Example check result as JSON:

```
{
    "days": 506
}
```

9.18.1 NRPE checks for Windows Hosts

Checks are based on nsclient++ v.0.4.1. For more info look: http://docs.nsclient.org/

CheckCounter()

Returns performance counters for a process(usedMemory/WorkingSet)

```
nagios().win('CheckCounter', process='eo_server')
```

Example check result as JSON:

used memory in bytes

```
{
    "ProcUsedMem": 811024384
}
```

CheckCPU()

```
nagios().win('CheckCPU')
```

Example check result as JSON:

```
{
    "1": 4,
    "10": 8,
    "5": 14
}
```

CheckDriveSize()

```
nagios().win('CheckDriveSize')
```

Example check result as JSON:

Used Space in MByte

```
{
    "C:\\ %": 61.0,
    "C:\\": 63328.469

}
```

CheckEventLog()

```
nagios().win('CheckEventLog', log='application', query='generated gt -7d AND␣
↪type=\'error\'')
```

'generated gt -7d' means in the last 7 days

Example check result as JSON:

```
{
    "eventlog": 20
}
```

CheckFiles()

```
nagios().win('CheckFiles', path='C:\\Import\\Exchange2Clearing', pattern='*.*',␣
↪query='creation lt -1h')
```

'creation lt -1h' means older than 1 hour

Example check result as JSON:

```
{
    "found files": 22
}
```

CheckLogFile()

```
nagios().win('CheckLogFile', logfile='c:\Temp\log\maxflow_portal.log', seperator=
↪' ', query='column4 = \'ERROR\' OR column4 = \'FATAL\'')
```

Example check result as JSON:

```
{
    "count": 4
}
```

CheckMEM()

```
nagios().win('CheckMEM')
```

Example check result as JSON:

used memory in MBytes

```
{
    "page file %": 16.0,
    "page file": 5534.105,
```

```
    "physical memory": 3331.109,
    "virtual memory": 268.777,
    "virtual memory %": 0.0,
    "physical memory %": 20.0
}
```

CheckProcState()

```
nagios().win('CheckProcState', process='check_mk_agent.exe')
```

Example check result as JSON:

```
{
    "status": "OK",
    "message": "check_mk_agent.exe: running"
}
```

CheckServiceState()

```
nagios().win('CheckServiceState', service='ENAIO_server')
```

Example check result as JSON:

```
{
    "status": "OK",
    "message": "ENAIO_server: started"
}
```

CheckUpTime()

```
nagios().win('CheckUpTime')
```

Example check result as JSON:

uptime in ms

```
{
    "uptime": 412488000
}
```

9.19 Ping

Simple ICMP ping function which returns `True` if the ping command returned without error and `False` otherwise.

ping(*timeout=1*)

```
ping()
```

The `timeout` argument specifies the timeout in seconds. Internally it just runs the following system command:

```
ping -c 1 -w <TIMEOUT> <HOST>
```

9.20 Redis

Read-only access to Redis servers is provided by the `redis()` function.

redis ([*port=6379*][, *db=0*])
: Returns a connection to the Redis server at *<host>*:*<port>*, where *<host>* is the value of the current entity's `host` attribute, and *<port>* is the given port (default `6379`). See below for a list of methods provided by the returned connection object.

Please also have a look at the Redis documentation.

9.20.1 Methods of the Redis Connection

The object returned by the `redis()` function provides the following methods:

llen (*key*)
: Returns the length of the list stored at *key*. If *key* does not exist, it's value is treated as if it were an empty list, and 0 is returned. If *key* exists but is not a list, an error is raised.

```
redis().llen("prod_eventlog_queue")
```

lrange (*key*, *start*, *stop*)
: Returns the elements of the list stored at *key* in the range [*start*, *stop*]. If *key* does not exist, it's value is treated as if it were an empty list. If *key* exists but is not a list, an error is raised.

  The parameters *start* and *stop* are zero-based indexes. Negative numbers are converted to indexes by adding the length of the list, so that `-1` is the last element of the list, `-2` the second-to-last element of the list, and so on.

  Indexes outside the range of the list are not an error: If both *start* and *stop* are less than 0 or greater than or equal to the length of the list, an empty list is returned. Otherwise, if *start* is less than 0, it is treated as if it were 0, and if *stop* is greater than or equal to the the length of the list, it is treated as if it were equal to the length of the list minus 1. If *start* is greater than *stop*, an empty list is returned.

  Note that this method is subtly different from Python's list slicing syntax, where `list[start:stop]` returns elements in the range [*start*, *stop*).

```
redis().lrange("prod_eventlog_queue", 0, 9)    # Returns *ten* elements!
redis().lrange("prod_eventlog_queue", 0, -1)   # Returns the entire list.
```

get (*key*)
: Returns the string stored at *key*. If *key* does not exist, returns `None`. If *key* exists but is not a string, an error is raised.

```
redis().get("example_redis_key")
```

keys (*pattern*)
: Returns list of keys from Redis matching pattern.

```
redis().keys("*downtime*")
```

hget (*key*, *field*)
: Returns the value of the field *field* of the hash *key*. If *key* does not exist or does not have a field named *field*, returns `None`. If *key* exists but is not a hash, an error is raised.

```
redis().hget("example_hash_key", "example_field_name")
```

hgetall (*key*)
Returns a `dict` of all fields of the hash *key*. If *key* does not exist, returns an empty `dict`. If *key* exists but is not a hash, an error is raised.

```
redis().hgetall("example_hash_key")
```

scan (*cursor*[, *match=None*][, *count=None*])
Returns a `set` with the next cursor and the results from this scan. Please see the Redis documentation on how to use this function exactly: http://redis.io/commands/scan

```
redis().scan(0, 'prefix*', 10)
```

smembers (*key*)
Returns members of set `key` in Redis.

```
redis().smembers("zmon:alert:1")
```

ttl (*key*)
Return the time to live of an expiring key.

```
redis().ttl('lock')
```

statistics ()
Returns a `dict` with general Redis statistics such as memory usage and operations/s. All values are extracted using the Redis INFO command.

Example result:

```
{
    "blocked_clients": 2,
    "commands_processed_per_sec": 15946.48,
    "connected_clients": 162,
    "connected_slaves": 0,
    "connections_received_per_sec": 0.5,
    "dbsize": 27351,
    "evicted_keys_per_sec": 0.0,
    "expired_keys_per_sec": 0.0,
    "instantaneous_ops_per_sec": 29626,
    "keyspace_hits_per_sec": 1195.43,
    "keyspace_misses_per_sec": 1237.99,
    "used_memory": 50781216,
    "used_memory_rss": 63475712
}
```

Please note that the values for both *used_memory* and *used_memory_rss* are in Bytes.

9.21 S3

Allows data to be pulled from S3 Objects.

s3 ()

9.21.1 Methods of S3

get_object_metadata(*bucket_name*, *key*)
Get the metadata associated with the given `bucket_name` and `key`. The metadata allows you to check for the existance of the key within the bucket and to check how large the object is without reading the whole object into memory.

Parameters

- **bucket_name** – the name of the S3 Bucket
- **key** – the key that identifies the S3 Object within the S3 Bucket

Returns an `S3ObjectMetadata` object

class S3ObjectMetadata

exists()
Will return True if the object exists.

size()
Returns the size in bytes for the object. Will return -1 for objects that do not exist.

Example usage:

```
s3().get_object_metadata('my bucket', 'mykeypart1/mykeypart2').exists()


s3().get_object_metadata('my bucket', 'mykeypart1/mykeypart2').size()
```

get_object(*bucket_name*, *key*)
Get the S3 Object associated with the given `bucket_name` and `key`. This method will cause the object to be read into memory.

Parameters

- **bucket_name** – the name of the S3 Bucket
- **key** – the key that identifies the S3 Object within the S3 Bucket

Returns an `S3Object` object

class S3Object

text()
Get the S3 Object data

json()
If the object exists, parse the object as JSON.
Returns a dict containing the parsed JSON or None if the object does not exist.

exists()
Will return True if the object exists.

size()
Returns the size in bytes for the object. Will return -1 for objects that do not exist.

Example usage:

```
s3().get_object('my bucket', 'mykeypart1/my_text_doc.txt').text()

s3().get_object('my bucket', 'mykeypart1/my_json_doc.json').json()
```

list_bucket (*bucket_name*, *prefix*, *max_items*)
List the S3 Object associated with the given `bucket_name`, matching `prefix`. By default, listing is possible for up to 1000 keys, so we use pagination internally to overcome this.

Parameters

- **bucket_name** – the name of the S3 Bucket
- **prefix** – the prefix to search under
- **max_items** – the maximum number of objects to list. Defaults to 100.

Returns an `S3FileList` object

class S3FileList

files()
Returns a list of dicts like

```
{
    "file_name": "foo",
    "size": 12345,
    "last_modified": datetime.datetime(2017, 7, 16, 1, 1, 21,␣
↪tzinfo=tzutc())
}
```

Example usage:

```
s3().list_bucket('my bucket', 'some_prefix').files()

files = s3().list_bucket('my bucket', 'some_prefix', 10000).files()  # for␣
↪listing a lot of keys
last_modified = files[0]["last_modified"].isoformat()  # returns a string that␣
↪can be passed to time()
age = time() - time(last_modified)
```

9.22 Scalyr

The `scalyr()` wrapper enables querying Scalyr from your AWS worker if the credentials have been specified for the worker instance(s).

count (*query*, *minutes=5*)
Run a count query against Scalyr, depending on number of queries you may run into rate limit.

```
scalyr().count(' ERROR ')
```

timeseries (*query*, *minutes=30*)
Runs a timeseries query agains Scalyr with more generous rate limits. (New time series are created on the fly by Scalyr)

facets (*filter*, *field*, *max_count=5*, *minutes=30*, *prio='low'*)
This method is used to retrieve the most common values for a field.

By default the Scalyr wrapper uses https://www.scalyr.com/ as the default region. Overriding is possible using `scalyr(scalyr_region='eu')` if you want to use their Europe environment https://eu.scalyr.com/.

```
scalyr(scalyr_region='eu').count(' ERROR ')
```

9.23 SNMP

Provides a wrapper for SNMP functions listed below. SNMP checks require specifying hosts in the entities filter. The partial object *snmp()* accepts a *timeout=seconds* parameter, default is 5 seconds timeout. **NOTE**: this timeout is per answer, so multiple answers will add up and may block the whole check

memory()

```
snmp().memory()
```

Returns host's memory usage statistics. All values are in KiB (1024 Bytes).

Example check result as JSON:

```
{
    "ram_buffer": 359404,
    "ram_cache": 6478944,
    "ram_free": 20963524,
    "ram_shared": 0,
    "ram_total": 37066332,
    "ram_total_free": 22963392,
    "swap_free": 1999868,
    "swap_min": 16000,
    "swap_total": 1999868,
}
```

load()

```
snmp().load()
```

Returns host's CPU load average (1 minute, 5 minute and 15 minute averages).

Example check result as JSON:

```
{"load1": 0.95, "load5": 0.69, "load15": 0.72}
```

cpu()

```
snmp().cpu()
```

Returns host's CPU usage in percent.

Example check result as JSON:

```
{"cpu_system": 0, "cpu_user": 17, "cpu_idle": 81}
```

df()

```
snmp().df()
```

Example check result as JSON:


```
{
    "/data/postgres-wal-nfs-example": {
        "available_space": 524287840,
        "device": "example0-2-stp-123:/vol/example_pgwal",
        "percentage_inodes_used": 0,
        "percentage_space_used": 0,
        "total_size": 524288000,
        "used_space": 160,
    }
}
```


logmatch()

```
snmp().logmatch()
```

interfaces()

```
snmp().interfaces()
```

Example check result as JSON:


```
{
    "lo": {
        "in_octets": 63481918397415,
        "in_discards": 11,
        "adStatus": 1,
        "out_octets": 63481918397415,
        "opStatus": 1,
        "out_discards": 0,
        "speed": "10",
        "in_error": 0,
        "out_error": 0
    },
    "eth1": {
        "in_octets": 55238870608924,
        "in_discards": 8344,
        "adStatus": 1,
        "out_octets": 6801703429894,
        "opStatus": 1,
        "out_discards": 0,
        "speed": "10000",
        "in_error": 0,
        "out_error": 0
    },
    "eth0": {
        "in_octets": 3538944286327,
        "in_discards": 1130,
        "adStatus": 1,
        "out_octets": 16706789573119,
        "opStatus": 1,
        "out_discards": 0,
        "speed": "10000",
        "in_error": 0,
        "out_error": 0
```

```
    }
}
```

get()

```
snmp().get('iso.3.6.1.4.1.42253.1.2.3.1.4.7.47.98.105.110.47.115.104', 'stunnel',␣
↪int)
```

Example check result as JSON:

```
{
    "stunnel": 0
}
```

9.24 SQL

sql([*shard*])

Provides a wrapper for connection to PostgreSQL database and allows executing queries. All queries are executed in read-only transactions. The connection wrapper requires one parameters: list of shard connections. The shard connections must come from the entity definition (see database-entities). Example query for log database which returns a primitive long value:

```
sql().execute("SELECT count(*) FROM zl_data.log WHERE log_created > now() - '1␣
↪hour'::interval").result()
```

Example query which will return a single dict with keys `a` and `b`:

```
sql().execute('SELECT 1 AS a, 2 AS b').result()
```

The SQL wrapper will automatically sum up values over all shards:

```
sql().execute('SELECT count(1) FROM zc_data.customer').result() # will return a␣
↪single integer value (sum over all shards)
```

It's also possible to query a single shard by providing its name:

```
sql(shard='customer1').execute('SELECT COUNT(1) AS c FROM zc_data.customer').
↪results() # returns list of values from a single shard
```

It's also possible to query another database on the same server overwriting the shards information:

```
sql(shards={'customer_db' : entity['host'] + ':' + str(entity['port']) + '/
↪another_db'}).execute('SELECT COUNT(1) AS c FROM my_table').results()
```

To execute a SQL statement on all LIVE customer shards, for example, use the following entity filter:

```
[
    {
        "type":         "database",
        "name":         "customer",
        "environment":  "live",
        "role":         "master"
    }
]
```

The check command will have the form

```
>>> sql().execute('SELECT 1 AS a').result()
8
# Returns a single value: the sum over the result from all shards

>>> sql().execute('SELECT 1 AS a').results()
[{'a': 1}, {'a': 1}, {'a': 1}, {'a': 1}, {'a': 1}, {'a': 1}, {'a': 1}, {'a': 1}]
# Returns a list of the results from all shards

>>> sql(shard='customer1').execute('SELECT 1 AS a').results()
[{'a': 1}]
# Returns the result from the specified shard in a list of length one

>>> sql().execute('SELECT 1 AS a, 2 AS b').result()
{'a': 8, 'b': 16}
# Returns a dict of the two values, which are each the sum over the result from␣
↪all shards
```

The results() function has several additional parameters:

```
sql().execute('SELECT 1 AS ONE, 2 AS TWO FROM dual').results([max_results=100],␣
↪[raise_if_limit_exceeded=True])
```

max_results The maximum number of rows you expect to get from the call. If not specified, defaults to 100. You cannot have an unlimited number of rows. There is an absolute maximum of 1,000,000 results that cannot be overridden. Note: If you require processing of larger dataset, it is recommended to revisit architecture of your monitoring subsystem and possibly move logic that does calculation into external web service callable by ZMON 2.0.

raise_if_limit_exceeded Raises an exception if the limit of rows would have been exceeded by the issued query.

orasql()

Provides a wrapper for connection to Oracle database and allows executing queries. All queries are executed in read-only transactions. The connection wrapper requires three parameters: host, port and sid, that must come from the entity definition (see database-entities). One idiosyncratic behaviour to be aware, is that when your query produces more than one value, and you get a dict with keys being the column names or aliases you used in your query, you will always get back those keys *in uppercase*. For clarity, we recommend that you write all aliases and column names in uppercase, to avoid confusion due to case changes. Example query of the simplest query, which returns a single value:

```
orasql().execute("SELECT 'OK' from dual").result()
```

Example query which will return a single dict with keys `ONE` and `TWO`:

```
orasql().execute('SELECT 1 AS ONE, 2 AS TWO from dual').result()
```

To execute a SQL statement on a LIVE server, tagged with the name business_intelligence, for example, use the following entity filter:

```
[
    {
        "type":         "oracledb",
        "name":         "business_intelligence",
```

```
        "environment": "live",
        "role":        "master"
    }
]
```

exacrm()
: Provides a wrapper for connection to the CRM Exasol database executing queries. The connection wrapper requires one parameter: the query.

  Example query:

```
exacrm().execute("SELECT 'OK';").result()
```

  To execute a SQL statement on the itr-crmexa* servers use the following entity filter:

```
[
    {
        "type": "host",
         "host_role_id": "117"
    }
]
```

mysql([*shard*])
: Provides a wrapper for connection to MySQL database and allows executing queries. The connection wrapper requires one parameters: list of shard connections. The shard connections must come from the entity definition (see database-entities). Example query of the simplest query, which returns a single value:

```
mysql().execute("SELECT count(*) FROM mysql.user").result()
```

  Example query which will return a single dict with keys `h` and `u`:

```
mysql().execute('SELECT host AS h, user AS u FROM mysql.user').result()
```

  The SQL wrapper will automatically sum up values over all shards:

```
mysql().execute('SELECT count(1) FROM zc_data.customer').result() # will return a␣
↪single integer value (sum over all shards)
```

  It's also possible to query a single shard by providing its name:

```
mysql(shard='customer1').execute('SELECT COUNT(1) AS c FROM zc_data.customer').
↪results() # returns list of values from a single shard
```

  To execute a SQL statement on all LIVE customer shards, for example, use the following entity filter:

```
[
    {
        "type":        "mysqldb",
        "name":        "lounge",
        "environment": "live",
        "role":        "master"
    }
]
```

9.25 TCP

This function opens a TCP connection to a host on a given port. If the connection succeeds, it returns 'OK'. The host can be provided directly for global checks or resolved from entities filter. Assuming that we have an entity filter type=host, the example below will try to connect to every host on port 22:

```
tcp().open(22)
```

9.26 Zomcat

Retrieve zomcat instance status (memory, CPU, threads).

```
zomcat().health()
```

This would return a dict like:

```
{
    "cpu_percentage": 5.44,
    "gc_percentage": 0.11,
    "gcs_per_sec": 0.25,
    "heap_memory_percentage": 6.52,
    "heartbeat_enabled": true,
    "http_errors_per_sec": 0.0,
    "jobs_enabled": true,
    "nonheap_memory_percentage": 20.01,
    "requests_per_sec": 1.09,
    "threads": 128,
    "time_per_request": 42.58
}
```

Most of the values are retrieved via JMX:

cpu_percentage CPU usage in percent (retrieved from JMX).

gc_percentage Percentage of time spent in garbage collection runs.

gcs_per_sec Garbage collections per second.

heap_memory_percentage Percentage of heap memory used.

nonheap_memory_percentage Percentage of non-heap memory (e.g. permanent generation) used.

heartbeat_enabled Boolean indicating whether heartbeat.jsp is enabled (`true`) or not (`false`). If `/heartbeat.jsp` cannot be retrieved, the value is `null`.

http_errors_per_sec Number of Tomcat HTTP errors per second (all 4xx and 5xx HTTP status codes).

jobs_enabled Boolean indicating whether jobs are enabled (`true`) or not (`false`). If `/jobs.monitor` cannot be retrieved, the value is `null`.

requests_per_sec Number of HTTP/AJP requests per second.

threads Total number of threads.

time_per_request Average time in milliseconds per HTTP/AJP request.

9.27 Helper Functions

The following general-purpose functions are available in check commands:

abs(*number*)
Returns the absolute value of the argument. Does not have overflow issues.

```
>>> abs(-1)
1
>>> abs(1)
1
>>> abs(-2147483648)
2147483648
```

all(*iterable*)
Returns `True` if none of the elements of *iterable* are falsy.

```
>>> all([4, 2, 8, 0, 3])
False

>>> all([])
True
```

any(*iterable*)
Returns `True` if at least one element of *iterable* is truthy.

```
>>> any([None, [], '', {}, 0, 0.0, False])
False

>>> any([])
False
```

avg(*results*)
Returns the arithmetic mean of the values in *results*. Returns `None` if there are no values. *results* must not be an iterator.

```
>>> avg([1, 2, 3])
2.0

>>> avg([])
None
```

basestring()
Superclass of `str` and `unicode` useful for checking whether a value is a string of some sort.

```
>>> isinstance('foo', basestring)
True
>>> isinstance(u'', basestring)
True
```

bin(*n*)
Returns a string of the given integer in binary representation.

```
>>> bin(1000)
'0b1111101000'
```

bool(*x*)
Returns `True` if *x* is truthy, and `False` otherwise. Does not parse strings. Also usable to check whether a value is Boolean.

```
>>> bool(None)
False

>>> bool('False')
True

>>> isinstance(False, bool)
True
```

chain(**iterables*)
Returns an iterator that that yields the elements of the first iterable, followed by the elements of the second iterable, and so on.

```
>>> list(chain([1, 2, 3], 'abc'))
[1, 2, 3, 'a', 'b', 'c']

>>> list(chain())
[]
```

chr(*n*)
Returns the character for the given ASCII code.

```
>>> chr(65)
'A'
```

class Counter([*iterable-or-mapping*])
Creates a specialized `dict` for counting things. See the official Python documentation for details.

dict([*mapping*][, ***kwargs*])
Creates a new `dict`. Usually, using a literal will be simpler, but the function may be useful to copy `dicts`, to covert a list of key/value pairs to a `dict`, or to check whether some object is a `dict`.

```
>>> dict(a=1, b=2, c=3)
{'a': 1, 'c': 3, 'b': 2}

>>> dict({'a': 1, 'b': 2, 'c': 3})
{'a': 1, 'c': 3, 'b': 2}   # This is a copy of the original dict.

>>> dict([['a', 1], ['b', 2], ['c', 3]])
{'a': 1, 'c': 3, 'b': 2}

>>> isinstance({}, dict)
True
```

divmod(x, y):
Performs integer division and modulo as a single operation.

```
>>> divmod(23, 5)
(4, 3)
```

empty(*v*)
Indicates whether *v* is falsy. Equivalent to `not v`.

```
>>> empty([])
True

>>> empty([0])
False
```

enumerate (*iterable*[, *start=0*])
Generates tuples `(start + 0, iterable[0]), (start + 1, iterable[1]), ...`. Useful to have access to the index in a loop.

```
>>> list(enumerate(['a', 'b', 'c'], start=1))
[(1, 'a'), (2, 'b'), (3, 'c')]
```

filter (*function*, *iterable*)
Returns a list of all objects in *iterable* for which *function* returns a truthy value. If function is `None`, the returned list contains all truthy objects in *iterable*.

```
>>> filter(lambda n: n % 3, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
[1, 2, 4, 5, 7, 8, 10]

>>> filter(None, [False, None, 0, 0.0, '', [], {}, 1000])
[1000]
```

float (*x*)
Returns *x* as a floating-point number. Parses stings.

```
>>> float('2.5')
2.5

>>> float('-inf')
-inf

>>> float(2)
2.0
```

This is useful to force proper division:

```
>>> 2 / 5
0

>>> float(2) / 5
0.4
```

Also usable to check whether a value is a floating-point number:

```
>>> isinstance(2.5, float)
True

>>> isinstance(2, float)
False
```

groupby (*iterable*[, *key*])
A somewhat obscure function for grouping consecutive equal elements in an iterable. See the official Python documentation for more details.

```
>>> [(k, list(v)) for k, v in groupby('abba')]
[('a', ['a']), ('b', ['b', 'b']), ('a', ['a'])]
```

hex(*n*)
Returns a string of the given integer in hexadecimal representation.

```
>>> hex(1000)
'0x3e8'
```

int(*x*[, *base*])
Returns *x* as an integer. Truncates floating-point numbers and parses strings. Also usable to check whether a value is an integer.

```
>>> int(2.5)
2

>>> int(-2.5)
2

>>> int('2')
2

>>> int('abba', 16)
43962

>>> isinstance(2, int)
True
```

isinstance(*object*, *classinfo*)
Indicates whether *object* is an instance of the given class or classes.

```
>>> isinstance(2, int)
True

>>> isinstance(2, (int, float))
True

>>> isinstance('2', int)
False
```

json(*s*)
Converts the given *JSON* string to a Python object.

```
>>> json('{"list": [1, 2, 3, 4]}')
{u'list': [1, 2, 3, 4]}
```

jsonpath_flat_filter(*obj*, *path*)
Executes json path expression using jsonpath_rw and returns a flat dict of (full_path, value).

```
>>> data = {"timers":{"/api/v1/":{"m1.rate": 12, "99th": "3ms"}}}
>>> jsonpath_flat_filter(data, "timers.*.*.'m1.rate'")
{"timers./api/v1/.m1.rate": 12}
```

jsonpath_parse(*path*)
Creates a json path parse object from the jsonpath_rw to be used in your check command.

len(*s*)
Returns the length of the given collection.

```
>>> len('foo')
3
```

```
>>> len([0, 1, 2])
3

>>> len({'a': 1, 'b': 2, 'c': 3})
3
```

list (*iterable*)
Creates a new list. Usually, using a literal will be simpler, but the function may be useful to copy lists, to covert some other iterable to a list, or to check whether some object is a list.

```
>>> list({'a': 1, 'b': 2, 'c': 3})
['a', 'c', 'b']

>>> list(chain([1, 2, 3], 'abc'))
[1, 2, 3, 'a', 'b', 'c']   # Without the list call, this would be a chain object.

>>> isinstance([1, 2, 3], list)
True
```

long (*x*[, *base*])
Converts a number or string to a long integer.

```
>>> long(2.5)
2L

>>> long(-2.5)
-2L

>>> long('2')
2L

>>> long('abba', 16)
43962L
```

map (*function*, *iterable*)
Calls *function* on each element of *iterable* and returns the results as a list.

```
>>> map(lambda n: n**2, [0, 1, 2, 3, 4, 5])
[0, 1, 4, 9, 16, 25]
```

max (*iterable*)
Returns the greatest element of *iterable*. With two or more arguments, returns the greatest argument instead.

```
>>> max([2, 4, 1, 3])
4

>>> max(2, 4, 1, 3)
4
```

min (*iterable*)
Returns the smallest element of *iterable*. With two or more arguments, returns the smallest argument instead.

```
>>> min([2, 4, 1, 3])
1
```

```
>>> min(2, 4, 1, 3)
1
```

normalvariate (*mu*, *sigma*)
Returns a normally distributed random variable with the given mean and standard derivation.

```
>>> normalvariate(0, 1)
-0.1711153439880709
```

oct (*n*)
Returns a string of the given integer in octal representation.

```
>>> oct(1000)
'01750'
```

ord (*n*)
Returns the ASCII code of the given character.

```
>>> ord('A')
65
```

pow (*x*, *y*[, *z*])
Computes *x* to the power of *y*. The result is modulo *z*, if *z* is given, and the function is much, much faster than `(x ** y) % z` in that case.

```
>>> pow(56876845793546543243783543735425734536873,␣
↪12425445412439354354394354397364398364378, 10)
9L
```

range ([*start*], *stop*[, *step*])
Returns a list of integers `[start, start + step * 1, start + step * 2, ...]` where all integers are less than *stop*, or greater than *stop* if *step* is negative.

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(1, 11)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> range(1, 1)
[]
>>> range(11, 1)
[]
>>> range(0, 10, 3)
[0, 3, 6, 9]
>>> range(10, -1, -1)
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

reduce (*function*, *iterable*[, *initializer*])
Calls `function(r, e)` for each element `e` in *iterable*, where `r` is the result of the last such call, or *initializer* for the first such call. If *iterable* has no elements, returns *initializer*.

If *initializer* is ommitted, the first element of *iterable* is removed and used in place of *initializer*. In that case, an error is raised if *iterable* has no elements.

```
>>> reduce(lambda a, b: a * b, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 1)
3628800  # 10!
```

Note: Because of a Python bug, `reduce` used to be unreliable. This issue should now be fixed.

reversed(*iterable*)
Returns an iterator that iterates over the elements in *iterable* in reverse order.

```
>>> list(reversed([1, 2, 3]))
[3, 2, 1]
```

round(*n*[, *digits=0*])
Rounds the given number to the given number of digits, rounding half away from zero.

```
>>> round(23.4)
23.0
>>> round(23.5)
24.0
>>> round(-23.4)
-23.0
>>> round(-23.5)
-24.0
>>> round(0.123456789, 3)
0.123
>>> round(987654321, -3)
987654000.0
```

set(*iterable*)
Returns a set built from the elements of *iterable*. Useful to remove duplicates from some collection.

```
>>> set([1, 2, 1, 4, 3, 2, 2, 3, 4, 1])
set([1, 2, 3, 4])
```

sorted(*iterable*[, *reverse*])
Returns a sorted list containing the elements of *iterable*.

```
>>> sorted([2, 4, 1, 3])
[1, 2, 3, 4]

>>> sorted([2, 4, 1, 3], reverse=True)
[4, 3, 2, 1]
```

str(*object*)
Returns the string representation of *object*. Also usable to check whether a value is a string. If the result would contain Unicode characters, the *unicode()* function must be used instead.

```
>>> str(2)
'2'

>>> str({'a': 1, 'b': 2, 'c': 3})
"{'a': 1, 'c': 3, 'b': 2}"

>>> isinstance('foo', str)
True
```

sum(*iterable*)
Returns the sum of the elements of *iterable*, or 0 if *iterable* is empty.

```
>>> sum([1, 2, 3, 4])
10

>>> sum([])
0
```

time([*spec*][, *utc*])
Given a time specification such as `'-10m'` for "ten minutes ago" or `'+3h'` for "in three hours", returns an object representing that timestamp. Valid units are `s` for seconds, `m` for minutes, `h` for hours, and `d` for days.

The time specification *spec* can also be a Unix epoch/timestamp or a valid ISO timestamp in of the following formats: `YYYY-MM-DD HH:MM:SS.mmmmm`, `YYYY-MM-DD HH:MM:SS`, `YYYY-MM-DD HH:MM` or `YYYY-MM-DD`.

If *spec* is omitted, the current time is used. If *utc* is True the timestamp uses UTC, otherwise it uses local time.

The returned object has two methods:

isoformat([*sep*])
Returns the timestamp as a string of the form `YYYY-MM-DD HH:MM:SS.mmmmmm`. The default behavior is to omit the `T` between date and time. This can be overridden by passing the optional *sep* parameter to the method.

```
>>> time('+4d').isoformat()
'2014-03-29 18:05:50.098919'

>>> time(1396112750).isoformat()
'2014-03-29 18:05:50'

>>> time('+4d').isoformat('T')
'2014-03-29T18:05:50.098919'
```

format(*fmt*)
Returns the timestamp as a string formatted according to the given format. See the official Python documentation for an incomplete list of supported format directives.

Additionally, the subtraction operator is overloaded and returns the time difference in seconds:

```
>>> time('2014-01-01 01:13') - time('2014-01-01 01:01')
12
```

timestamp()
Returns Unix time stamp. This wraps time.time()

tuple(*iterable*)
Returns the given iterable as a tuple (an immutable list, basically). Also usable to check whether a value is a tuple.

```
>>> tuple([1, 2, 3])
(1, 2, 3)
>>> isinstance((1, 2, 3), tuple)
True
```

unicode(*object*)
Returns the string representation of *object* as a Unicode string. Also usable to check whether a value is a Unicode string.

```
>>> unicode({u'α': 1, u'β': 2, u'γ': 3})
u"{u'\\u03b1': 1, u'\\u03b3': 3, u'\\u03b2': 2}"

>>> isinstance(u'', unicode)
True
```

unichr(*n*)
Returns the unicode character with the given code point. Might be limited to code points less than 0x10000.

```
>>> unichr(0x2a13)  # LINE INTEGRATION WITH SEMICIRCULAR PATH AROUND POLE
u''
```

xrange([*start*], *stop*[, *step*])
As *range()*, but returns an iterator rather than a list.

zip(**iterables*)
Returns a list of tuples where the *i*-th tuple contains the *i*-th element from each of the given iterables. Uses the lowest length if the iterables have different lengths.

```
>>> zip(['a', 'b', 'c'], [1, 2, 3])
[('a', 1), ('b', 2), ('c', 3)]
>>> zip(['A', 'B', 'C'], ['a', 'b', 'c'], [1, 2, 3])
[('A', 'a', 1), ('B', 'b', 2), ('C', 'c', 3)]
>>> zip([], [1, 2, 3])
[]
```

re()
Python regex `re` module for all regex operations.

```
>>> re.match(r'^ab.*', 'a123b') != None
False

>>> re.match(r'^ab.*', 'ab123') != None
True
```

math()
Python `math` module for all math operations.

```
>>> math.log(4, 2)
2.0
```

CHAPTER 10

Alert Functions Reference

10.1 Time Specifications

Whenever one of these functions takes an argument named `time_spec`, that argument is a string of the form *<magnitude><unit>*, where *<magnitude>* is an positive integer, and *<unit>* is one of `s` (for seconds), `m` (for minutes), `h` (for hours), and `d` (for days).

Therefore, a value of `5m` would indicate that all values gathered in the last five minutes should be taken into account.

Note: Trial Run doesn't provide any previous values. Please check how functions depending on check values behave in case values were not available.

10.2 Timeseries functions

All of the `timeseries_*` functions below additionally accept a named parameter `key=func` which can be used to extract the wanted value from a dict or an array. To get the value of the key `my-key` from a dict, you can use e.g.

```
res = timeseries_sum('5m', key=lambda x: x.get('my-key', 0))
```

Note: The values for the `timeseries_*` functions are retrieved from the local redis instance. By default the last 20 check results are kept in this instance. Time ranges which exceed 20 times the check interval will lead to unexpected results.

10.3 Previous Check results

The data source for the `alert_series` and `value_series` is the same as for the `timeseries_*` functions. Both functions return **up to** the requested number of results - as much as data is available. By default the maximum is 20 (see the above note for the timeseries functions).

10.4 Alert condition functions

The following functions are available in the alert condition expression:

alert_series (*f*[, *n=1*])
Returns True if function f either raises exception or returns True for the last n check values for the given entity. Use this function to build an alert that only is raised if the last n intervals are up. This can solve alert where you face flapping due to technical issues.

```
# check that the value is bigger than 5 the last 3 runs
alert_series(lambda v: v > 5, 3)
```

Note: If number of check values is less than `n`, then `f` will be evaluated for those values and alerts could be raised accordingly.

capture (*value*)
capture (*name=value*)
Saves the given value as a capture, and returns it unaltered. In the first form, the capture receives a generated name (`capture_N`). In the second form, the specified name is used as the name of the capture.

Example: `capture(foo=1)` saves the value `1` in a capture named `foo` and returns `1`.

entity_results ()
List for every entity containing a dict with the following keys: `value` (the most recent value for the alert's check on that entity), `ts` (the time when the check evaluation was started, in seconds since the epoch, as a floating-point number), and `td` (the check's duration, in seconds, as a floating-point number). Works regardless of the type of value. DOES NOT WORK in Trial Run right now!

entity_values ()
Returns a list for each entity containing the most recent value for the alert's check on that entity. Works regardless of the type of value. DOES NOT WORK in Trial Run right now!

monotonic ([*count=2*, *increasing=True*, *strictly=False*, *data=None*])
Returns true if the values in `data` are (strictly) monotonic increasing / decreasing values. When `data` is not given, uses the result of `value_series(count)` as data (only works for checks returning a single value).

```
# check that the value of `some_key` is monotonic increasing for the last 5␣
↪checks (including this one)
monotonic(data=[v.get('some_key', 0) for v in value_series(5)])
```

Note: The order of the `data` is expected to have the latest value first and the oldest last

timeseries_avg (*time_spec*)
The arithmetic mean of the check values gathered in the specified time period. Returns `None` if there are no values. Only works for numeric values.

Example: The check has gathered the values 5, 12, 14, 13, and 6 over the last five minutes. `timeseries_avg('5m')` is (5 + 12 + 14 + 13 + 6) / 5 = 10.

`timeseries_median`(*time_spec*)
The median of the check values gathered in the specified time period. If the number of such values is even, the arithmetic mean of the two middle values is returned. Returns `None` if there are no values. Equivalent to `timeseries_percentile(time_spec, 0.5)`. Only works for numeric values.

Example 1: The check has gathered the values 5, 12, 14, 13, and 6 over the last five minutes. Sorting these values gives 5, 6, 12, 13, 14. The middle value is 12. Therefore, `timeseries_median('5m')` is 12.

Example 2: The check has gathered the values 12, 14, 13, and 6 over the last four minutes. Sorting these values gives 6, 12, 13, 14. The two middle values are 12 and 13. Therefore, `timeseries_median('4m')` is (12 + 13) / 2 = 12.5.

`timeseries_percentile`(*time_spec*, *percent*)
The *P*-th percentile of the values gathered in the specified time period, where *P* = *percent* × 100, using linear interpolation. Only works for numeric values.

The *P*-th percentile of *N* values is $V(K) + (V(K)\ \ V(K)) \times (K\ \ K)$, where $K = (N\ \ 1) \times P / 100$ and $V(I)$ for I in [0, *N*) is the *I*-th element of the list of values sorted in ascending order. Returns `None` if there are no values.

Example 1: The check has gathered the values 5, 12, 14, 13, and 6 over the last five minutes. Sorting these values gives 5, 6, 12, 13, 14. Let *P* = 30. There are *N* = 5 values, and $K = (N\ \ 1) \times P / 100 = (5\ \ 1) \times 30 / 100 = 1.2$. The value at index 1.2 = 1 is 6, and the value at index 1.2 = 2 is 12. Therefore, `timeseries_percentile('5m', 0.3)` is $6 + (12\ \ 6) \times (1.2\ \ 1.2) = 7.2$.

Example 2: The check has gathered the values 5, 12, 14, 13, and 6 over the last five minutes. Sorting these values gives 5, 6, 12, 13, 14. Let *P* = 25. There are *N* = 5 values, and $K = (N\ \ 1) \times P / 100 = (5\ \ 1) \times 25 / 100 = 1.\ 1 = 1 = 1$. The value at index 1 is 6. Therefore, `timeseries_percentile('5m', 0.25)` is $6 + (6\ \ 6) \times (1\ \ 1) = 6$.

`timeseries_first`(*time_spec*)
The oldest value among the values gathered in the specified time period. Returns `None` if there are no values. Works regardless of the type of value.

Example: The check has gathered the values 5, 12, 14, 13, and 6 over the last five minutes. The oldest value is 5. Therefore, `timeseries_first('5m')` is 5.

`timeseries_delta`(*time_spec*)
The newest value among the values gathered in the specified time period minus the oldest one. Returns `0` if there are no values. Only works for numeric values.

Example 1: The check has gathered the values 5, 12, 14, 13, and 6 over the last five minutes. The newest value is 6 and the oldest value is 5. Therefore, `timeseries_delta('5m')` is $6\ \ 5 = 1$.

Example 2: The check has gathered the values 12, 14, 13, and 6 over the last four minutes. The newest value is 6 and the oldest value is 12. Therefore, `timeseries_delta('4m')` is $6\ \ 12 = 6$ (not 6).

`timeseries_min`(*time_spec*)
The smallest value among the values gathered in the specified time period. Returns `None` if there are no values. Works regardless of the type of value, but is unlikely to be particularly useful for non-numeric values.

Example: The check has gathered the values 5, 12, 14, 13, and 6 over the last five minutes. The smallest value is 5. Therefore, `timeseries_min('5m')` is 5.

`timeseries_max`(*time_spec*)
The largest value among the values gathered in the specified time period. Returns `None` if there are no values. Works regardless of the type of value, but is unlikely to be particularly useful for non-numeric values.

Example: The check has gathered the values 5, 12, 14, 13, and 6 over the last five minutes. The largest value is 14. Therefore, `timeseries_max('5m')` is 14.

timeseries_sum(*time_spec*)
The sum of the values gathered in the specified time period. Returns `0` if there are no values. Only works for numeric values.

Example: The check has gathered the values 5, 12, 14, 13, and 6 over the last five minutes. Therefore, `timeseries_sum('5m')` is 5 + 12 + 14 + 13 + 6 = 50.

value_series(*[n=1]*)
Returns the last n values for the underlying checks and the current entity. Return `[]` if there are no values.

10.5 History distance functionality

The history distance functionality currently only works for numeric values, and not for structured ones, or arrays. Call for a DistanceWrapper object.

```
history().distance([weeks=4], [bin_size='1h'], [snap_to_bin = True], [dict_extractor_
↪path=''])
```

An object will be returned, where you can call additional functions on. The default parameters should be good for most cases, but in case you'd like to change them:

weeks Changes how far you'd like to look into the past. It is good to average more than one week, since you might have seen something unusual a week ago, and I assume you would like to get warned in the next week if something similar happens.

bin_size Defines the size of the bins you are using to aggregate the history. Defaults to 1h. Is a `time_spec`. See the next parameter for an explanation of the bins.

snap_to_bin Determines wether you'd like to have sliding bins, or fixed bin start points. Consider the following example: You run your check at monday, 10.30 AM. If `snap_to_bin` is `True`, you would gather data from the past 4 weeks, every monday from 10 AM to 11 AM, and then calculate the mean and standard deviation to use in the functions below. If the value is `snap_to_bin` is `False`, you would gather data from every monday, 9.30 AM to 10.30 AM.

Setting the value to `True` allows for some internal caching of already-calculated values for a bin, since the mean and standard deviation don't change for about an hour, so you don't stress the network and servers as much as with having it set to `False`. **Attention:** Caching optimizations for `snap_to_bin` not yet implemented. Please use it nevertheless, so that we can benefit from optimizations in the future.

dict_extractor_path Takes a string that is used for accessing the `value` if it is not a scalar value, but a dict. Normally, the history functionality only works for scalar values. Using this access string, you can use structured values, too. The *dict_extractor_path* is of the form 'a.b.c' for a dict with the structure {'a':{'b':{'c':5}}} to extract the value 5. Effectively, you use the *dict_extractor_path* to boil a structured check value down to a scalar value. The *dict_extractor_path* is applied on the historic values, and on the parameters of the `sigma()` and `absolute()` functions.

Example: Your check gives you a map of data instead of a single value: `{"CREDITCARD": 25, "PAYPAL": 10, "MAK": 10, "PTF": 30}` which contains the number of requests for the payment methods CREDITCARD, PAYPAL, MAKSUTURVA and PRZELEWY24 of the last few minutes. If you want to check the history of Paypal orders, take this one:

```
history().distance(dict_extractor_path = 'PAYPAL').sigma(value) < 2.0
```

which will take a look at the history of Paypal orders only and warn you if there is something unusual (too low number of requests). An even better query would be:

```
capture(suspect_payment_methods=
    {
        k: value[k]
        for k,v in
            {
                payment_method: history().distance(dict_extractor_path = payment_
↪method).sigma(value)
                for payment_method in value.keys()
            }.items()
        if v < -2.0
    }
)
```

which takes a look at the history of every payment method and then tells you in a capture which payment methods are suspect and should be looked at manually.

Attention: Some structured values are not written to the history (when they are too complex). If you have trouble, try to change your check to return less complex values. Lists are currently not supported.

absolute(*value*)
Returns the absolute distance of the actual value to the history of the check that is linked to this function. The absolute distance is just the difference of the value provided and the mean of the history values.

Example: You can use it e.g. to warn when you get 5 more exceptions than you would get on average:

```
history().distance().absolute(value) < 5
```

The distance is directed, which means that you will not get warned if you get "too little" exceptions. You can use abs() to get an undirected value.

sigma(*value*)
Returns the distance of the actual value to the history of the check, normalized by the standard deviation.

Example: You can use it e.g. to get warned when you get more exceptions than usual:

```
history().distance().sigma(value) < 2.0
```

This check warns you in 4% of all cases on average. You will not be warned if there are some small spikes in the exception count, but you will be warned if there are spikes that are twice as far away from the mean as what is usual.

The distance is directed, which means that you will not get warned if you get "too little" exceptions. You can use abs() to get an undirected value.

bin_mean()
Returns the mean of the bins that were aggregated.

bin_standard_deviation()
Returns the standard deviation of the bins that were aggregated.

10.6 Additional helper functions

You can also use some additional functions that are used in check commands.

- *time()*
- kairosdb()

CHAPTER 11

Notifications Reference

ZMON provides several means of notification in case of alerts. Notifications will be triggered when alert status change. Please refer to *Notification options* for different worker configuration options.

11.1 Hipchat

Notify Hipchat room with alert status.

send_hipchat(*room=None*, *message=None*, *token=None*, *notify=False*, *color='red'*, *link=False*, *link_text='go to alert'*)

Send Hipchat notification to specified room.

Parameters

- **room** (str) – Room to be notified.
- **message** (str) – Message to be sent. If None, then a message constructed from the alert will be sent.
- **token** (str) – Hipchat API token.
- **notify** (bool) – Hipchat notify flag. Default is False.
- **color** (str) – Message color. Default is red if alert is raised.
- **link** (bool) – Add link to Hipchat message. Default is False.
- **link_text** (str) – if link param is True, this will be displayed as a link in the hipchat message. Default is go to alert.

Note: Message color will be determined based on alert status. If alert has ended, then color will be green, otherwise color argument will be used.

Example message:

```
{
    "message": "NEW ALERT: Requests failing with status 500 on host-production-1-
↪entity",
    "color": "red",
    "notify": true,
}
```

11.2 HTTP

Provides notification by invoking HTTP call to certain endpoint. HTTP notification uses `POST` method when invoking the call.

notify_http(*url=None*, *body=None*, *params=None*, *headers=None*, *timeout=5*, *oauth2=False*, *include_alert=True*)

Send HTTP notification to specified endpoint.

Parameters

- **url** (`str`) – HTTP endpoint URL. If not passed, then default URL will be used in worker configuration.
- **body** (`dict`) – Request body.
- **params** (`dict`) – Request URL params.
- **headers** (`dict`) – HTTP headers.
- **timeout** (`int`) – Request timeout. Default is 5 seconds.
- **oauth2** (`bool`) – Add OAUTH2 authentication headers. Default is False.
- **include_alert** (`bool`) – Include alert data in request body. Default is `True`.

Example:

```
notify_http('https://some-notification-service/alert', body={'zmon': True},␣
↪headers={'X-TOKEN': 1234})
```

Note: If `include_alert` is `True`, then request body will include alert data. This is usually useful, since it provides valuable info like `is_alert` and `changed` which can indicate whether the alert has **started** or **ended**.

```
{
    "body": null,
    "alert": {
        "is_alert": true,
        "changed": true,
        "duration": 2.33,
        "captures": {},
        "entity": {"type": "GLOBAL", "id": "GLOBAL"},
        "worker": "plocal.zmon",
        "value": {"td": 0.00037, "worker": "plocal.zmon", "ts": 1472032348.665247,
↪"value": 51.67797677979191},
        "alert_def": {
            "name": "Random Example Alert", "parameters": null, "check_id": 4,
↪"entities_map": [], "responsible_team": "ZMON", "period": "", "priority": 1,
            "notifications": ["notify_http()"], "team": "ZMON", "id": 3, "condition":
↪">40"
```

```
        }
    }
}
```

11.3 Hubot

Send Hubot notification.

notify_hubot(*queue*, *hubot_url*, *message=None*)
Send Hubot notification.

Parameters

- **queue** (str) – Hubot queue.
- **hubot_url** (str) – Hubot url.
- **message** (str) – Notification message.

11.4 Mail

Send email notifications.

send_mail(*subject=None*, *cc=None*, *html=False*, *hide_recipients=True*, *include_value=True*, *include_definition=True*, *include_captures=True*, *include_entity=True*, *per_entity=True*)
Send email notification.

Parameters

- **subject** (str) – Email subject.
- **cc** (list) – List of CC recipients.
- **html** (bool) – HTML email.
- **hide_recipients** (bool) – Hide recipients. Will be sent as BCC.
- **include_value** (bool) – Include alert value in notification message.
- **include_definition** (bool) – Include alert definition details in notification message.
- **include_captures** (bool) – Include alert captures in message.
- **include_entity** (bool) – Include affected entities in notification message.
- **per_entity** (bool) – Send new email notification per entity. Default is True.

Note: send_email is an alias for this notification function.

11.5 Opsgenie

Notify Opsgenie of a new alert status. If alert is **active**, then a new opsgenie alert will be created. If alert is **inactive** then the alert will be closed.

notify_opsgenie(*message=''*, *teams=None*, *per_entity=False*, *priority=None*, *include_alert=True*, *description=''*, ***kwargs*)

Send notifications to Opsgenie.

Parameters

- **message** (str) – Alert message. If empty, then a message will be generated from the alert data.
- **teams** (*str | list*) – Opsgenie teams to be notified. Value can be a single team or a list of teams.
- **per_entity** (bool) – Send new alert per entity. This affects the alias value and impacts how de-duplication is handled in Opsgenie. Default is False.
- **priority** (str) – Set Opsgenie priority for this notification. Valid values are P1, P2, P3, P4 or P5.
- **include_alert** (bool) – Include alert data in alert body details. Default is True.
- **description** (str) – An optional description. If present, this is inserted into the opsgenie alert description field.

Example:

```
notify_opsgenie(teams=['zmon', 'ops'], message='Number of failed requests is too␣
↪high!', include_alert=True)
```

Note: If priority is not set, then ZMON will set the priority according to the alert priority.

11.6 Pagerduty

Notify Pagerduty of a new alert status. If alert is **active**, then a new pagerduty incident with type trigger will be sent. If alert is **inactive** then incident type will be updated to resolve.

Note: Pagerduty notification plugin uses API v2.

notify_pagerduty(*message=''*, *per_entity=False*, *include_alert=True*, *routing_key=None*, *alert_class=None*, *alert_group=None*, ***kwargs*)

Send notifications to Pagerduty.

Parameters

- **message** (str) – Incident message. If empty, then a message will be generated from the alert data.
- **per_entity** (bool) – Send new alert per entity. This affects the dedup_key value and impacts how de-duplication is handled in Pagerduty. Default is False.
- **include_alert** (bool) – Include alert data in incident payload custom_details. Default is True.
- **routing_key** (str) – Pagerduty service routing_key. If not specified, then the *service key configured* for the worker will be used.
- **alert_class** (str) – Set the Pagerduty incident class.

- **alert_group** (str) – Set the Pagerduty incident group.

Example:

```
notify_pagerduty(message='Number of failed requests is too high!', include_
↪alert=True, alert_class='API health', alert_group='production')
```

11.7 Push

Send push notification via ZMON notification service.

send_push(*url=None*, *key=None*, *message=None*)
Send Push notification to mobile devices.

Parameters

- **url** (str) – Notification service base URL.
- **key** (str) – Notification service API key.
- **message** (str) – Message to be sent in notification.

Note: If Message is None then it will be generated from alert status.

11.8 Slack

Notify Slack channel with alert status. A webhook is required for notifications.

notify_slack(*webhook=None*, *channel='#general'*, *message=None*)
Send Slack notification to specified channel.

Parameters

- **webhook** (str) – Slack webhook. If not set, then webhook set in configuration will be used.
- **channel** (str) – Channel to be notified. Default is #general.
- **message** (str) – Message to be sent. If None, then a message constructed from the alert will be sent.

11.9 Twilio

Use Twilio to receive phone calls if alerts pop up. This includes basic ACK and escalation. Requires account at Twilio and the notifiction service deployed. Low investment to get going though. WORK IN PROGRESS.

notifiy_twilio(*numbers=[]*, *message="ZMON Alert Up: Some Alert"*)
Make phone call to supplied numbers. First number will be called immediately. After two minutes, another call is made to that number if no ACK. Other numbers follow at 5min interval without ACK.

Parameters

- **message** (str) – Message to be sent. If None, then a message constructed from the alert will be sent.

- **numbers** – Numbers to call

Note: Remember to configure your worker for this.

```
NOTIFICATION_SERVICE_URL
NOTIFICATION_SERVICE_KEY
```

CHAPTER 12

Monitoring on AWS

This section assumes that you're running zmon-aws-agent, which automatically discovers your EC2 instances, auto-scaling of groups, ELBs, and more.

ZMON AWS agent syncs the following entities from AWS infrastructure:

- EC2 instances
- Auto-Scaling groups
- ELBs (classic and ELBv2)
- Elasticaches
- RDS instances
- DynamoDB tables
- IAM/ACM certificates

Note: ZMON AWS Agent can be also deployed via a single appliance, which runs AWS Agent, ZMON worker and ZMON scheduler.

12.1 CloudWatch Metrics

You can achieve most basic monitoring with AWS CloudWatch. CloudWatch EC2 metrics contain the following information:

- CPU Utilization
- Network traffic
- Disk throughput/operations per second (only for ephemeral storage; EBS volumes are not included)

ZMON allows querying arbitrary CloudWatch metrics using the cloudwatch() wrapper.

12.2 Security Groups

Depending on your AWS setup, you'll probably have to open particular ports/instances to access from ZMON. Using a limited set of ports to expose management APIs and the Prometheus node exporter will make your life easier. ZMON allows parsing of Prometheus metrics via the http().prometheus().

You can deploy ZMON into each of your AWS accounts to allow cross-team monitoring and dashboards. Make sure that your security groups allow ZMON to connect to port 9100 of your monitored instances.

Not having the proper security groups configured is mainly visible by not getting the expected results at all, as packages are dropped by the EC2 instance rather then e.g. getting a connection refused.

12.3 Low-Level or Basic Properties

12.3.1 EC2 Instances

Having enough **diskspace** on your instance is important; here's a sample check. By default, you can only get space used from CloudWatch. Using Amazon's own script, you can push free space to CloudWatch and pull this data via ZMON. Alternatively, you can run the Prometheus Node exporter to pull disk space data from the EC2 node itself via HTTP.

Similarly, you can pull CPU-related metrics from CloudWatch. The Prometheus Node exporter also exposes these metrics.

You also need enough available **INodes**.

Regarding **memory**, you can either query via CloudWatch, use Prometheus Node exporter to feed ZMON, or go with low-level `snmp()` [not recommended].

The following block shows *part* of EC2 instance entity properties:

```
id: a-app-1-2QBrR1[aws:123456789:eu-west-1]
type: instance
aws_id: i-87654321
created_by: agent
host: 172.33.173.201
infrastructure_account: aws:123456789
instance_type: t2.medium
ip: 172.33.173.201
ports:
  '5432': 5432
  '8008': 8008
region: eu-west-1
```

An example check using cloudwatch wrapper and entity properties would look like the following:

```
cloudwatch().query_one({'InstanceId': entity['aws_id']}, 'CPUUtilization', 'Average',
↪'AWS/EC2', period=120)
```

12.3.2 Elastic Load Balancers

You can query AWS CloudWatch to get ELB-specific metrics. The ZMON agent will put data into the ELB entity, allowing you to monitor instance and healthy instance count.

```
id: elb-a-app-1[aws:123456789:eu-west-1]
type: elb
elb_type: classic
active_members: 1
created_by: agent
dns_name: internal-a-app-1.eu-west-1.elb.amazonaws.com
host: internal-a-app-1.eu-west-1.elb.amazonaws.com
infrastructure_account: aws:123456789
members: 3
region: eu-west-1
scheme: internal
```

ZMON AWS agent will detect both ELBs, classic and application load balancers. Both ELBs entities will be created in ZMON with `type:elb`. In order to distinguish between them in your checks, there is another property `elb_type` which holds either `classic` or `application`.

Since Cloudwatch metrics are different for each ELB type, please check CloudWatch ELB metrics for detailed reference. An example check using Cloudwatch wrapper and entity properties would look like the following:

```
# Classic ELB
lb_name = entity['name']
key = 'LoadBalancerName'
namespace = 'AWS/ELB'

# Check if Application ELBv2 entity
if entity.get('elb_type') == 'application':
    lb_name = entity['cloudwatch_name']
    key = 'LoadBalancer'
    namespace = 'AWS/ApplicationELB'

cloudwatch().query_one({key: lb_name}, 'RequestCount', 'Sum', namespace)
```

Note: ELB entities contain a special flag `dns_traffic` which is an indicator about the load balancer being actively serving traffic.

12.3.3 Auto-Scaling Groups

ZMON's agent creates an auto-scaling group entity that provides you with the number of desired instances and the number of instances in a healthy state. This enables you to monitor whether the ASG actually works and hosts spawn into a productive state.

```
id: asg-proxy-1[aws:123456789:eu-central-1]
type: asg
name: proxy-1
created_by: agent
desired_capacity: 2
dns_traffic: 'true'
dns_weight: 200
infrastructure_account: aws:123456789
instances:
- aws_id: i-123456
  ip: 172.33.109.201
- aws_id: i-654321
  ip: 172.33.109.202
```

```
max_size: 4
min_size: 2
region: eu-central-1
```

12.3.4 RDS Instances

ZMON AWS agent will detect RDS instances and store them as entities with type `database`.

```
id: rds-db-1[aws:123456789]
type: database
name: db-1
created_by: agent
engine: postgres
host: db-1.rds.amazonaws.com
infrastructure_account: aws:123456789
port: 5432
region: eu-west-1
```

```
cloudwatch().query_one({'DBInstanceIdentifier': entity['name']}, 'DatabaseConnections
↪', 'Sum', 'AWS/RDS')
```

12.3.5 ElastiCache Redis

Elasticache instances are stored as entities with type `elc`.

```
id: elc-redis-1[aws:123456789:eu-central-1]
type: elc
cluster_id: all-redis-001
cluster_num_nodes: 1
created_by: agent
engine: redis
host: redis-1.cache.amazonaws.com
infrastructure_account: aws:123456789
port: 6379
region: eu-central-1
```

12.3.6 IAM/ACM Certificates

ZMON AWS agent will also sync IAM/ACM SSL certificates, with type `certificate`. Certificate entities could be used to create an alert in case a certificate is about to expire for instance.

```
id: cert-acm-example.org[aws:123456789:eu-central-1]
type: certificate
name: '*.example.org'
status: ISSUED
arn: arn:aws:acm:eu-central-1:123456789:certificate/123456-123456-123456-123456
certificate_type: acm
created_by: agent
expiration: '2017-07-28T12:00:00+00:00'
infrastructure_account: aws:123456789
region: eu-central-1
```

12.4 Application API Monitoring

When monitoring an application, you'll usually want to check the number of received requests, latency patterns, and the number of returned status codes. These data points form a pretty clear picture of what is going on with the application.

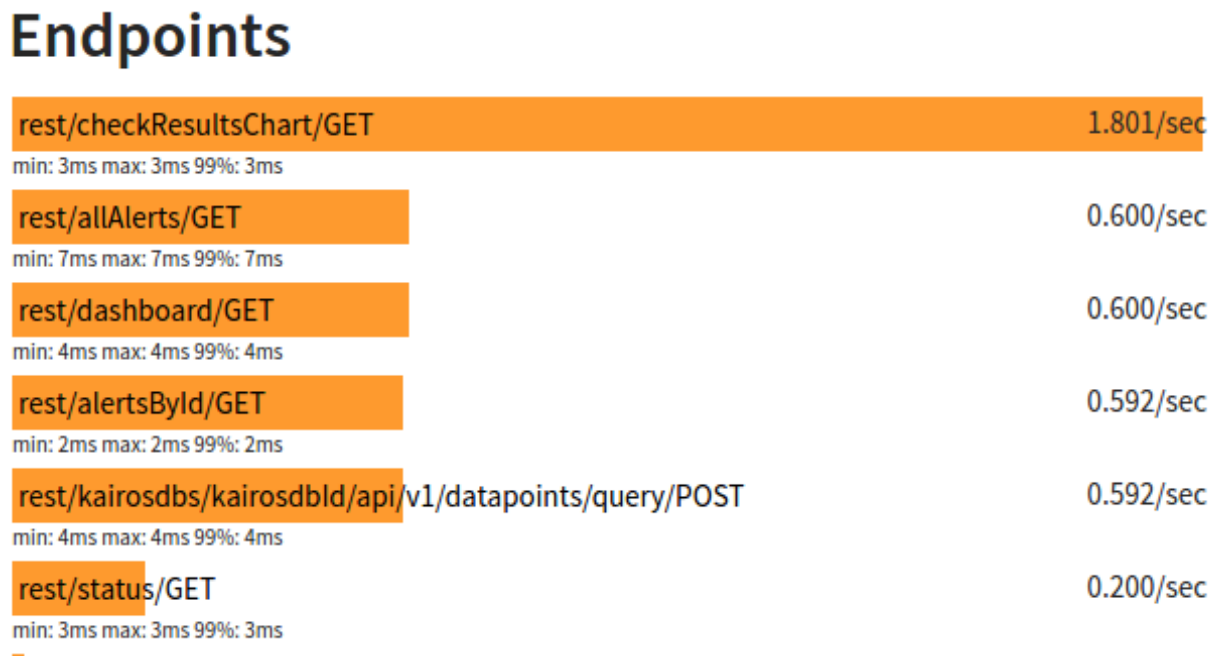
Additional metrics will help you find problems as well as opportunities for improvement. Assuming that your applications provide HTTP APIs hidden behind ELBs, you can use ZMON to gather this data from CloudWatch.

For more detailed data, ZMON offers options for different languages and frameworks. One is zmon-actuator for Spring Boot. ZMON gathers the data by querying a JSON endpoint `/metrics` adhering to the DropWizard metrics layout with some convention on the naming of timers. Basically on timer per API path and status code.

We also recommend checking out Friboo for working with Clojure, the Python/Flask framework Connexion or Markscheider for Play/Scala development.

The http(url=...).actuator_metrics() will parse the data into a Python dict that allows you to easily monitor and alert on changes in API behavior.

This also drives ZMON's cloud UI.



CHAPTER 13

Requirements

The requirements below are all open soure technologies that need to be available for ZMON to run with all its features.

13.1 Redis

The Redis service is one of the core dependencies, ZMON uses Redis for its task queue and to store its current state.

13.2 PostgreSQL

PostgreSQL is ZMONs data store for entities, checks, alerts, dashboards and Grafana dashboards. The entities service relies on PostgreSQL's jsonb data type thus you need a PostgreSQL 9.4+ running.

13.3 Cassandra

Cassandra needs to be available for KairosDB if you want to have historic data and make use of Grafana, this is highly suggested. We strongly recommend to run Cassandra 3.7+ and using TimeWindow compaction strategy for KairosDB. This will nicely split your SSTables into a single file per day (depending on your config).

13.4 KairosDB

KairosDB is our time series database of choice, however by now we are running our own fork. This is not required for standard volume scenarios we believe. ZMON will store every metric gathered in KairosDB so that you can use it directly or via Graphana to access historic data. ZMON itself allows you to plot charts from KairosDB in Dashboard widgets or go to check/alert specific charts directly.

CHAPTER 14

Essential ZMON Components

To use ZMON requires these four components: zmon-controller, zmon-scheduler, zmon-worker, and zmon-eventlog-service.

14.1 Controller

zmon-controller runs ZMON's AngularJS frontend and serves as an endpoint for retrieving data and managing your ZMON deployment via REST API (with help from the command line client). It needs a connection configured to:

- PostgreSQL to store/retrieve all kind of data: entities, checks, dashboards, alerts
- Redis, to keep the state of ZMON's alerts
- KairosDB, if you want charts/Grafana

To provide a means of authentication and authorization, you can choose between the following options:

- A basic credential file
- An OAuth2 identity provider, e.g., GitHub

14.2 Scheduler

zmon-scheduler is responsible for keeping track of all existing entities, checks and alerts and scheduling checks in time for applicable entities, which are then executed by the worker.

Needs connections to:

- Redis, which serves ZMON as a task queue
- Controller, to get check/alerts/entities
- Custom adapters might need connections for entity discovery in your platform

14.3 Worker

zmon-worker does the heavy lifting — executing tasks against entities and evaluating all alerts assigned to this check. Tasks are picked up from Redis and the resulting check value plus alert state changes are written back to Redis.

Needs connection to:

- Redis to retrieve tasks and update current state
- KairosDB if you want to have metrics
- EventLog service to store history events for alert state changes

14.4 EventLog Service

zmon-eventlog-service is our slim implementation of an event store, keeping track of Events related to alert state changes as well as events like alert and check modification by the user.

Needs connection to:

- PostgreSQL to store events using jsonb

CHAPTER 15

Component Configuration

In this section we assume that you want to use Docker as means of deployment. The ZMON Dockerimages in Zalando's Open Source registry are exactly the ones we use ourselves, injecting all configuartion via environment variables.

If this does not fit your needs you can run the artifacts directly and decide to use environment variables or modify the example config files.

At this point we also assume the requirements in terms of PostgreSQL, Redis and KairosDB are available and you have the credentials at hand. If not see *Requirements*. The minimal configuration options below are taken from the Demo's Bootstrap script!

15.1 Authentication

For the ZMON controller we assume that it is publicly accessible. Thus the UI always requires users to login and the REST API, too. The REST API relies on tokens via the `Authorization: Bearer <token>` header to allow access. For environments where you have no OAauth2 setup you can configure pre-shared keys for API access.

Note: Feel free to look at Zalando's Plan-B, which is a freely available OAuth2 provider we use for our platform to secure service to service communication.

Creating a preshared token can be achieved like this and adding them to the Controller configuration.

```
SCHEDULER_TOKEN=$(makepasswd --string=0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ --chars 32)
```

Warning: Due to magic in matching env vars token must be ALL UPPERCASE

Scheduler and worker both at times call the controller's REST API thus you need to configure tokens for them. For the scheduler, KairosDB, eventlog-service and metric-cache if deployed we assume for now they are private. Theses

services are accessed only by worker and controller and do not need to be public. Same is true for Redis, PostgreSQL and Cassandra. However in general we advise you to setup proper credentials and roles where possible.

15.2 Running Docker

First we need to figure out what tags to run. Belows bash snippet helps you to retrieve and set the latest available tags.

```
function get_latest () {
    name=$1
    # REST API returns tags sorted by time
    tag=$(curl --silent https://registry.opensource.zalan.do/teams/stups/artifacts/
↪$name/tags | jq .[].name -r | tail -n 1)
    echo "$name:$tag"
}

echo "Retrieving latest versions.."
REPO=registry.opensource.zalan.do/stups
POSTGRES_IMAGE=$REPO/postgres:9.4.5-1
REDIS_IMAGE=$REPO/redis:3.2.0-alpine
CASSANDRA_IMAGE=$REPO/cassandra:2.1.5-1
ZMON_KAIROSDB_IMAGE=$REPO/$(get_latest kairosdb)
ZMON_EVENTLOG_SERVICE_IMAGE=$REPO/$(get_latest zmon-eventlog-service)
ZMON_CONTROLLER_IMAGE=$REPO/$(get_latest zmon-controller)
ZMON_SCHEDULER_IMAGE=$REPO/$(get_latest zmon-scheduler)
ZMON_WORKER_IMAGE=$REPO/$(get_latest zmon-worker)
ZMON_METRIC_CACHE=$REPO/$(get_latest zmon-metric-cache)
```

To run the selected images use Docker's run command together with the options explained below. We use the following wrapper for this:

```
function run_docker () {
    name=$1
    shift 1
    echo "Starting Docker container ${name}.."
    # ignore non-existing containers
    docker kill $name &> /dev/null || true
    docker rm -f $name &> /dev/null || true
    docker run --restart "on-failure:10" --net zmon-demo -d --name $name $@
}

run_docker zmon-controller \
            # -e ......... \
            # -e ......... \
           $ZMON_CONTROLLER_IMAGE
```

15.3 Controller

15.3.1 Authentication

Configure your Github application

```
-e SPRING_PROFILES_ACTIVE=github \
-e ZMON_OAUTH2_SSO_CLIENT_ID=64210244ddd8378699d6 \
-e ZMON_OAUTH2_SSO_CLIENT_SECRET=48794a58705d1ba66ec9b0f06a3a44ecb273c048 \
```

Make everyone admin for now:

```
-e ZMON_AUTHORITIES_SIMPLE_ADMINS=* \
```

15.3.2 Dependencies

Configure PostgreSQL access:

```
-e POSTGRES_URL=jdbc:postgresql://$PGHOST:5432/local_zmon_db \
-e POSTGRES_PASSWORD=$PGPASSWORD \
```

Setup Redis connection:

```
-e REDIS_HOST=zmon-redis \
-e REDIS_PORT=6379 \
```

Set CORS allowed origins:

```
-e ENDPOINTS_CORS_ALLOWED_ORIGINS=https://demo.zmon.io \
```

Setup URLs for other services:

```
-e ZMON_EVENTLOG_URL=http://zmon-eventlog-service:8081/ \
-e ZMON_KAIROSDB_URL=http://zmon-kairosdb:8083/ \
-e ZMON_METRICCACHE_URL=http://zmon-metric-cache:8086/ \
-e ZMON_SCHEDULER_URL=http://zmon-scheduler:8085/ \
```

And last but not least, configure a preshared token, to allow the scheduler and worker to access the REST API. Remember tokens need to all uppercase here.

```
-e PRESHARED_TOKENS_${SCHEDULER_TOKEN}_UID=zmon-scheduler \
-e PRESHARED_TOKENS_${SCHEDULER_TOKEN}_EXPIRES_AT=1758021422 \
-e PRESHARED_TOKENS_${SCHEDULER_TOKEN}_AUTHORITY=user
```

15.3.3 Firebase and Webpush

Enable desktop push notification UI with the following options:

```
-e ZMON_ENABLE_FIREBASE=true \
-e ZMON_NOTIFICATIONSERVICE_URL=http://zmon-notification-service:8087/ \
-e ZMON_FIREBASE_API_KEY="AIzaSyBM1ktKS5u_d2jxWPHVU7Xk39s-PG5gy7c" \
-e ZMON_FIREBASE_AUTH_DOMAIN="zmon-demo.firebaseapp.com" \
-e ZMON_FIREBASE_DATABASE_URL="https://zmon-demo.firebaseio.com" \
-e ZMON_FIREBASE_STORAGE_BUCKET="zmon-demo.appspot.com" \
-e ZMON_FIREBASE_MESSAGING_SENDER_ID="280881042812" \
```

This feature requires additional config for the worker and to run the notification-service.

15.4 Scheduler

Specify the Redis server you want to use:

```
-e SCHEDULER_REDIS_HOST=zmon-redis \
-e SCHEDULER_REDIS_PORT=6379 \
```

Setup access to the controller and entity service (both provided by the controller): Not the reuse of the above defined pre shared key!

```
-e SCHEDULER_OAUTH2_STATIC_TOKEN=$SCHEDULER_TOKEN \
-e SCHEDULER_URLS_WITHOUT_REST=true \
-e SCHEDULER_ENTITY_SERVICE_URL=http://zmon-controller:8080/ \
-e SCHEDULER_CONTROLLER_URL=http://zmon-controller:8080/ \
```

If you run into scenarios of different queues or the demand for different levels of parallelism, e.g. limiting number of queries run at MySQL/PostgreSQL databases use the following as an example:

```
-e SPRING_APPLICATION_JSON='{"scheduler":{"queue_property_mapping":{"zmon:queue:mysql
↪":[{"type":"mysql"}]}}}'
```

This will route checks agains entities of type "mysql" to another queue.

15.5 Worker

The worker configuration is split into essential configuration options, like Redis and KairosDB and the plugin configuration, e.g. PostgreSQL credentials, ...

15.5.1 Essential Options

Configure Redis Access:

```
-e WORKER_REDIS_SERVERS=zmon-redis:6379 \
```

Configure parallelism and throughput:

```
-e WORKER_ZMON_QUEUES=zmon:queue:default/25,zmon:queue:mysql/3
```

Specify the number of worker processes that are polling the queues and execute tasks. You can specify multiple queues here to listen to.

Configure KairosDB:

```
-e WORKER_KAIROSDB_HOST=zmon-kairosdb \
```

Configure EventLog service:

```
-e WORKER_EVENTLOG_HOST=zmon-eventlog-service \
-e WORKER_EVENTLOG_PORT=8081 \
```

Configure Worker token to access controller API: (relying on Python tokens library here)

```
-e  OAUTH2_ACCESS_TOKENS=uid=$WORKER_TOKEN \
```

Configure Worker named tokens to access external APIs:

```
-e WORKER_PLUGIN_HTTP_OAUTH2_TOKENS=token_name1=scope1,scope2,scope3:token_
↪name2=scope1,scope2
```

Configure Metric Cache (optional):

```
-e WORKER_METRICCACHE_URL=http://zmon-metric-cache:8086/api/v1/rest-api-metrics/ \
-e WORKER_METRICCACHE_CHECK_ID=9 \
```

15.5.2 Notification Options

Firebase and Webpush

To trigger notifications for desktop web and mobile apps set the following params to point to notification service.

WORKER_NOTIFICATION_SERVICE_URL Notification service base url

WORKER_NOTIFICATION_SERVICE_KEY (optional, if not using oauth2) A shared key configured in the notification service

Hipchat

WORKER_NOTIFICATIONS_HIPCHAT_TOKEN Access token for HipChat notifications.

WORKER_NOTIFICATIONS_HIPCHAT_URL URL of HipChat server.

HTTP

This allows to trigger HTTP Post calls to arbitrary services.

WORKER_NOTIFICATIONS_HTTP_DEFAULT_URL HTTP endpoint default URL.

WORKER_NOTIFICATIONS_HTTP_WHITELIST_URLS List of whitelist URL endpoints. If URL is not in this list, then exception will be raised.

WORKER_NOTIFICATIONS_HTTP_ALLOW_ALL Allow any URL to be used in HTTP notification.

WORKER_NOTIFICATIONS_HTTP_HEADERS Default headers to be used in HTTP requests.

Mail

WORKER_NOTIFICATIONS_MAIL_HOST SMTP host for email notifications.

WORKER_NOTIFICATIONS_MAIL_PORT SMTP port for email notifications.

WORKER_NOTIFICATIONS_MAIL_SENDER Sender address for email notifications.

WORKER_NOTIFICATIONS_MAIL_USER SMTP user for email notifications.

WORKER_NOTIFICATIONS_MAIL_PASSWORD SMTP password for email notifications.

Slack

WORKER_NOTIFICATIONS_SLACK_WEBHOOK Slack webhook for channel notifications.

Twilio

WORKER_NOTIFICATION_SERVICE_URL URL of notification service (needs to be publicly accessible)

WORKER_NOTIFICATION_SERVICE_KEY (optional, if not using oauth2) Preshared key to call notification service

Pagerduty

WORKER_NOTIFICATIONS_PAGERDUTY_SERVICEKEY Routing key for a Pagerduty service

Plug-In Options

All plug-in options have the prefix `WORKER_PLUGIN_<plugin-name>_`, i.e. if you want to set option "bar" of the plugin "foo" to "123" via environment variable:

```
WORKER_PLUGIN_FOO_BAR=123
```

If you plan to access your PostgreSQL cluster specify the credentials below. We suggest to use a distinct user for ZMON with limited read only privileges.

```
WORKER_PLUGIN_SQL_USER
WORKER_PLUGIN_SQL_PASS
```

If you need to access MySQL specify the user credentials below, again we suggest to use a user with limited privileges only.

```
WORKER_PLUGIN_MYSQL_USER
WORKER_PLUGIN_MYSQL_PASS
```

15.6 Notification Service

Optional component to service mobile API, push notifications and Twilio notifications.

15.6.1 Authentication

SPRING_APPLICATION_JSON Use this to define pre-shared keys if not using OAuth2. Specify key and max validity.

```
{"notifications":{"shared_keys":{"<your random key>": 1504981053654}}}
```

15.6.2 Firebase and Web Push

NOTIFICATIONS_GOOGLE_PUSH_SERVICE_API_KEY Private Firebase messaging server key

NOTIFICATIONS_ZMON_URL ZMON's base URL

15.6.3 Twilio options

NOTIFICATIONS_TWILIO_API_KEY Private API Key

NOTIFICATIONS_TWILIO_USER User

NOTIFICATIONS_TWILIO_PHONE_NUMBER Phone number to use

NOTIFICATIONS_DOMAIN Domain under which notification service is reachable

CHAPTER 16

Rest API

16.1 Authentication & Authorization

You need to obtain a token to access ZMON's REST API. For the default deployment using Github rely on access tokens from Github, otherwise it depends on your selected provider.

Your application should always examine the HTTP status of the response. Any value other than 200 indicates a failure.

Here are some examples:

Request with invalid credentials:

```
HTTP/1.1 401 Unauthorized
Content-Type: application/json;charset=UTF-8
Content-Length: 29
Date: Thu, 21 Aug 2014 10:28:10 GMT

{"message":"Bad credentials"}
```

Request without proper authentication:

```
HTTP/1.1 401 Unauthorized
Content-Type: application/json;charset=UTF-8
Content-Length: 69
Date: Thu, 21 Aug 2014 10:29:14 GMT

{"message":"Full authentication is required to access this resource"}
```

Request without proper authorization:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json;charset=UTF-8
Content-Length: 30
Date: Thu, 21 Aug 2014 10:31:20 GMT

{"message":"Access is denied"}
```

16.2 Entities

see *CLI entities*

16.3 Check Definitions

see *CLI check definitions*

16.4 Dashboards

16.5 Downtimes

For more info about this feature, please check this

16.5.1 Scheduling a downtime

Resource URL: POST /api/v1/downtimes

Description

Create a new downtime, returning the id of the newly created resource. If none of the alert definition entities match this request it will succeed and return an empty list of entities/alert definitions. Any attempt to execute this method without proper authentication will result in a 401. If the user does not have enough permissions (role: api-writer) this method will return an HTTP 403. In case of malformed syntax or missing mandatory fields this method will return an HTTP 400 and the client SHOULD NOT repeat the request without modifications. In case of success this method will return HTTP 200.

> **Note:** Alerts and checks with hard-coded entity identifiers in the check command are not covered.

Parameters:

Name	Data Type	Mandatory	Description
comment	String	yes	Downtime comment
start_time	Number	no	The start time in seconds since epoch. **Default:** current time
end_time	Number	yes	The end time in seconds since epoch. **Precondition:** end_time > start_time
entities	Array	yes	Array of entities to set in downtime. (e.g. htt01:4420) **Precondition:** The array should have at least one element
alert_definitions	Array	no	Alert definition ids. If specified, only entities belonging to these alert definitions will be set in downtime.

Example:

```
curl -v --user hjacobs:test 'https://zmon.example.com/api/v1/downtimes' \
   -H 'Content-Type: application/json' \
   --data-binary $'{"comment":"Cities downtime","end_time":1408665600,"entities":["cd-
↪kinshasa", "cn-peking"]}'
```

Request:

```
POST /api/v1/downtimes HTTP/1.1
Authorization: Basic aGphY29iczp0ZXN0
User-Agent: curl/7.30.0
Host: zmon.example.com
Accept: */*
Content-Type: application/json
Content-Length: 91

{"comment":"Cities downtime","end_time":1408665600,"entities":["cd-kinshasa", "cn-
↪peking"]}
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Thu, 21 Aug 2014 14:26:02 GMT

{"comment":"Cities downtime","start_time":1408631162,"end_time":1408665600,"created_by
↪":"hjacobs",
"id":"cf6ada50-3eb2-4c17-8d09-4eb03dc19cf5","entities":["cn-peking","cd-kinshasa"],
↪"alert_definitions":[704]}
```

16.5.2 Deleting a downtime

Resource URL: DELETE /api/v1/downtimes/{id}

Description

Attempt to delete the downtime with the specified id. If the downtime ID doesn't exist, the request will succeed and return an empty list of entities/alert definitions. Any attempt to execute this method without proper authentication will result in a 401. If the user doesn't have enough permissions (role: api-writer) this method will return an HTTP 403. In case of malformed syntax or missing mandatory fields this method will return an HTTP 400 and the client SHOULD NOT repeat the request without modifications. In case of success this method will return HTTP 200.

Parameters:

Name	Data Type	Mandatory	Description
id	String	yes	Id of the downtime to delete

Example:

```
curl -v --user hjacobs:test 'https://zmon.example.com/api/v1/downtimes/cf6ada50-3eb2-
↪4c17-8d09-4eb03dc19cf5' \
  -H 'Content-Type: application/json' \
  -X DELETE
```

Request:

```
DELETE /api/v1/downtimes/cf6ada50-3eb2-4c17-8d09-4eb03dc19cf5 HTTP/1.1
Authorization: Basic aGphY29iczp0ZXN0
User-Agent: curl/7.30.0
Host: zmon.example.com
Accept: */*
Content-Type: application/json
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Thu, 21 Aug 2014 15:16:51 GMT

{"comment":"Cities downtime","start_time":1408633908,"end_time":1408665600,"created_by
↪":"hjacobs",
"id":"0ff6ed67-9521-42a7-8132-5ab837193af9","entities":["cn-peking","cd-kinshasa"],
↪"alert_definitions":[704]}
```

16.6 Alert Definitions

For more info about this feature, please check *this*

16.6.1 Creating a new Alert Definition

Resource URL: POST /api/v1/alert-definitions

Description

Create a new alert definition, returning the id of the newly created resource. Alert definitions can be created based on another alert definition whereby a child reuses attributes from the parent. Each alert definition can only inherit from a single alert definition (single inheritance).

One can also create templates. A Template is basically an alert definition with a subset of mandatory attributes that is not evaluated and is only used for extension.

Any attempt to execute this method without proper authentication will result in a 401. In case of success this method will return HTTP 200.

Parameters:

Name	Data Type	Mandatory	Inherited	Description
name	String	yes	yes	The alert's display name on the dashboard. This field can contain curly-brace variables like {mycapture} that are replaced by capture's value when the alert is triggered. It's also possible to format decimal precision (e.g. "My alert {mycapture:.2f}" would show as "My alert 123.45" if mycapture is 123.456789). To include a comma separated list of entities as part of the alert's name, just use the special placeholder {entities}. This field can be omitted if the new definition extends an existing one with this field defined (templates might not have all fields).
description	String	yes	yes	Meaningful text for people trying to handle the alert. This field can be omitted if the new definition extends an existing one with this field defined.
team	String	yes	no	Team dashboard to show the alert on.
responsible_team	String	yes	no	Additional team field that allows one to delegate alert monitoring to other teams. The responsible team's name will be shown on the dashboard. This team is responsible for fixing the problem in case the alert is triggered.
entities	Array	yes	yes	Filter used to select a subset of check definition entities. If empty, the condi-

Example:

```
curl --user hjacobs:test 'https://zmon.example.com/api/v1/alert-definitions' -H
↪'Content-Type: application/json' \
    --data-binary $'{"name": "City Longitude >0", "description": "Test whether a city␣
↪lies east or west", "team": "Platform/Software", "responsible_team": "Platform/
↪Software", "entities": [{"type": "city"}], "entities_exclude": [], "condition":
↪"capture(longitude=float(value)) > longitude_param", "notifications": [], "check_
↪definition_id": 20, "status": "ACTIVE", "priority": 2, "period": "", "template":␣
↪false, "parameters": {"longitude_param": {"comment": "Longitude parameter","type":
↪"float", "value": 0}}, "tags": ["CITY"]}'
```

Request:

```
POST /api/v1/alert-definitions HTTP/1.1
Authorization: Basic aGphY29iczp0ZXN0
User-Agent: curl/7.30.0
Host: zmon.example.com
Accept: */*
Content-Type: application/json
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Tue, 26 Aug 2014 18:02:29 GMT

{"id":788,"name":"City Longitude >0","description":"Test whether a city lies east or␣
↪west",
"team":"Platform/Software","responsible_team":"Platform/Software","entities":[{"type":
↪"city"}],
"entities_exclude":[],"condition":"capture(longitude=float(value)) > longitude_param",
↪"notifications":[],
"check_definition_id":20,"status":"ACTIVE","priority":2,"last_modified":1409076149956,
↪"last_modified_by":"hjacobs",
"period":"","template":false,"parent_id":null,
"parameters":{"longitude_param":{"value":0,"comment":"Longitude parameter","type":
↪"float"}},"tags":["CITY"]}
```

16.6.2 Updating an Alert Definition

Resource URL: PUT /api/v1/alert-definitions/{id}

Description

Updates an existing alert definition. If the alert definintion doesn't exist, this method will return a 404.

For more info about the parameters, please check *how to create a new Alert Definition*

Example:

```
curl --user hjacobs:test 'https://zmon.example.com/api/v1/alert-definitions/788' \
-H 'Content-Type: application/json' \
--data-binary $'{"name": "City Longitude >0", "description": "Checks whether a city␣
↪lies east or west", "team": "Platform/Software", "responsible_team": "Platform/
↪Software", "entities": [{"type": "city"}], "entities_exclude": [], "condition":
↪"capture(longitude=float(value)) > longitude_param", "notifications": [], "check_
↪definition_id": 20, "status": "ACTIVE", "priority": 2, "period": "", "template":␣
↪false, "parameters": {"longitude_param": {"comment": "Longitude parameter","type":
↪"float", "value": 0}}, "tags": ["CITY"]}' \
```

```
-X PUT
```

Request:

```
PUT /api/v1/alert-definitions/788 HTTP/1.1
Authorization: Basic aGphY29iczp0ZXN0
User-Agent: curl/7.30.0
Host: zmon.example.com
Accept: */*
Content-Type: application/json
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Tue, 26 Aug 2014 18:47:00 GMT

{"id":788,"name":"City Longitude >0","description":"Checks whether a city lies east␣
↪or west",
"team":"Platform/Software","responsible_team":"Platform/Software","entities":[{"type":
↪"city"}],
"entities_exclude":[],"condition":"capture(longitude=float(value)) > longitude_param",
↪"notifications":[],
"check_definition_id":20,"status":"ACTIVE","priority":2,"last_modified":1409078820694,
↪"last_modified_by":"hjacobs",
"period":"","template":false,"parent_id":null,
"parameters":{"longitude_param":{"value":0,"comment":"Longitude parameter","type":
↪"float"}},"tags":["CITY"]}
```

16.6.3 Find an Alert Defintion by ID

Resource URL: GET /api/v1/alert-definitions/{id}

Description

Find an existing alert definition by id. If the alert definintion doesn't exist, this method will return a 404.

Example:

```
curl -v --user hjacobs:test 'https://zmon.example.com/api/v1/alert-definitions/788' \
-H 'Content-Type: application/json'
```

Request:

```
GET /api/v1/alert-definitions/788 HTTP/1.1
Authorization: Basic aGphY29iczp0ZXN0
User-Agent: curl/7.30.0
Host: zmon.example.com
Accept: */*
Content-Type: application/json
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
```

```
Date: Tue, 26 Aug 2014 18:47:00 GMT

{"id":788,"name":"City Longitude >0","description":"Checks whether a city lies east␣
↪or west",
"team":"Platform/Software","responsible_team":"Platform/Software","entities":[{"type":
↪"city"}],
"entities_exclude":[],"condition":"capture(longitude=float(value)) > longitude_param",
↪"notifications":[],
"check_definition_id":20,"status":"ACTIVE","priority":2,"last_modified":1409078820694,
↪"last_modified_by":"hjacobs",
"period":"","template":false,"parent_id":null,
"parameters":{"longitude_param":{"value":0,"comment":"Longitude parameter","type":
↪"float"}},"tags":["CITY"]}
```

16.6.4 Retrieving Alert Status

Resource URL: GET /api/v1/status/alert/{alert ids}/

Description

Returns current status of the given alert IDs. The information comes directly from Redis and represents results of the last alert evaluation

The results are returned in the following format (so basically for each alert and entity you get information

- when alert started (**ts**)
- how long has evaluation taken (**td**)
- are there any downtimes (**downtimes**)
- capture values, if available (**captures**)
- which worker has processed the value (**worker**)
- the latest check value (**value**)

NOTE Please keep in mind that this request will only work if you specify trailing slash (as in the example below).

```
{"alert id":
    {
            "entity name":
            {
                    "td":0.013866,
                    "downtimes":[],
                    "captures":{"count":1},
                    "start_time":1.416391418749185E9,
                    "worker":"p3426.itr-monitor01",
                    "ts":1.4164876292204E9,
                    "value":1
            }
    }
}
```

Any attempt to execute this method without proper authentication will result in a 401. In case of success this method will return HTTP 200.

Example:

```
curl --user hjacobs:test 'https://zmon.example.com/api/v1/status/alert/69,3454/'
```

Request:

```
GET https://zmon.example.com/api/v1/status/alert/69,3454/ HTTP/1.1
Authorization: Basic aGphY29iczp0ZXN0
User-Agent: curl/7.30.0
Host: zmon.example.com
Accept: */*
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Vary: Accept-Encoding
Date: Thu, 20 Nov 2014 12:47:37 GMT

{"69":{"itr-elsn02:5827":{"td":0.013866,"downtimes":[],"captures":{"count":1},"start_
↪time":1.416391418749185E9,"worker":"p3426.itr-monitor01","ts":1.4164876292204E9,
↪"value":1},"elsn03:5827":{"td":0.015576,"downtimes":[],"captures":{"count":8},
↪"start_time":1.416391397741839E9,"worker":"p3426.monitor02","ts":1.
↪416487629218565E9,"value":8},"elsn02:5827":{"td":0.024973,"downtimes":[],"captures":
↪{"count":9},"start_time":1.416330457394862E9,"worker":"p3426.itr-monitor01","ts":1.
↪416487629223615E9,"value":9},"itr-elsn03:5827":{"td":0.020491,"downtimes":[],
↪"captures":{"count":1},"start_time":1.416255229204794E9,"worker":"p3426.itr-
↪monitor01","ts":1.41648762923005E9,"value":1},"elsn01:5827":{"td":0.019912,
↪"downtimes":[],"captures":{"count":8},"start_time":1.416391418966269E9,"worker":
↪"p3426.monitor03","ts":1.416487629216758E9,"value":8},"itr-elsn01:5827":{"td":0.
↪015741,"downtimes":[],"captures":{"count":2},"start_time":1.416391429438217E9,
↪"worker":"p3426.itr-monitor01","ts":1.416487629224237E9,"value":2}},"3454":{
↪"monitor02":{"td":0.027714,"downtimes":[],"captures":{},"start_time":1.
↪414754929626809E9,"worker":"p3426.monitor02","ts":1.416487578812573E9,"value":{
↪"load1":8.71,"load15":9.73,"load5":10.22}},"monitor03":{"td":0.028951,"downtimes
↪":[],"captures":{},"start_time":1.41475492971822E9,"worker":"p3426.monitor02","ts
↪":1.41648757881069E9,"value":{"load1":9.25,"load15":11.17,"load5":10.9}}}}}
```

CHAPTER 17

Command Line Client

The command line client makes your life easier when interacting with the REST API. The ZMON scheduler will refresh modified data (checks, alerts, entities every 60 seconds).

17.1 Installation

```
pip3 install --upgrade zmon-cli
```

17.1.1 Configuration

Configure your zmon cli by running `configure-`

```
zmon configure
```

17.1.2 Authentication

ZMON CLI tool must authenticate against ZMON. Internally it uses zign to obtain access token, but you can override that behaviour by exporting a variable ZMON_TOKEN.

```
export ZMON_TOKEN=myfancytoken
```

If you are using github for authentication, have an unprivileged personal access token ready.

17.2 Entities

17.2.1 Create or update

Pushing entities with the zmon cli is as easy as:

```
zmon entities push \
  '{"id":"localhost:3421","name":"zmon-scheduler-ng","host":"localhost","ports":{"3421
↪":3421}}'
```

Existing entities with the same ID will be updated.

The client however also supports loading data from .json and .yaml files, both may contain a list for creating/updating many entities at once.

```
zmon entities push your-entities.yaml
```

Tip: All commands and subcommands can be abbreviated, i.e. the following lines are equivalent:

```
$ zmon entities push my-data.yaml
$ zmon ent pu my-data.yaml
```

17.2.2 Search and filter

Show all entities:

```
zmon entities
```

Filter by type "instance"

```
zmon entities filter type instance
```

17.3 Check Definitions

17.3.1 Initializing

When starting from scratch use:

```
zmon check-definition init your-new-check.yaml
```

17.3.2 Get

Retrieve an existing check defintion as YAML.

```
zmon check-definition get 1234
```

17.3.3 Create and Update

Create or update from file, existing check with same "owning_team" and "name" will be updated.

```
zmon check-definition update your-check.yaml
```

17.4 Alert Definitions

Similar to check defintions you can also manage your alert definitions via the ZMON cli.

Keep in mind that for alerts the same constraints apply as in the UI. For creating/modifying an alert you need to be a member of the team selected for "team" (unlike the responsible team).

17.4.1 Init

```
zmon alert-definition init your-new-alert.yaml
```

17.4.2 Create

```
zmon alert-definition create your-new-alert.yaml
```

17.4.3 Get

```
zmon alert-definition get 1999
```

17.4.4 Update

```
zmon alert-definition update host-load-5.yaml
```

CHAPTER 18

Python Client

ZMON provides a python client library that can be imported and used in your own software.

18.1 Installation

ZMON python client library is part of *ZMON CLI*.

```
pip3 install --upgrade zmon-cli
```

18.2 Usage

Using ZMON client is pretty straight forward.

```
>>> from zmon_cli.client import Zmon

>>> zmon = Zmon('https://zmon.example.org', token='123')

>>> entity = zmon.get_entity('entity-1')
{
    'id': 'entity-1',
    'team': 'ZMON',
    'type': 'instance',
    'data': {'host': '192.168.20.16', 'port': 8080, 'name': 'entity-1-instance'}
}

>>> zmon.delete_entity('entity-102')
True

>>> check = zmon.get_check_definition(123)

>>> check['command']
```

```
http('http://www.custom-service.example.org/health').code()

>>> check['command'] = "http('http://localhost:9090/health').code()"

>>> zmon.update_check_definition(check)
{
    'command': "http('http://localhost:9090/health').code()",
    'description': 'Check service health',
    'entities': [{'application_id': 'custom-service', 'type': 'instance'}],
    'id': 123,
    'interval': 60,
    'last_modified_by': 'admin',
    'name': 'Check service health',
    'owning_team': 'ZMON',
    'potential_analysis': None,
    'potential_impact': None,
    'potential_solution': None,
    'source_url': None,
    'status': 'ACTIVE',
    'technical_details': None
}
```

18.3 Client

18.3.1 Exceptions

class zmon_cli.client.**ZmonError**(*message=''*)
ZMON client error.

class zmon_cli.client.**ZmonArgumentError**(*message=''*)
A ZMON client error indicating that a supplied object has missing or invalid attributes.

18.3.2 Zmon

class zmon_cli.client.**Zmon**(*url*, *token=None*, *username=None*, *password=None*, *timeout=10*, *verify=True*, *user_agent='zmon-client/1.1.56'*)
ZMON client class that enables communication with ZMON backend.

Parameters

- **url** (str) – ZMON backend base url.
- **token** (str) – ZMON authentication token.
- **username** (str) – ZMON authentication username. Ignored if `token` is used.
- **password** (str) – ZMON authentication password. Ignored if `token` is used.
- **timeout** (int) – HTTP requests timeout. Default is 10 sec.
- **verify** (bool) – Verify SSL connection. Default is `True`.
- **user_agent** (str) – ZMON user agent. Default is generated by ZMON client and includes lib version.

add_entity(*entity: dict*) → requests.models.Response
Create or update an entity on ZMON.

Note: ZMON PUT entity API doesn't return JSON response.

Parameters **entity** (dict) – Entity dict.

Returns Response object.

Return type requests.Response

alert_details_url(*alert: dict*) → str
Return direct deeplink to alert details view on ZMON UI.

Parameters **alert** (dict) – alert dict.

Returns Deeplink to alert details view.

Return type *str*

check_definition_url(*check_definition: dict*) → str
Return direct deeplink to check definition view on ZMON UI.

Parameters **check_definition** (dict) – check_difinition dict.

Returns Deeplink to check definition view.

Return type *str*

create_alert_definition(*alert_definition: dict*) → dict
Create new alert definition.

Attributes last_modified_by and check_definition_id are required. If status is not set, then it will be set to ACTIVE.

Parameters **alert_definition** (dict) – ZMON alert definition dict.

Returns Alert definition dict.

Return type *dict*

create_downtime(*downtime: dict*) → dict
Create a downtime for specific entities.

Atrributes entities list, start_time and end_time timestamps are required.

Parameters **downtime** (dict) – Downtime dict.

Returns Downtime dict.

Return type *dict*

Example downtime:

```
{
    "entities": ["entity-id-1", "entity-id-2"],
    "comment": "Planned maintenance",
    "start_time": 1473337437.312921,
    "end_time": 1473341037.312921,
}
```

dashboard_url(*dashboard_id: int*) → str
Return direct deeplink to ZMON dashboard.

Parameters **dashboard_id** (int) – ZMON Dashboard ID.

Returns Deeplink to dashboard.

Return type *str*

delete_alert_definition (*alert_definition_id: int*) → dict
Delete existing alert definition.

Parameters **alert_definition_id** (int) – ZMON alert definition ID.

Returns Alert definition dict.

Return type *dict*

delete_check_definition (*check_definition_id: int*) → requests.models.Response
Delete existing check definition.

Parameters **check_definition_id** (int) – ZMON check definition ID.

Returns HTTP response.

Return type requests.Response

delete_entity (*entity_id: str*) → bool
Delete entity from ZMON.

Note: ZMON DELETE entity API doesn't return JSON response.

Parameters **entity_id** (str) – Entity ID.

Returns True if succeeded, False otherwise.

Return type *bool*

get_alert_data (*alert_id: int*) → dict
Retrieve alert data.

Response is a dict with entity ID as a key, and check return value as a value.

Parameters **alert_id** (int) – ZMON alert ID.

Returns Alert data dict.

Return type *dict*

Example:

```
{
    "entity-id-1": 122,
    "entity-id-2": 0,
    "entity-id-3": 100
}
```

get_alert_definition (*alert_id: int*) → dict
Retrieve alert definition.

Parameters **alert_id** (int) – Alert definition ID.

Returns Alert definition dict.

Return type *dict*

get_alert_definitions () → list
Return list of all active alert definitions.

Returns List of alert-defs.

Return type *list*

get_check_definition(*definition_id: int*) → dict
Retrieve check defintion.

Parameters **defintion_id** (int) – Check defintion id.

Returns Check definition dict.

Return type *dict*

get_check_definitions() → list
Return list of all active check definitions.

Returns List of check-defs.

Return type *list*

get_dashboard(*dashboard_id: str*) → dict
Retrieve a ZMON dashboard.

Parameters **dashboard_id** (int, str) – ZMON dashboard ID.

Returns Dashboard dict.

Return type *dict*

get_entities(*query=None*) → list
Get ZMON entities, with optional filtering.

Parameters **query** (dict) – Entity filtering query. Default is None. Example query {'type': 'instance'} to return all entities of type: instance.

Returns List of entities.

Return type *list*

get_entity(*entity_id: str*) → str
Retrieve single entity.

Parameters **entity_id** (str) – Entity ID.

Returns Entity dict.

Return type *dict*

get_grafana_dashboard(*grafana_dashboard_id: str*) → dict
Retrieve Grafana dashboard.

Parameters **grafana_dashboard_id** (str) – Grafana dashboard ID.

Returns Grafana dashboard dict.

Return type *dict*

get_onetime_token() → str
Retrieve new one-time token.

You can use *zmon_cli.client.Zmon.token_login_url()* to return a deeplink to one-time login.

Returns One-time token.

Retype str

grafana_dashboard_url(*dashboard: dict*) → str
Return direct deeplink to Grafana dashboard.

Parameters **dashboard** (dict) – Grafana dashboard dict.

Returns Deeplink to Grafana dashboard.

Return type *str*

list_onetime_tokens() → list
List exisitng one-time tokens.

Returns List of one-time tokens, with relevant attributes.

Retype list

Example:

```
- bound_at: 2016-09-08 14:00:12.645999
  bound_expires: 1503744673506
  bound_ip: 192.168.20.16
  created: 2016-08-26 12:51:13.506000
  token: 9pSzKpcO
```

search(*q*, *limit: int = None*, *teams: list = None*) → dict
Search ZMON dashboards, checks, alerts and grafana dashboards with optional team filtering.

Parameters

- **q** (str) – search query.
- **teams** (list) – List of team IDs. Default is None.

Returns Search result.

Return type *dict*

Example:

```
{
    "alerts": [{"id": "123", "title": "ZMON alert", "team": "ZMON"}],
    "checks": [{"id": "123", "title": "ZMON check", "team": "ZMON"}],
    "dashboards": [{"id": "123", "title": "ZMON dashboard", "team": "ZMON"}],
    "grafana_dashboards": [{"id": "123", "title": "ZMON grafana", "team": ""}
↪],
}
```

status() → dict
Return ZMON status from status API.

Returns ZMON status.

Return type *dict*

token_login_url(*token: str*) → str
Return direct deeplink to ZMON one-time login.

Parameters **token** (str) – One-time token.

Returns Deeplink to ZMON one-time login.

Return type *str*

update_alert_definition(*alert_definition: dict*) → dict
Update existing alert definition.

Atrributes id, last_modified_by and check_definition_id are required. If status is not set, then it will be set to ACTIVE.

Parameters **alert_definition** (dict) – ZMON alert definition dict.

Returns Alert definition dict.

Return type *dict*

update_check_definition(*check_definition: dict*, *skip_validation: bool = False*) → dict
Update existing check definition.

Atrribute owning_team is required. If status is not set, then it will be set to ACTIVE.

Parameters

- **check_definition** (dict) – ZMON check definition dict.
- **skip_validation** (bool) – Skip validation of the check command syntax.

Returns Check definition dict.

Return type *dict*

update_dashboard(*dashboard: dict*) → dict
Create or update dashboard.

If dashboard has an id then dashboard will be updated, otherwise a new dashboard is created.

Parameters **dashboard** (int, str) – ZMON dashboard dict.

Returns Dashboard dict.

Return type *dict*

update_grafana_dashboard(*grafana_dashboard: dict*) → dict
Update existing Grafana dashboard.

Atrributes id and title are required.

Parameters **grafana_dashboard** (dict) – Grafana dashboard dict.

Returns Grafana dashboard dict.

Return type *dict*

static validate_check_command(*src*)
Validates if check command is valid syntax. Raises exception in case of invalid syntax.

Parameters **src** (str) – Check command python source code.

Raises ZmonError

CHAPTER 19

A Short Python Tutorial

This tutorial explains by example how to process a *dict* using Python's list comprehension facilities.

Suppose we're interested in the total number or order failures.

1. First, we need to query the appropriate endpoint to get the data, and call the `json()` method.

   ```
   http('http://www.example.com/foo/bar/data.json').json()
   ```

   This endpoint returns JSON data that is structured as follows (with much of the data omitted):

   ```
   {
       ...
       "itr-http04_orderfails": [1, 0],
       "itr-http05_addtocart": [0.05, 0.0875],
       "http17_addtocart": [0.075, 0.066667],
       "http27_requests": [14.666667, 12.195833],
       "http13_orderfails": [null, 2],
       ...
   }
   ```

   The parsed object will therefore be a *dict* mapping strings to lists of numbers, which may contain `None` values.

2. We need to find all entries ending in `_orderfails`. In Python, we can transform a *dict* in a list of tuples (*key*, *value*) using the `items()` method:

   ```
   http(...).json().items()
   ```

   We now need to filter this list to include only order failure information. Using a loop and an if statement, this could be accomplished like this:

   ```
   result = []
   for key, value in http(...).json().items():
       if key.endswith('_orderfails'):
           result.append(value)
   ```

(Note how the tuples in the list returned by `items()` are automatically "unpacked", their elements being assigned to `key` and `value`, respectively.)

Since the check command needs to be a single expression, not a series of statements, this is unfortunately not an option. Fortunately, Python provides a feature called list comprehension, which allows us to express the code above as follows:

```
[value for key, value in http(...).json().items() if key.endswith('_orderfails')]
```

That is, code of the form

```
result = []
for ELEMENT in LIST:
    if CONDITION:
        result.append(RESULT_ELEMENT)
```

becomes

```
[RESULT_ELEMENT for ELEMENT in LIST if CONDITION]
```

(The `if CONDITION` part is optional.)

We now have a list of lists `[[1, 0], [None, 2]]`.

3. In order to sum the list, we'd need to flatten it first, so that it has the form `[1, 0, None, 2]`. This can be accomplished with the *`chain()`* function. Given one or more iterable objects (such as lists), *`chain()`* returns a new iterable object produced by concatenating the given objects. That is

   ```
   chain([1, 0], [None, 2])
   ```

   would return

   ```
   [1, 0, None, 2]
   ```

   Unfortunately, the lists we want to chain are themselves elements of a list, and calling `chain([[1, 0], [None, 2]])` would just concatenate the list with nothing and return the it unchanged. We therefore need to tell Python to unpack the list, so that each of its elements becomes a new argument for the invocation of *`chain()`*.

   This can be accomplished by the `*` operator:

   ```
   chain(*[[1, 0], [None, 2]])
   ```

   That is, out expression is now

   ```
   chain(*[value for key, value in http(...).json().items() if key.endswith('_
   ↪orderfails')])
   ```

4. Now we need to remove that pesky `None` from the list. This could be accomplished with another list comprehension:

   ```
   [value for value in chain(...) if value is not None]
   ```

   For didactic reasons, we shall use the *`filter()`* function instead. *`filter()`* takes two arguments: a function that is called for each element in the filtered list and indicates whether that element should be in the resulting list, and the list that is to be filtered itself. We can create an anonymous function for this purpose using a lambda expression:

```
filter(lambda element: element is not None, chain(...))
```

In this case, we can use a somewhat obscure shortcut, though. If the function given to *filter()* is None, the identity function is used. Therefore, objects will be included in the resulting list if and only if they are "truthy", which None isn't. The integer 0 isn't truthy either, but this isn't a problem in this case since the presence or absence of zeros does not affect the sum. Therefore, we can use the expression

```
filter(None, chain(*[value for key, value in http(...).json().items() if key.
↪endswith('_orderfails')]))
```

5. Finally, we need to sum the elements of the list. For that, we can just use the *sum()* function, so that the expression is now

```
sum(filter(None, chain(*[value for key, value in http(...).json().items() if key.
↪endswith('_orderfails')])))
```

19.1 Python Recipes

Merging Data Into One Result

You can merge heterogeneous data into a single result object:

```
{
    'http_data': http(...).json()[...],
    'jmx_data':  jmx().query(...).results()[...],
    'sql_data':  sql().execute(...)[...],
}
```

Mapping SQL Results by ID

The SQL `results()` methods returns a list of maps (`[{'id': 1, 'data': 1000}, {'id': 2, 'data': 2000}]`). You can convert this to a single map (`{1: 1000, 2: 2000}`) like this:

```
{ row['id']: row['data'] for row in sql().execute(...).results() }
```

Using Multiple Captures

If you have a alert condition such as

```
FOO > 10 or BAR > 10
```

adding capures is a bit tricky. If you use

```
capture(foo=FOO) > 10 or capture(bar=BAR) > 10
```

and both `FOO` and `BAR` are greater than 10, only `foo` will be captured because the `or` uses short-circuit evaluation (`True or X` is true for all `X`, so `X` doesn't need to be evaluated). Instead, you can use

```
any([capture(foo=FOO) > 10, capture(bar=BAR) > 10])
```

which will always evaluate both comparisons and thus capture both values.

Defining Temporary Variables

You aren't supposed to be able to do define variables, but you can work around this restriction as follows:

```
(lambda x:
    # Some complex operation using x multiple times
```

```
)(
    x = sql().execute(...)  # Some complex or expensive query
)
```

Defining Functions

Since you can define variables with the trick above, you can also define functions:

```
(lambda f:
    # Some complex operation calling f multiple times
)(
    f = lambda a, b, c: sql().execute(...)  # Some code using the arguments a, b,␣
↪and c
)
```

CHAPTER 20

Tests

20.1 Acceptance and Unit Tests

These tests must be run from inside the vagrant box.:

```
$ vagrant ssh
vagrant@zmon:~$ cd /vagrant/vagrant/
vagrant@zmon:/vagrant/vagrant$ sudo ./test.sh
```

An example output of the previous command can look similar to this:

```
Starting Xvfb...
[13:36:12] Using gulpfile /vagrant/zmon-controller/src/main/webapp/gulpfile.js
[13:36:12] Starting 'test'...
Starting selenium standalone server...
Selenium standalone server started at http://10.0.2.15:47833/wd/hub
Testing dashboard features
  should display the search form - pass

Finished in 3.24 seconds
1 test, 1 assertion, 0 failures

Shutting down selenium standalone server.
[13:36:22] Finished 'test' after 10 s
```

Only one single acceptance test and no unit tests are provided so far. This is still a work in progress.

CHAPTER 21

Redis Data Structure

ZMON stores its primary working data in Redis. This page describes the used Redis keys and data structures.

Queues are Redis keys like `zmon:queue:<NAME>` of type "list", e.g. `zmon:queue:default`.

New queue items are added by the ZMON Scheduler via the Redis "rpush" command.

Important Redis key patterns are:

`zmon:queue:<QUEUE-NAME>` List of worker tasks for given queue.

`zmon:checks` Set of all executed check IDs.

`zmon:checks:<CHECK-ID>` Set of entity IDs having check results.

`zmon:checks:<CHECK-ID>:<ENTITY-ID>` List of last N check results. The first list item contains the most recent check result. Each check result is a JSON object with the keys `ts` (result timestamp), `td` (check duration), `value` (actual result value) and `worker` (ID of worker having produced the check result).

`zmon:alerts` Set of all active alert IDs.

`zmon:alerts:<ALERT-ID>` Set of entity IDs in alert state.

`zmon:alerts:<ALERT-ID>:entities` Hash of entity IDs to alert captures. This hash contains *all* entity IDs matched by the alert, i.e. not only entities in alert state.

`zmon:alerts:<ALERT-ID>:<ENTITY-ID>` Alert detail JSON containing alert start time, captures, worker, etc.

`zmon:downtimes` Set of all alert IDs having downtimes.

`zmon:downtimes:<ALERT-ID>` Set of all entity IDs having a downtime for this alert.

`zmon:downtimes:<ALERT-ID>:<ENTITY-ID>` Hash of downtimes for this entity/alert. Each hash value is a JSON object with keys `start_time`, `end_time` and `comment`.

`zmon:active_downtimes` Set of currently active downtimes. Each set item has the form `<ALERT-ID>:<ENTITY-ID>:<DOWNTIME-ID>`.

`zmon:metrics` Set of worker and scheduler IDs with metrics.

`zmon:metrics:<WORKER-OR-SCHEDULER-ID>:ts` Timestamp of last worker or scheduler metrics update.

zmon:metrics:<WORKER-OR-SCHEDULER-ID>:check.count Increasing counter of executed (or scheduled) checks.

CHAPTER 22

Glossary

alert definition Alert definitions define when to trigger an alert and for which entity. See *Alert Definitions*

alert condition Python expression defining the "threshold" when to trigger an alert. See *Condition*.

check command Python expression defining the value of a check. See *Check Command Reference*.

check definition A check definition provides a source of data for alerts to monitor. See *Check Definitions*

dashboard A dashboard is the main monitoring page of ZMON and consists of widgets and the list of active alerts. See *Dashboards*

downtime In ZMON, downtime refers to a period of time where certain alerts/entities should not be triggered. One use case for downtimes are scheduled maintenance works. See downtimes

entity Entities are "objects" to be monitored. Entities can be hosts, Zomcat instances, but they can also be more abstract things like app domains. See *Entities*

JSON JavaScript Object Notation. A minimal data interchange format. You probably already know it. If you don't, there's good documentation on its official page.

Markdown A simple markup language that can mostly pass for plain text. There's an introduction and a syntax reference on its official page.

time period Alert definition's time period can restrict its active alerting to certain time frames. This allows for alerts to be active e.g. only during work hours. See *Time periods*

YAML Not actually Yet Another Markup Language. A powerful but succinct data interchange format. This document should be sufficient to learn how to use YAML in ZMON. In case it isn't, the Wikipedia entry on YAML is actually slightly more useful that the official documentation.

Note that YAML is a strict superset of *JSON*. That is, wherever YAML is required, JSON can be used instead.

CHAPTER 23

Introduction

ZMON is a flexible and extensible open-source platform monitoring tool developed at Zalando and is in production use since early 2014. It offers proven scaling with its distributed nature and fast storage with KairosDB on top of Cassandra. ZMON splits checking(data acquisition) from the alerting responsibilities and uses abstract entities to describe what's being monitored. Its checks and alerts rely on Python expressions, giving the user a lot of power and connectivity. Besides the UI it provides RESTful APIs to manage and configure most properties automatically.

Anyone can use ZMON, but offers particular advantages for technical organizations with many autonomous teams. Its front end (see Demo / Bootstrap / Kubernetes/ Vagrant) comes with Grafana3 "built-in," enabling teams to create and manage their own data-driven dashboards along side ZMON's own team/personal dashboards for alerts and custom widgets. Being able to inherit and clone alerts makes it easier for teams to reuse and share code. Alerts can trigger HipChat, Slack, and E-Mail notifications. iOS and Android clients are works in progress, but push notifications are already implemented.

ZMON also enables painless integration with CMDBs and deployment tools. It also supports service discovery via custom adapters or its built-in entity service's REST API. For an example, see zmon-aws-agent to learn how we connect AWS service discovery with our monitoring in the cloud.

Feel free to contact us via slack.zmon.io.

CHAPTER 24

ZMON Components

A minimum ZMON setup requires these four components:

- zmon-controller: UI/Grafana/Oauth2 Login/Github Login
- zmon-scheduler: Scheduling check/alert evaluation
- zmon-worker: Doing the heavy lifting
- zmon-eventlog-service: History for state changes and modifications

Plus the storage covered in the *Requirements* section.

The following components are optional:

- zmon-cli: A command line client for managing entities/checks/alerts if needed
- zmon-aws-agent: Works with the AWS API to retrieve "known" applications
- zmon-data-service: API for multi DC federation: receiver for remote workers primarily
- zmon-metric-cache: Small scale special purpose metric store for API metrics in ZMON's cloud UI
- zmon-notification-service: Provides mobile API and push notification support for GCM to Android/iOS app
- zmon-android: An Android client for ZMON monitoring
- zmon-ios: An iOS client for ZMON monitoring

CHAPTER 25

ZMON Origins

ZMON was born in late 2013 during Zalando's annual Hack Week, when a group of Zalando engineers aimed to develop a replacement for ICINGA. Scalability, manageability and flexibility were all critical, as Zalando's small teams needed to be able to monitor their services independent of each other. In early 2014, Zalando teams began migrating all checks to ZMON, which continues to serve Zalando Tech.

CHAPTER 26

Entities

ZMON uses entities to describe your infrastructure or platform, and to bind check variables to fixed values.

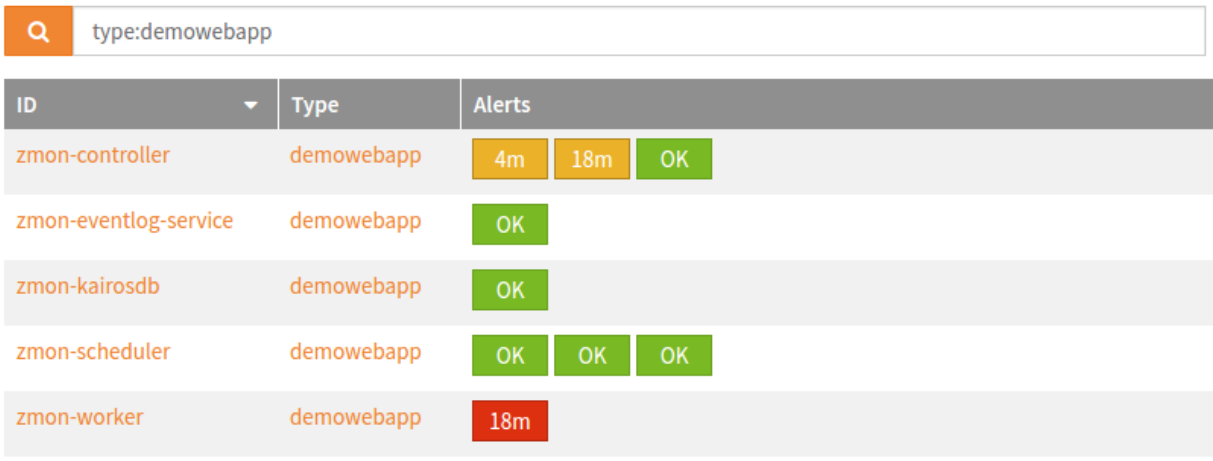
```
{
      "type":"host",
      "id":"cassandra01",
      "host":"cassandra01",
      "role":"cassandra-host",
      "ip":"192.168.1.17",
      "dc":"data-center-1"
}
```

Or more abstract objects:

```
{
      "type":"postgresql-cluster",
      "id":"article-cluster",
      "name":"article-cluster",
      "shards": {
              "shard1":"articledb01:5432/shard1",
              "shard2":"articledb02:5432/shard2"
      }
}
```

Entity properties are not defined in any schema, so you can add properties as you see fit. This enables finer-grained filtering or selection of entities later on. As an example, host entities can include a physical model to later select the proper hardware checks.

Below you see an exmple of the entity view with alerts per entity.


type:demowebapp
ID
Type
Alerts
zmon-controller
demowebapp
4m
18m
OK
zmon-eventlog-service
demowebapp
OK
zmon-kairosdb
demowebapp
OK
zmon-scheduler
demowebapp
OK
OK
OK
zmon-worker
demowebapp
18m

CHAPTER 27

Checks

A check describes how data is acquired. Its key properties are: a command to execute and an entity filter. The filter selects a subset of entities by requiring an overlap on specified properties. An example:

```
{
  "type":"postgresql-cluster", "name":"article-cluster"
}
```

The check command itself is an executable Python expression. ZMON provides many custom wrappers that bind to the selected entity. The following example uses a PostgreSQL wrapper to execute a query on every shard defined above:

```
# sql() in this context is aware of the "shards" property

sql().execute('SELECT count(1) FROM articles "total"').result()
```

A check command always returns a value to the alert. This can be of any Python type.

Not familiar with Python's functional expressions? No worries: ZMON allows you to define a top-level function and define your command in an easier, less functional way:

```
def check():
  # sql() binds to the entity used and thus knows the connection URLs
  return sql().execute('SELECT count(1) FROM articles "total"').result()
```

CHAPTER 28

Alerts

A basic alert consists of an alert condition, an entity filter, and a team. An alert has only two states: up or down. An alert is up if it yields anything but False; this also includes exceptions thrown during evaluation of the check or alert, e.g. in the event of connection problems. ZMON does not support levels of criticality, or something like "unknown", but you have a color option to customize sort and style on your dashboard (red, orange, yellow).

Let's revisit the above PostgreSQL check again. The alert below would either popup if there are no articles found or if we get an exception connecting to the PostgreSQL database.

```
team: database
entities:
  - type: postgresql-cluster
alert_condition: |
  value <= 0
```

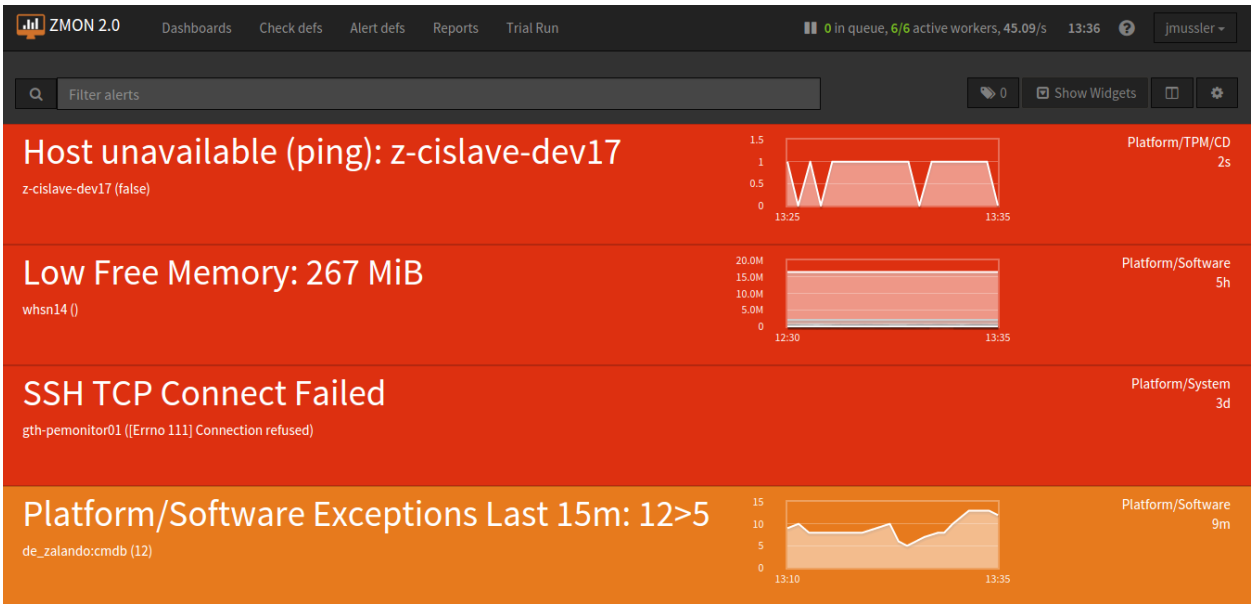
Alerts raised by exceptions are marked in the dashboard with a "!".

Via ZMON's UI, alerts support parameters to the alert condition. This makes it easy for teams/users to implement different thresholds, and — with the priority field defining the dashboard color — render their dashboards to reflect their priorities.

CHAPTER 29

Dashboards

Dashboards include a widget area where you can render important data with charts, gauges, or plain text. Another section features rendering of all active alerts for the team filter, defined at the dashboard level. Using the team filter, select the alerts you want your dashboard to include. Specify multiple teams, if necessary. TAGs are supported to subselect topics.



CHAPTER 30

REST API and CLI

To make your life easier, ZMON's REST API manages all the essential moving parts to support your daily work — creating and updating entities to allow for sync-up with your existing infrastructure. When you create and modify checks and alerts, the scheduler will quickly pick up these changes so you won't have to restart or deploy anything.

And ZMON's command line client - a slim wrapper around the REST API - also adds usability by making it simpler to work with YAML files or push collections of entities.

CHAPTER 31

Development Status

The team behind ZMON continues to improve performance and functionality. Please let us know via GitHub's issues tracker if you find any bugs or issues.

CHAPTER 32

Indices and Tables

- genindex
- modindex
- search

Index

D

E

F

G

H

I

J

K

L

U

V

X

Y

Z