grafanalib

grafanalib community

Nov 02, 2022

CONTENTS:

CHAPTER

ONE

GETTING STARTED WITH GRAFANALIB

Do you like Grafana but wish you could version your dashboard configuration? Do you find yourself repeating common patterns? If so, grafanalib is for you.

grafanalib lets you generate Grafana dashboards from simple Python scripts.

Grafana migrates dashboards to the latest Grafana schema version on import, meaning that dashboards created with grafanalib are supported by all versions of Grafana. You may find that some of the latest features are missing from grafanalib, please refer to the module documentation for information about supported features. If you find a missing feature please raise an issue or submit a PR to the GitHub repository

1.1 Writing dashboards

The following will configure a dashboard with a couple of example panels that use the random walk and Prometheus datasources.

```
from grafanalib.core import (
    Dashboard, TimeSeries, GaugePanel,
    Target, GridPos,
    OPS_FORMAT
)

dashboard = Dashboard(
    title="Python generated example dashboard",
    description="Example dashboard using the Random Walk and default Prometheus␣
↪datasource",
    tags=[
        'example'
    ],
    timezone="browser",
    panels=[
        TimeSeries(
            title="Random Walk",
            dataSource='default',
            targets=[
                Target(
                    datasource='grafana',
                    expr='example',
                ),
            ],
```

(continues on next page)

(continued from previous page)

```
            gridPos=GridPos(h=8, w=16, x=0, y=0),
        ),
        GaugePanel(
            title="Random Walk",
            dataSource='default',
            targets=[
                Target(
                    datasource='grafana',
                    expr='example',
                ),
            ],
            gridPos=GridPos(h=4, w=4, x=17, y=0),
        ),
        TimeSeries(
            title="Prometheus http requests",
            dataSource='prometheus',
            targets=[
                Target(
                    expr='rate(prometheus_http_requests_total[5m])',
                    legendFormat="{{ handler }}",
                    refId='A',
                ),
            ],
            unit=OPS_FORMAT,
            gridPos=GridPos(h=8, w=16, x=0, y=10),
        ),
    ],
).auto_panel_ids()
```

There is a fair bit of repetition here, but once you figure out what works for your needs, you can factor that out. See our Weave-specific customizations for inspiration.

1.2 Generating dashboards

If you save the above as `example.dashboard.py` (the suffix must be `.dashboard.py`), you can then generate the JSON dashboard with:

```
$ generate-dashboard -o frontend.json example.dashboard.py
```

1.3 Uploading dashboards from code

Sometimes you may need to generate and upload dashboard directly from Python code. The following example provides minimal code boilerplate for it:

```
from grafanalib.core import Dashboard
from grafanalib._gen import DashboardEncoder
import json
import requests
```

(continues on next page)

(continued from previous page)

```
from os import getenv


def get_dashboard_json(dashboard, overwrite=False, message="Updated by grafanlib"):
    '''
    get_dashboard_json generates JSON from grafanalib Dashboard object

    :param dashboard - Dashboard() created via grafanalib
    '''

    # grafanalib generates json which need to pack to "dashboard" root element
    return json.dumps(
        {
            "dashboard": dashboard.to_json_data(),
            "overwrite": overwrite,
            "message": message
        }, sort_keys=True, indent=2, cls=DashboardEncoder)


def upload_to_grafana(json, server, api_key, verify=True):
    '''
    upload_to_grafana tries to upload dashboard to grafana and prints response

    :param json - dashboard json generated by grafanalib
    :param server - grafana server name
    :param api_key - grafana api key with read and write privileges
    '''

    headers = {'Authorization': f"Bearer {api_key}", 'Content-Type': 'application/json'}
    r = requests.post(f"https://{server}/api/dashboards/db", data=json, headers=headers,␣
↪verify=verify)
    # TODO: add error handling
    print(f"{r.status_code} - {r.content}")


grafana_api_key = getenv("GRAFANA_API_KEY")
grafana_server = getenv("GRAFANA_SERVER")

my_dashboard = Dashboard(title="My awesome dashboard", uid='abifsd')
my_dashboard_json = get_dashboard_json(my_dashboard, overwrite=True)
upload_to_grafana(my_dashboard_json, grafana_server, grafana_api_key)
```

Alternatively Grafana supports file based provisioning, where dashboard files are periodically loaded into the Grafana database. Tools like Anisble can assist with the deployment.

1.4 Writing Alerts

Between Grafana versions there have been significant changes in how alerts are managed. Bellow are some example of how to configure alerting in Grafana v8 and Grafana v9.

1.4.1 Alerts in Grafana v8

The following will configure a couple of alerts inside a group.

```
"""Example grafana 8.x+ Alert"""


from grafanalib.core import (
    AlertGroup,
    AlertRulev8,
    Target,
    AlertCondition,
    LowerThan,
    OP_OR,
    OP_AND,
    RTYPE_LAST
)

# An AlertGroup is one group contained in an alert folder.
alertgroup = AlertGroup(
    name="Production Alerts",
    # Each AlertRule forms a separate alert.
    rules=[
        AlertRulev8(
            # Each rule must have a unique title
            title="Database is unresponsive",
            # Several triggers can be used per alert, a trigger is a combination of a␣
↪Target and its AlertCondition in a tuple.
            triggers=[
                (
                    # A target refId must be assigned, and exist only once per AlertRule.
                    Target(
                        expr='sum(kube_pod_container_status_ready{exported_pod=~
↪"database-/*"})',
                        # Set datasource to name of your datasource
                        datasource="VictoriaMetrics",
                        refId="A",
                    ),
                    AlertCondition(
                        evaluator=LowerThan(1),
                        # To have the alert fire when either of the triggers are met in␣
↪the rule, set both AlertCondition operators to OP_OR.
                        operator=OP_OR,
                        reducerType=RTYPE_LAST
                    )
                ),
                (
```

(continues on next page)

(continued from previous page)

```
                Target(
                    expr='sum by (app) (count_over_time({app="database"}[5m]))',
                    # Set datasource to name of your datasource
                    datasource="Loki",
                    refId="B",
                ),
                AlertCondition(
                    evaluator=LowerThan(1000),
                    operator=OP_OR,
                    reducerType=RTYPE_LAST
                )
            )
        ],
        annotations={
            "summary": "The database is down",
            "runbook_url": "runbook-for-this-scenario.com/foo",
        },
        labels={
            "environment": "prod",
            "slack": "prod-alerts",
        },
        evaluateInterval="1m",
        evaluateFor="3m",
    ),

    # Second alert
    AlertRulev8(
        title="Service API blackbox failure",
        triggers=[
            (
                Target(
                    expr='probe_success{instance="my-service.foo.com/ready"}',
                    # Set datasource to name of your datasource
                    datasource="VictoriaMetrics",
                    refId="A",
                ),
                AlertCondition(
                    evaluator=LowerThan(1),
                    operator=OP_AND,
                    reducerType=RTYPE_LAST,
                )
            )
        ],
        annotations={
            "summary": "Service API has been unavailable for 3 minutes",
            "runbook_url": "runbook-for-this-scenario.com/foo",
        },
        labels={
            "environment": "prod",
            "slack": "prod-alerts",
        },
        evaluateInterval="1m",
```

(continues on next page)

(continued from previous page)

```
            evaluateFor="3m",
        )
    ]
)
```

Although this example has a fair amount of boilerplate, when creating large numbers of similar alerts it can save lots of time to programmatically fill these fields.

Each `AlertGroup` represents a folder within Grafana's alerts tab. This consists of one or more `AlertRulev8`, which contains one or more triggers. Triggers define what will cause the alert to fire.

A trigger is made up of a `Target` (a Grafana query on a datasource) and an `AlertCondition` (a condition this query must satisfy in order to alert).

Finally, there are additional settings like:

- How the alert will behave when data sources have problems (`noDataAlertState` and `errorAlertState`)
- How frequently the trigger is evaluated (`evaluateInterval`)
- How long the AlertCondition needs to be met before the alert fires (`evaluateFor`)
- Annotations and labels, which help provide contextual information and direct where your alerts will go

1.4.2 Alerts in Grafana v9

The following will configure a couple of alerts inside a group for Grafana v9.x+.

```
"""Example grafana 9.x+ Alert"""


from grafanalib.core import (
    AlertGroup,
    AlertRulev9,
    Target,
    AlertCondition,
    AlertExpression,
    GreaterThan,
    OP_AND,
    RTYPE_LAST,
    EXP_TYPE_CLASSIC,
    EXP_TYPE_REDUCE,
    EXP_TYPE_MATH
)

# An AlertGroup is one group contained in an alert folder.
alertgroup = AlertGroup(
    name="Production Alerts",
    # Each AlertRule forms a separate alert.
    rules=[
        # Alert rule using classic condition > 3
        AlertRulev9(
            # Each rule must have a unique title
            title="Alert for something 1",
            uid='alert1',
```

(continues on next page)

(continued from previous page)

```
            # Several triggers can be used per alert
            condition='B',
            triggers=[
                # A target refId must be assigned, and exist only once per AlertRule.
                Target(
                    expr="from(bucket: \"sensors\")\n  |> range(start: v.timeRangeStart,␣
↪stop: v.timeRangeStop)\n  |> filter(fn: (r) => r[\"_measurement\"] == \"remote_cpu\")\
↪n  |> filter(fn: (r) => r[\"_field\"] == \"usage_system\")\n  |> filter(fn: (r) => r[\
↪"cpu\"] == \"cpu-total\")\n  |> aggregateWindow(every: v.windowPeriod, fn: mean,␣
↪createEmpty: false)\n  |> yield(name: \"mean\")",
                    # Set datasource to name of your datasource
                    datasource="influxdb",
                    refId="A",
                ),
                AlertExpression(
                    refId="B",
                    expressionType=EXP_TYPE_CLASSIC,
                    expression='A',
                    conditions=[
                        AlertCondition(
                            evaluator=GreaterThan(3),
                            operator=OP_AND,
                            reducerType=RTYPE_LAST
                        )
                    ]
                )
            ],
            annotations={
                "summary": "The database is down",
                "runbook_url": "runbook-for-this-scenario.com/foo",
            },
            labels={
                "environment": "prod",
                "slack": "prod-alerts",
            },
            evaluateFor="3m",
        ),
        # Alert rule using reduce and Math
        AlertRulev9(
            # Each rule must have a unique title
            title="Alert for something 2",
            uid='alert2',
            condition='C',
            # Several triggers can be used per alert
            triggers=[
                # A target refId must be assigned, and exist only once per AlertRule.
                Target(
                    expr="from(bucket: \"sensors\")\n  |> range(start: v.timeRangeStart,␣
↪stop: v.timeRangeStop)\n  |> filter(fn: (r) => r[\"_measurement\"] == \"remote_cpu\")\
↪n  |> filter(fn: (r) => r[\"_field\"] == \"usage_system\")\n  |> filter(fn: (r) => r[\
↪"cpu\"] == \"cpu-total\")\n  |> aggregateWindow(every: v.windowPeriod, fn: mean,␣
↪createEmpty: false)\n  |> yield(name: \"mean\")",
```

(continues on next page)

(continued from previous page)

```
                    # Set datasource to name of your datasource
                    datasource="influxdb",
                    refId="A",
                ),
                AlertExpression(
                    refId="B",
                    expressionType=EXP_TYPE_REDUCE,
                    expression='A',
                    reduceFunction='mean',
                    reduceMode='dropNN'
                ),
                AlertExpression(
                    refId="C",
                    expressionType=EXP_TYPE_MATH,
                    expression='$B < 3'
                )
            ],
            annotations={
                "summary": "The database is down",
                "runbook_url": "runbook-for-this-scenario.com/foo",
            },
            labels={
                "environment": "prod",
                "slack": "prod-alerts",
            },
            evaluateFor="3m",
        )
    ]
)
```

Although this example has a fair amount of boilerplate, when creating large numbers of similar alerts it can save lots of time to programmatically fill these fields.

Each `AlertGroup` represents a folder within Grafana's alerts tab. This consists of one or more `AlertRulev9`, which contains a list of triggers, that define what will cause the alert to fire.

A trigger can either be a `Target` (a Grafana query on a datasource) or an `AlertExpression` (a expression performed on one of the triggers).

An `AlertExpression` can be one of 4 types

- Classic - Contains and list of `AlertCondition`'s that are evaluated
- Reduce - Reduce the queried data
- Resample - Resample the queried data
- Math - Expression with the condition for the rule

Finally, there are additional settings like:

- How the alert will behave when data sources have problems (`noDataAlertState` and `errorAlertState`)
- How frequently the each rule in the Alert Group is evaluated (`evaluateInterval`)
- How long the AlertCondition needs to be met before the alert fires (`evaluateFor`)
- Annotations and labels, which help provide contextual information and direct where your alerts will go

1.5 Generating Alerts

If you save either of the above examples for Grafana v8 or v9 as `example.alertgroup.py` (the suffix must be `.alertgroup.py`), you can then generate the JSON alert with:

```
$ generate-alertgroup -o alerts.json example.alertgroup.py
```

1.6 Uploading alerts from code

As Grafana does not currently have a user interface for importing alertgroup JSON, you must either upload the alerts via Grafana's REST API or use file based provisioning.

1.6.1 Uploading alerts from code using REST API

The following example provides minimal code boilerplate for it:

```
from grafanalib.core import AlertGroup
from grafanalib._gen import DashboardEncoder, loader
import json
import requests
from os import getenv


def get_alert_json(alert: AlertGroup):
    '''
    get_alert_json generates JSON from grafanalib AlertGroup object

    :param alert - AlertGroup created via grafanalib
    '''

    return json.dumps(alert.to_json_data(), sort_keys=True, indent=4,␣
↪cls=DashboardEncoder)


def upload_to_grafana(alertjson, folder, server, api_key, session_cookie, verify=True):
    '''
    upload_to_grafana tries to upload the AlertGroup to grafana and prints response
    WARNING: This will first delete all alerts in the AlertGroup before replacing them␣
↪with the provided AlertGroup.

    :param alertjson - AlertGroup json generated by grafanalib
    :param folder - Folder to upload the AlertGroup into
    :param server - grafana server name
    :param api_key - grafana api key with read and write privileges
    '''
    groupName = json.loads(alertjson)['name']

    headers = {}
    if api_key:
        print("using bearer auth")
```

(continues on next page)

(continued from previous page)

```
        headers['Authorization'] = f"Bearer {api_key}"

    if session_cookie:
        print("using session cookie")
        headers['Cookie'] = session_cookie

    print(f"deleting AlertGroup {groupName} in folder {folder}")
    r = requests.delete(f"https://{server}/api/ruler/grafana/api/v1/rules/{folder}/
↪{groupName}", headers=headers, verify=verify)
    print(f"{r.status_code} - {r.content}")

    headers['Content-Type'] = 'application/json'

    print(f"ensuring folder {folder} exists")
    r = requests.post(f"https://{server}/api/folders", data={"title": folder},␣
↪headers=headers)
    print(f"{r.status_code} - {r.content}")

    print(f"uploading AlertGroup {groupName} to folder {folder}")
    r = requests.post(f"https://{server}/api/ruler/grafana/api/v1/rules/{folder}",␣
↪data=alertjson, headers=headers, verify=verify)
    # TODO: add error handling
    print(f"{r.status_code} - {r.content}")


grafana_api_key = getenv("GRAFANA_API_KEY")
grafana_server = getenv("GRAFANA_SERVER")
grafana_cookie = getenv("GRAFANA_COOKIE")

# Generate an alert from the example
my_alergroup_json = get_alert_json(loader("./grafanalib/tests/examples/example.
↪alertgroup.py"))
upload_to_grafana(my_alergroup_json, "testfolder", grafana_server, grafana_api_key,␣
↪grafana_cookie)
```

1.6.2 Uploading alerts from code using File Based Provisioning

The alternative to using Grafana's REST API is to use its file based provisioning for alerting.

The following example uses the `AlertFileBasedProvisioning` class to provision a list of alert groups:

```
"""Example grafana 9.x+ Alert"""


from grafanalib.core import (
    AlertGroup,
    AlertRulev9,
    Target,
    AlertCondition,
    AlertExpression,
    AlertFileBasedProvisioning,
```

(continues on next page)

(continued from previous page)

```
    GreaterThan,
    OP_AND,
    RTYPE_LAST,
    EXP_TYPE_CLASSIC,
    EXP_TYPE_REDUCE,
    EXP_TYPE_MATH
)

# An AlertGroup is one group contained in an alert folder.
alertgroup = AlertGroup(
    name="Production Alerts",
    # Each AlertRule forms a separate alert.
    rules=[
        # Alert rule using classic condition > 3
        AlertRulev9(
            # Each rule must have a unique title
            title="Alert for something 3",
            uid='alert3',
            # Several triggers can be used per alert
            condition='B',
            triggers=[
                # A target refId must be assigned, and exist only once per AlertRule.
                Target(
                    expr="from(bucket: \"sensors\")\n  |> range(start: v.timeRangeStart,␣
↪stop: v.timeRangeStop)\n  |> filter(fn: (r) => r[\"_measurement\"] == \"remote_cpu\")\
↪n  |> filter(fn: (r) => r[\"_field\"] == \"usage_system\")\n  |> filter(fn: (r) => r[\
↪"cpu\"] == \"cpu-total\")\n  |> aggregateWindow(every: v.windowPeriod, fn: mean,␣
↪createEmpty: false)\n  |> yield(name: \"mean\")",
                    # Set datasource to name of your datasource
                    datasource="influxdb",
                    refId="A",
                ),
                AlertExpression(
                    refId="B",
                    expressionType=EXP_TYPE_CLASSIC,
                    expression='A',
                    conditions=[
                        AlertCondition(
                            evaluator=GreaterThan(3),
                            operator=OP_AND,
                            reducerType=RTYPE_LAST
                        )
                    ]
                )
            ],
            annotations={
                "summary": "The database is down",
                "runbook_url": "runbook-for-this-scenario.com/foo",
            },
            labels={
                "environment": "prod",
                "slack": "prod-alerts",
```

(continues on next page)

(continued from previous page)

```
            },
            evaluateFor="3m",
        ),
        # Alert rule using reduce and Math
        AlertRulev9(
            # Each rule must have a unique title
            title="Alert for something 4",
            uid='alert4',
            condition='C',
            # Several triggers can be used per alert
            triggers=[
                # A target refId must be assigned, and exist only once per AlertRule.
                Target(
                    expr="from(bucket: \"sensors\")\n  |> range(start: v.timeRangeStart,␣
↪stop: v.timeRangeStop)\n  |> filter(fn: (r) => r[\"_measurement\"] == \"remote_cpu\")\
↪n  |> filter(fn: (r) => r[\"_field\"] == \"usage_system\")\n  |> filter(fn: (r) => r[\
↪"cpu\"] == \"cpu-total\")\n  |> aggregateWindow(every: v.windowPeriod, fn: mean,␣
↪createEmpty: false)\n  |> yield(name: \"mean\")",
                    # Set datasource to name of your datasource
                    datasource="influxdb",
                    refId="A",
                ),
                AlertExpression(
                    refId="B",
                    expressionType=EXP_TYPE_REDUCE,
                    expression='A',
                    reduceFunction='mean',
                    reduceMode='dropNN'
                ),
                AlertExpression(
                    refId="C",
                    expressionType=EXP_TYPE_MATH,
                    expression='$B < 3'
                )
            ],
            annotations={
                "summary": "The database is down",
                "runbook_url": "runbook-for-this-scenario.com/foo",
            },
            labels={
                "environment": "prod",
                "slack": "prod-alerts",
            },
            evaluateFor="3m",
        )
    ]
)

alertfilebasedprovisioning = AlertFileBasedProvisioning([alertgroup])
```

Save the above example as `example.alertfilebasedprovisioning.py` (the suffix must be `.alertfilebasedprovisioning.py`), you can then generate the JSON alert with:

```
$ generate-alertgroup -o alerts.json example.alertfilebasedprovisioning.py
```

Then place the file in the `provisioning/alerting` directory and start Grafana Tools like Anisble can assist with the deployment of the alert file.

1.7 Installation

grafanalib is just a Python package, so:

```
$ pip install grafanalib
```

1.8 Support

This library is in its very early stages. We'll probably make changes that break backwards compatibility, although we'll try hard not to.

grafanalib works with Python 3.7, 3.8, 3.9 and 3.10.

1.9 Developing

If you're working on the project, and need to build from source, it's done as follows:

```
$ virtualenv .env
$ . ./.env/bin/activate
$ pip install -e .
```

1.10 Configuring Grafana Datasources

This repo used to contain a program `gfdatasource` for configuring Grafana data sources, but it has been retired since Grafana now has a built-in way to do it. See https://grafana.com/docs/administration/provisioning/#datasources

CHAPTER

TWO

GRAFANALIB PACKAGE

2.1 Submodules

2.2 grafanalib.cloudwatch module

Helpers to create Cloudwatch-specific Grafana queries.

class grafanalib.cloudwatch.**CloudwatchLogsInsightsTarget**(*expression=''*, *id=''*, *logGroupNames=[]*, *namespace=''*, *refId=''*, *region='default'*, *statsGroups=[]*, *hide=False*, *datasource=None*)

Bases: `object`

Generates Cloudwatch Logs Insights target JSON structure.

Grafana docs on using Cloudwatch: https://grafana.com/docs/grafana/latest/datasources/cloudwatch/

AWS docs on Cloudwatch Logs Insights: https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/AnalyzingLogData.html

Parameters

- **expression** – Cloudwatch Logs Insights expressions
- **id** – unique id
- **logGroupNames** – List of Cloudwatch log groups to query
- **namespace** – Cloudwatch namespace
- **refId** – target reference id
- **region** – Cloudwatch region
- **statsGroups** – Cloudwatch statsGroups
- **hide** – controls if given metric is displayed on visualization
- **datasource** – Grafana datasource name

to_json_data()

class grafanalib.cloudwatch.**CloudwatchMetricsTarget**(*alias=''*, *dimensions={}*, *expression=''*, *id=''*, *matchExact=True*, *metricName=''*, *namespace=''*, *period=''*, *refId=''*, *region='default'*, *statistics=['Average']*, *hide=False*, *datasource=None*)

Bases: `object`

Generates Cloudwatch target JSON structure.

Grafana docs on using Cloudwatch: https://grafana.com/docs/grafana/latest/datasources/cloudwatch/

AWS docs on Cloudwatch metrics: https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/aws-services-cloudwatch-metrics.html

Parameters

- **alias** – legend alias
- **dimensions** – Cloudwatch dimensions dict
- **expression** – Cloudwatch Metric math expressions
- **id** – unique id
- **matchExact** – Only show metrics that exactly match all defined dimension names.
- **metricName** – Cloudwatch metric name
- **namespace** – Cloudwatch namespace
- **period** – Cloudwatch data period
- **refId** – target reference id
- **region** – Cloudwatch region
- **statistics** – Cloudwatch mathematic statistic
- **hide** – controls if given metric is displayed on visualization
- **datasource** – Grafana datasource name

`to_json_data()`

2.3 grafanalib.core module

Low-level functions for building Grafana dashboards.

The functions in this module don't enforce Weaveworks policy, and only mildly encourage it by way of some defaults. Rather, they are ways of building arbitrary Grafana JSON.

class `grafanalib.core.`**`Ae3ePlotly`**(*dataSource=None*, *targets=NOTHING*, *title=''*, *cacheTimeout=None*, *description=None*, *editable=True*, *error=False*, *height=None*, *gridPos=None*, *hideTimeOverride=False*, *id=None*, *interval=None*, *links=NOTHING*, *maxDataPoints=100*, *minSpan=None*, *repeat=NOTHING*, *span=None*, *thresholds=NOTHING*, *thresholdType='absolute'*, *timeFrom=None*, *timeShift=None*, *transparent=False*, *transformations=NOTHING*, *extraJson=None*, *configuration=NOTHING*, *data=NOTHING*, *layout=NOTHING*, *script="console.log(data)\n var trace = {\n x: data.series[0].fields[0].values.buffer,\n y: data.series[0].fields[1].values.buffer\n };\n return {data:[trace],layout:{title:'My Chart'}};"*, *clickScript=''*)

Bases: *Panel*

Generates ae3e plotly panel json structure GitHub repo of the panel: https://github.com/ae3e/ae3e-plotly-panel :param configuration in json format: Plotly configuration. Docs: https://plotly.com/python/configuration-options/ :param data: Plotly data: https://plotly.com/python/figure-structure/ :param layout: Layout of the chart in json format. Plotly docs: https://plotly.com/python/reference/layout/ :param script: Script executed whenever new data is available. Must return an object with one or more of the

following properties : data, layout, config f(data, variables){…your code…}

Parameters

clickScript – Script executed when chart is clicked. f(data){…your code…}

to_json_data()

class grafanalib.core.**Alert**(*name*, *message*, *alertConditions*, *executionErrorState='alerting'*, *frequency='60s'*, *handler=1*, *noDataState='no_data'*, *notifications=NOTHING*, *gracePeriod='5m'*, *alertRuleTags=NOTHING*)

Bases: `object`

Parameters

alertRuleTags – Key Value pairs to be sent with Alert notifications.

to_json_data()

class grafanalib.core.**AlertCondition**(*target=None*, *evaluator=None*, *timeRange=None*, *operator='and'*, *reducerType='last'*, ***, *type='query'*)

Bases: `object`

A condition on an alert.

Parameters

- **target** (`Target`) – Metric the alert condition is based on. Not required at instantiation for Grafana 8.x alerts.
- **evaluator** (`Evaluator`) – How we decide whether we should alert on the metric. e.g. `GreaterThan(5)` means the metric must be greater than 5 to trigger the condition. See `GreaterThan`, `LowerThan`, `WithinRange`, `OutsideRange`, `NoValue`.
- **timeRange** (`TimeRange`) – How long the condition must be true for before we alert. For Grafana 8.x alerts, this should be specified in the AlertRule instead.
- **operator** – One of `OP_AND` or `OP_OR`. How this condition combines with other conditions.
- **reducerType** – RTYPE_* Supported reducer types: RTYPE_AVG = 'avg' RTYPE_MIN = 'min' RTYPE_MAX = 'max' RTYPE_SUM = 'sum' RTYPE_COUNT = 'count' RTYPE_LAST = 'last' RTYPE_MEDIAN = 'median' RTYPE_DIFF = 'diff' RTYPE_PERCENT_DIFF = 'percent_diff' RTYPE_COUNT_NON_NULL = 'count_non_null'
- **type** – CTYPE_*

to_json_data()

class grafanalib.core.**AlertExpression**(*refId*, *expression*, *conditions=NOTHING*, *expressionType='classic_conditions'*, *hide=False*, *intervalMs=1000*, *maxDataPoints=43200*, *reduceFunction='mean'*, *reduceMode='strict'*, *reduceReplaceWith=0*, *resampleWindow='10s'*, *resampleDownsampler='mean'*, *resampleUpsampler='fillna'*)

Bases: `object`

A alert expression to be evaluated in Grafana v9.x+

Parameters

- **refId** – Expression reference ID (A,B,C,D,...)
- **expression** – Input reference ID (A,B,C,D,...) for expression to evaluate, or in the case of the Math type the expression to evaluate
- **conditions** – list of AlertConditions
- **expressionType** – Expression type EXP_TYPE_* Supported expression types: EXP_TYPE_CLASSIC EXP_TYPE_REDUCE EXP_TYPE_RESAMPLE EXP_TYPE_MATH
- **hide** – Hide alert boolean
- **intervalMs** – Expression evaluation interval
- **maxDataPoints** – Maximum number fo data points to be evaluated
- **reduceFunction** – Reducer function (Only used if expressionType=EXP_TYPE_REDUCE) Supported reducer functions: EXP_REDUCER_FUNC_MIN EXP_REDUCER_FUNC_MAX EXP_REDUCER_FUNC_MEAN EXP_REDUCER_FUNC_SUM EXP_REDUCER_FUNC_COUNT EXP_REDUCER_FUNC_LAST
- **reduceMode** – Reducer mode (Only used if expressionType=EXP_TYPE_REDUCE) Supported reducer modes: EXP_REDUCER_MODE_STRICT EXP_REDUCER_FUNC_DROP_NN EXP_REDUCER_FUNC_REPLACE_NN
- **reduceReplaceWith** – When using mode EXP_REDUCER_FUNC_REPLACE_NN number that will replace non numeric values
- **resampleWindow** – Intervale to resample to eg. 10s, 1m, 30m, 1h
- **resampleDownsampler** – 'mean', 'min', 'max', 'sum'
- **resampleUpsampler** – 'fillna' - Fill with NaN's 'pad' - fill with the last known value 'backfilling' - fill with the next know value

to_json_data()

class grafanalib.core.**AlertFileBasedProvisioning**(*groups*)

Bases: `object`

Used to generate JSON data valid for file based alert provisioning

param alertGroup: List of AlertGroups

to_json_data()

class grafanalib.core.**AlertGroup**(*name*, *rules=NOTHING*, *folder='alert'*, *evaluateInterval='1m'*)

Bases: `object`

Create an alert group of Grafana 8.x alerts

Parameters

- **name** – Alert group name
- **rules** – List of AlertRule

- **folder** – Folder to hold alert (Grafana 9.x)
- **evaluateInterval** – Interval at which the group of alerts is to be evaluated

group_rules(*rules*)

to_json_data()

class grafanalib.core.**AlertList**(*dashboardTags=NOTHING*, *description=''*, *gridPos=None*, *id=None*, *limit=10*, *links=NOTHING*, *nameFilter=''*, *onlyAlertsOnDashboard=True*, *show='current'*, *sortOrder=1*, *span=6*, *stateFilter=NOTHING*, *title=''*, *transparent=False*, *alertName=''*)

Bases: object

Generates the AlertList Panel.

Parameters

- **dashboardTags** – A list of tags (strings) for the panel.
- **description** – Panel description, supports markdown and links.
- **gridPos** – describes the panel size and position in grid coordinates.
- **id** – panel id
- **limit** – Max number of alerts that can be displayed in the list.
- **nameFilter** – Show only alerts that contain nameFilter in their name.
- **onlyAlertsOnDashboard** – If true, shows only alerts from the current dashboard.
- **links** – Additional web links to be presented in the panel. A list of instantiation of DataLink objects.
- **show** – Show the current alert list (ALERTLIST_SHOW_CURRENT) or only the alerts that were changed (ALERTLIST_SHOW_CHANGES).
- **sortOrder** – Defines the sorting order of the alerts. Gets one of the following values as input: SORT_ASC, SORT_DESC and SORT_IMPORTANCE.
- **span** – Defines the number of spans that will be used for the panel.
- **stateFilter** – Show alerts with statuses from the stateFilter list. The list can contain a subset of the following statuses: [ALERTLIST_STATE_ALERTING, ALERTLIST_STATE_OK, ALERTLIST_STATE_NO_DATA, ALERTLIST_STATE_PAUSED, ALERTLIST_STATE_EXECUTION_ERROR, ALERTLIST_STATE_PENDING]. An empty list means all alerts.
- **title** – The panel title.
- **transparent** – If true, display the panel without a background.
- **alertName** – Show only alerts that contain alertName in their name.

to_json_data()

class grafanalib.core.**AlertRulev8**(*title*, *triggers*, *annotations={}*, *labels={}*, *evaluateInterval='1m'*, *evaluateFor='5m'*, *noDataAlertState='Alerting'*, *errorAlertState='Alerting'*, *timeRangeFrom=300*, *timeRangeTo=0*, *uid=None*, *dashboard_uid=''*, *panel_id=0*, *rule_group=''*)

Bases: object

Create a Grafana 8.x Alert Rule

Parameters

- **title** – The alert's title, must be unique per folder
- **triggers** – A list of Target and AlertCondition tuples, [(Target, AlertCondition)]. The Target specifies the query, and the AlertCondition specifies how this is used to alert. Several targets and conditions can be defined, alerts can fire based on all conditions being met by specifying OP_AND on all conditions, or on any single condition being met by specifying OP_OR on all conditions.
- **annotations** – Summary and annotations
- **labels** – Custom Labels for the metric, used to handle notifications
- **evaluateInterval** – The frequency of evaluation. Must be a multiple of 10 seconds. For example, 30s, 1m
- **evaluateFor** – The duration for which the condition must be true before an alert fires
- **noDataAlertState** – Alert state if no data or all values are null Must be one of the following: [ALERTRULE_STATE_DATA_OK, ALERTRULE_STATE_DATA_ALERTING, ALERTRULE_STATE_DATA_NODATA]
- **errorAlertState** – Alert state if execution error or timeout Must be one of the following: [ALERTRULE_STATE_DATA_OK, ALERTRULE_STATE_DATA_ALERTING, ALERTRULE_STATE_DATA_ERROR]
- **timeRangeFrom** – Time range interpolation data start from
- **timeRangeTo** – Time range interpolation data finish at
- **uid** – Alert UID should be unique
- **dashboard_uid** – Dashboard UID that should be use for linking on alert message
- **panel_id** – Panel ID that should should be use for linking on alert message

to_json_data()

class grafanalib.core.**AlertRulev9**(*title*, *triggers=[]*, *annotations={}*, *labels={}*, *evaluateFor='5m'*, *noDataAlertState='Alerting'*, *errorAlertState='Alerting'*, *condition='B'*, *timeRangeFrom=300*, *timeRangeTo=0*, *uid=None*, *dashboard_uid=''*, *panel_id=0*)

Bases: `object`

Create a Grafana 9.x+ Alert Rule

Parameters

- **title** – The alert's title, must be unique per folder
- **triggers** – A list of Targets and AlertConditions. The Target specifies the query, and the AlertCondition specifies how this is used to alert.
- **annotations** – Summary and annotations Dictionary with one of the following key or custom key ['runbook_url', 'summary', 'description', '__alertId__', '__dashboardUid__', '__panelId__']
- **labels** – Custom Labels for the metric, used to handle notifications
- **condition** – Set one of the queries or expressions as the alert condition by refID (Grafana 9.x)
- **evaluateFor** – The duration for which the condition must be true before an alert fires The Interval is set by the alert group

- **noDataAlertState** – Alert state if no data or all values are null Must be one of the following: [ALERTRULE_STATE_DATA_OK, ALERTRULE_STATE_DATA_ALERTING, ALERTRULE_STATE_DATA_NODATA]
- **errorAlertState** – Alert state if execution error or timeout Must be one of the following: [ALERTRULE_STATE_DATA_OK, ALERTRULE_STATE_DATA_ALERTING, ALERTRULE_STATE_DATA_ERROR]
- **timeRangeFrom** – Time range interpolation data start from
- **timeRangeTo** – Time range interpolation data finish at
- **uid** – Alert UID should be unique
- **dashboard_uid** – Dashboard UID that should be use for linking on alert message
- **panel_id** – Panel ID that should should be use for linking on alert message

to_json_data()

class grafanalib.core.**Annotations**(*list=NOTHING*)

Bases: object

to_json_data()

class grafanalib.core.**BarGauge**(*dataSource=None*, *targets=NOTHING*, *title=''*, *cacheTimeout=None*, *description=None*, *editable=True*, *error=False*, *height=None*, *gridPos=None*, *hideTimeOverride=False*, *id=None*, *interval=None*, *links=NOTHING*, *maxDataPoints=100*, *minSpan=None*, *repeat=NOTHING*, *span=None*, *thresholdType='absolute'*, *timeFrom=None*, *timeShift=None*, *transparent=False*, *transformations=NOTHING*, *extraJson=None*, *allValues=False*, *calc='mean'*, *dataLinks=NOTHING*, *decimals=None*, *displayMode='lcd'*, *format='none'*, *label=None*, *limit=None*, *max=100*, *min=0*, *orientation='horizontal'*, *rangeMaps=NOTHING*, *thresholdLabels=False*, *thresholdMarkers=True*, *thresholds=NOTHING*, *valueMaps=NOTHING*)

Bases: *Panel*

Generates Bar Gauge panel json structure

Parameters

- **allValue** – If All values should be shown or a Calculation
- **calc** – Calculation to perform on metrics
- **dataLinks** – list of data links hooked to datapoints on the graph
- **decimals** – override automatic decimal precision for legend/tooltips
- **displayMode** – style to display bar gauge in
- **format** – defines value units
- **labels** – option to show gauge level labels
- **limit** – limit of number of values to show when not Calculating
- **max** – maximum value of the gauge
- **min** – minimum value of the gauge
- **orientation** – orientation of the bar gauge
- **rangeMaps** – the list of value to text mappings

- **thresholdLabel** – label for gauge. Template Variables: "$__series_namei" "$__field_name" "$__cell_{N} / $__calc"
- **thresholdMarkers** – option to show marker of level on gauge
- **thresholds** – single stat thresholds
- **valueMaps** – the list of value to text mappings

to_json_data()

class grafanalib.core.**Column**(*text='Avg'*, *value='avg'*)

Bases: object

Details of an aggregation column in a table panel.

Parameters

- **text** – name of column
- **value** – aggregation function

to_json_data()

class grafanalib.core.**ColumnSort**(*col=None*, *desc=False*)

Bases: object

to_json_data()

class grafanalib.core.**ColumnStyle**(*alias=''*, *pattern=''*, *align='auto'*, *link=False*, *linkOpenInNewTab=False*, *linkUrl=''*, *linkTooltip=''*, *type=NOTHING*)

Bases: object

to_json_data()

class grafanalib.core.**ConstantInput**(*name*, *label*, *value*, *description=''*)

Bases: object

to_json_data()

class grafanalib.core.**Dashboard**(*title*, *annotations=NOTHING*, *description=''*, *editable=True*, *gnetId=None*, *graphTooltip=0*, *hideControls=False*, *id=None*, *inputs=NOTHING*, *links=NOTHING*, *panels=NOTHING*, *refresh='10s'*, *rows=NOTHING*, *schemaVersion=12*, *sharedCrosshair=False*, *style='dark'*, *tags=NOTHING*, *templating=NOTHING*, *time=NOTHING*, *timePicker=NOTHING*, *timezone='utc'*, *version=0*, *uid=None*)

Bases: object

auto_panel_ids()

Give unique IDs all the panels without IDs.

Returns a new Dashboard that is the same as this one, except all of the panels have their id property set. Any panels which had an id property set will keep that property, all others will have auto-generated IDs provided for them.

to_json_data()

class grafanalib.core.**DashboardLink**(*dashboard*, *uri*, *keepTime=True*, *title=None*, *type='dashboard'*)

Bases: object

to_json_data()

class grafanalib.core.**DashboardList**(*dataSource=None*, *targets=NOTHING*, *title=''*, *cacheTimeout=None*, *description=None*, *editable=True*, *error=False*, *height=None*, *gridPos=None*, *hideTimeOverride=False*, *id=None*, *interval=None*, *links=NOTHING*, *maxDataPoints=100*, *minSpan=None*, *repeat=NOTHING*, *span=None*, *thresholds=NOTHING*, *thresholdType='absolute'*, *timeFrom=None*, *timeShift=None*, *transparent=False*, *transformations=NOTHING*, *extraJson=None*, *showHeadings=True*, *showSearch=False*, *showRecent=False*, *showStarred=True*, *maxItems=10*, *searchQuery=''*, *searchTags=NOTHING*, *overrides=NOTHING*)

Bases: *Panel*

Generates Dashboard list panel json structure Grafana doc on Dashboard list: https://grafana.com/docs/grafana/latest/panels/visualizations/dashboard-list-panel/

Parameters

- **showHeadings** – The chosen list selection (Starred, Recently viewed, Search) is shown as a heading
- **showSearch** – Display dashboards by search query or tags. Requires you to enter at least one value in Query or Tags
- **showRecent** – Display recently viewed dashboards in alphabetical order
- **showStarred** – Display starred dashboards in alphabetical order
- **maxItems** – Sets the maximum number of items to list per section
- **searchQuery** – Enter the query you want to search by
- **searchTags** – List of tags you want to search by
- **overrides** – To override the base characteristics of certain data

to_json_data()

class grafanalib.core.**DataLink**(*title*, *linkUrl=''*, *isNewTab=False*)

Bases: object

to_json_data()

class grafanalib.core.**DataSourceInput**(*name*, *label*, *pluginId*, *pluginName*, *description=''*)

Bases: object

to_json_data()

class grafanalib.core.**DateColumnStyleType**(*dateFormat='YYYY-MM-DD HH:mm:ss'*)

Bases: object

TYPE = 'date'

to_json_data()

class grafanalib.core.**Discrete**(*dataSource=None*, *targets=NOTHING*, *title=''*, *cacheTimeout=None*, *description=None*, *editable=True*, *error=False*, *height=None*, *gridPos=None*, *hideTimeOverride=False*, *id=None*, *interval=None*, *links=NOTHING*, *maxDataPoints=100*, *minSpan=None*, *repeat=NOTHING*, *span=None*, *thresholds=NOTHING*, *thresholdType='absolute'*, *timeFrom=None*, *timeShift=None*, *transparent=False*, *transformations=NOTHING*, *extraJson=None*, *backgroundColor=RGBA(r=128, g=128, b=128, a=0.1)*, *lineColor=RGBA(r=0, g=0, b=0, a=0.1)*, *metricNameColor='#000000'*, *timeTextColor='#d8d9da'*, *valueTextColor='#000000'*, *decimals=0*, *legendPercentDecimals=0*, *rowHeight=50*, *textSize=24*, *textSizeTime=12*, *units='none'*, *legendSortBy='-ms'*, *highlightOnMouseover=True*, *showLegend=True*, *showLegendPercent=True*, *showLegendNames=True*, *showLegendValues=True*, *showTimeAxis=True*, *use12HourClock=False*, *writeMetricNames=False*, *writeLastValue=True*, *writeAllValues=False*, *showDistinctCount=None*, *showLegendCounts=None*, *showLegendTime=None*, *showTransitionCount=None*, *colorMaps=[]*, *rangeMaps=[]*, *valueMaps=[]*)

Bases: *Panel*

Generates Discrete panel json structure. https://grafana.com/grafana/plugins/natel-discrete-panel/

Parameters

- **colorMaps** – list of DiscreteColorMappingItem, to color values (note these apply **after** value mappings)
- **backgroundColor** – dito
- **lineColor** – Separator line color between rows
- **metricNameColor** – dito
- **timeTextColor** – dito
- **valueTextColor** – dito
- **decimals** – number of decimals to display
- **rowHeight** – dito
- **units** – defines value units
- **legendSortBy** – time (desc: '-ms', asc: 'ms), count (desc: '-count', asc: 'count')
- **highlightOnMouseover** – whether to highlight the state of hovered time falls in.
- **showLegend** – dito
- **showLegendPercent** – whether to show percentage of time spent in each state/value
- **showLegendNames** –
- **showLegendValues** – whether to values in legend
- **legendPercentDecimals** – number of decimals for legend
- **showTimeAxis** – dito
- **use12HourClock** – dito
- **writeMetricNames** – dito
- **writeLastValue** – dito

- **writeAllValues** – whether to show all values
- **showDistinctCount** – whether to show distinct values count
- **showLegendCounts** – whether to show value occurrence count
- **showLegendTime** – whether to show of each state
- **showTransitionCount** – whether to show transition count
- **colorMaps** – list of DiscreteColorMappingItem
- **rangeMaps** – list of RangeMap
- **valueMaps** – list of ValueMap

to_json_data()

class grafanalib.core.**DiscreteColorMappingItem**(*text*, *color=RGBA(r=216, g=200, b=27, a=0.27)*)

Bases: object

Generates json structure for the value mapping item for the StatValueMappings class:

Parameters

- **text** – String to color
- **color** – To color the text with

to_json_data()

class grafanalib.core.**Evaluator**(*type*, *params*)

Bases: object

to_json_data()

class grafanalib.core.**ExternalLink**(*uri*, *title*, *keepTime=False*)

Bases: object

ExternalLink creates a top-level link attached to a dashboard.

Parameters

- **url** – the URL to link to
- **title** – the text of the link
- **keepTime** – if true, the URL params for the dashboard's current time period are appended

to_json_data()

class grafanalib.core.**Gauge**(*minValue=0*, *maxValue=100*, *show=False*, *thresholdLabels=False*, *thresholdMarkers=True*)

Bases: object

to_json_data()

class grafanalib.core.**GaugePanel**(*dataSource=None*, *targets=NOTHING*, *title=''*, *cacheTimeout=None*, *description=None*, *editable=True*, *error=False*, *height=None*, *gridPos=None*, *hideTimeOverride=False*, *id=None*, *interval=None*, *links=NOTHING*, *maxDataPoints=100*, *minSpan=None*, *repeat=NOTHING*, *span=None*, *thresholdType='absolute'*, *timeFrom=None*, *timeShift=None*, *transparent=False*, *transformations=NOTHING*, *extraJson=None*, *allValues=False*, *calc='mean'*, *dataLinks=NOTHING*, *decimals=None*, *format='none'*, *label=None*, *limit=None*, *max=100*, *min=0*, *rangeMaps=NOTHING*, *thresholdLabels=False*, *thresholdMarkers=True*, *thresholds=NOTHING*, *valueMaps=NOTHING*)

Bases: *Panel*

Generates Gauge panel json structure

Parameters

- **allValue** – If All values should be shown or a Calculation
- **calc** – Calculation to perform on metrics
- **dataLinks** – list of data links hooked to datapoints on the graph
- **decimals** – override automatic decimal precision for legend/tooltips
- **format** – defines value units
- **labels** – option to show gauge level labels
- **limit** – limit of number of values to show when not Calculating
- **max** – maximum value of the gauge
- **min** – minimum value of the gauge
- **rangeMaps** – the list of value to text mappings
- **thresholdLabel** – label for gauge. Template Variables: "$__series_namei" "$__field_name" "$__cell_{N} / $__calc"
- **thresholdMarkers** – option to show marker of level on gauge
- **thresholds** – single stat thresholds
- **valueMaps** – the list of value to text mappings

to_json_data()

class grafanalib.core.**Graph**(*dataSource=None*, *targets=NOTHING*, *title=''*, *cacheTimeout=None*, *description=None*, *editable=True*, *height=None*, *gridPos=None*, *hideTimeOverride=False*, *id=None*, *interval=None*, *links=NOTHING*, *maxDataPoints=100*, *minSpan=None*, *repeat=NOTHING*, *span=None*, *thresholdType='absolute'*, *timeFrom=None*, *timeShift=None*, *transparent=False*, *transformations=NOTHING*, *extraJson=None*, *alert=None*, *alertThreshold=True*, *aliasColors=NOTHING*, *align=False*, *alignLevel=0*, *bars=False*, *dataLinks=NOTHING*, *error=False*, *fill=1*, *fillGradient=0*, *grid=NOTHING*, *isNew=True*, *legend=NOTHING*, *lines=True*, *lineWidth=2*, *nullPointMode='connected'*, *percentage=False*, *pointRadius=5*, *points=False*, *renderer='flot'*, *seriesOverrides=NOTHING*, *stack=False*, *steppedLine=False*, *tooltip=NOTHING*, *thresholds=NOTHING*, *xAxis=NOTHING*, *yAxes=NOTHING*)

Bases: *Panel*

Generates Graph panel json structure.

Parameters

- **alert** – List of AlertConditions
- **align** – Select to align left and right Y-axes by value
- **alignLevel** – Available when Align is selected. Value to use for alignment of left and right Y-axes
- **bars** – Display values as a bar chart
- **dataLinks** – List of data links hooked to datapoints on the graph
- **fill** – Area fill, amount of color fill for a series. (default 1, 0 is none)
- **fillGradient** – Degree of gradient on the area fill. (0 is no gradient, 10 is a steep gradient. Default is 0.)
- **lines** – Display values as a line graph
- **points** – Display points for values (default False)
- **pointRadius** – Controls how large the points are
- **stack** – Each series is stacked on top of another
- **percentage** – Available when Stack is selected. Each series is drawn as a percentage of the total of all series
- **thresholds** – List of GraphThresholds - Only valid when alert not defined

auto_ref_ids()

Give unique IDs all the panels without IDs.

Returns a new `Graph` that is the same as this one, except all of the metrics have their `refId` property set. Any panels which had an `refId` property set will keep that property, all others will have auto-generated IDs provided for them.

to_json_data()

class grafanalib.core.**GraphThreshold**(*value*, *colorMode='critical'*, *fill=True*, *line=True*, *op='gt'*, *yaxis='left'*, *fillColor=RGBA(r=245, g=54, b=54, a=0.9)*, *lineColor=RGBA(r=245, g=54, b=54, a=0.9)*)

Bases: `object`

Threshold for for Graph panel

Parameters

- **colorMode** – Color mode of the threshold, value can be *ok*, *warning*, *critical* or *custom*. If *custom* is selcted a lineColor and fillColor should be provided
- **fill** – Display threshold fill, defaults to True
- **line** – Display threshold line, defaults to True
- **value** – When to use this color will be null if index is 0
- **op** – EVAL_LT for less than or EVAL_GT for greater than to indicate what the threshold applies to.
- **yaxis** – Choose left or right for Graph panels

- **fillColor** – Fill color of the threshold, when colorMode = "custom"
- **lineColor** – Line color of the threshold, when colorMode = "custom"

Example:

thresholds = [
GraphThreshold(colorMode="ok", value=10.0), GraphThreshold(colorMode="critical", value=90.0)
]

to_json_data()

grafanalib.core.**GreaterThan**(*value*)

class grafanalib.core.**Grid**(*threshold1=None*, *threshold1Color=NOTHING*, *threshold2=None*, *threshold2Color=NOTHING*)

Bases: object

to_json_data()

class grafanalib.core.**GridPos**(*h*, *w*, *x*, *y*)

Bases: object

GridPos describes the panel size and position in grid coordinates.

Parameters

- **h** – height of the panel, grid height units each represents 30 pixels
- **w** – width of the panel 1-24 (the width of the dashboard is divided into 24 columns)
- **x** – x cordinate of the panel, in same unit as w
- **y** – y cordinate of the panel, in same unit as h

to_json_data()

class grafanalib.core.**Heatmap**(*dataSource=None*, *targets=NOTHING*, *title=''*, *cacheTimeout=None*, *description=None*, *editable=True*, *error=False*, *height=None*, *gridPos=None*, *hideTimeOverride=False*, *id=None*, *interval=None*, *links=NOTHING*, *maxDataPoints=100*, *minSpan=None*, *repeat=NOTHING*, *span=None*, *thresholds=NOTHING*, *thresholdType='absolute'*, *timeFrom=None*, *timeShift=None*, *transparent=False*, *transformations=NOTHING*, *extraJson=None*, *legend={'show': False}*, *tooltip=NOTHING*, *cards={'cardPadding': None, 'cardRound': None}*, *color=NOTHING*, *dataFormat='timeseries'*, *hideZeroBuckets=False*, *highlightCards=True*, *options=None*, *xAxis=NOTHING*, *xBucketNumber=None*, *xBucketSize=None*, *yAxis=NOTHING*, *yBucketBound=None*, *yBucketNumber=None*, *yBucketSize=None*, *reverseYBuckets=False*)

Bases: *Panel*

Generates Heatmap panel json structure (https://grafana.com/docs/grafana/latest/features/panels/heatmap/)

Parameters

- **heatmap** – dict
- **cards** – A heatmap card object: keys "cardPadding", "cardRound"
- **color** – Heatmap color object
- **dataFormat** – 'timeseries' or 'tsbuckets'

- **yBucketBound** – 'auto', 'upper', 'middle', 'lower'
- **reverseYBuckets** – boolean
- **xBucketSize** – Size
- **xBucketNumber** – Number
- **yBucketSize** – Size
- **yBucketNumber** – Number
- **highlightCards** – boolean
- **hideZeroBuckets** – boolean
- **transparent** – defines if the panel should be transparent

heatmap = {}

to_json_data()

class grafanalib.core.**HeatmapColor**(*cardColor='#b4ff00'*, *colorScale='sqrt'*, *colorScheme='interpolateOranges'*, *exponent=0.5*, *mode='spectrum'*, *max=None*, *min=None*)

Bases: object

A Color object for heatmaps

Parameters

- **cardColor** – color
- **colorScale** – scale
- **colorScheme** – scheme
- **exponent** – exponent
- **max** – max
- **min** – min
- **mode** – mode

to_json_data()

class grafanalib.core.**HiddenColumnStyleType**

Bases: object

TYPE = 'hidden'

to_json_data()

class grafanalib.core.**Histogram**(*dataSource=None*, *targets=NOTHING*, *title=''*, *cacheTimeout=None*, *description=None*, *editable=True*, *error=False*, *height=None*, *gridPos=None*, *hideTimeOverride=False*, *id=None*, *interval=None*, *links=NOTHING*, *maxDataPoints=100*, *minSpan=None*, *repeat=NOTHING*, *span=None*, *thresholds=NOTHING*, *thresholdType='absolute'*, *timeFrom=None*, *timeShift=None*, *transparent=False*, *transformations=NOTHING*, *extraJson=None*, *bucketOffset=0*, *bucketSize=0*, *colorMode='thresholds'*, *combine=False*, *fillOpacity=80*, *legendDisplayMode='list'*, *legendPlacement='bottom'*, *lineWidth=0*, *mappings=NOTHING*, *overrides=NOTHING*)

Bases: *Panel*

Generates Histogram panel json structure Grafana docs on Histogram panel: https://grafana.com/docs/grafana/latest/visualizations/histogram/#

Parameters

- **bucketOffset** – Bucket offset for none-zero-based buckets
- **bucketSize** – Bucket size, default Auto
- **colorMode** – Default thresholds
- **combine** – Combine all series into a single histogram
- **fillOpacity** – Controls the opacity of state regions, default 0.9
- **legendDisplayMode** – refine how the legend appears, list, table or hidden
- **legendPlacement** – bottom or top
- **lineWidth** – Controls line width of state regions
- **mappings** – To assign colors to boolean or string values, use Value mappings
- **overrides** – To override the base characteristics of certain data

to_json_data()

class grafanalib.core.**Legend**(*avg=False*, *current=False*, *max=False*, *min=False*, *show=True*, *total=False*, *values=None*, *alignAsTable=False*, *hideEmpty=False*, *hideZero=False*, *rightSide=False*, *sideWidth=None*, *sort=None*, *sortDesc=False*)

Bases: object

to_json_data()

class grafanalib.core.**Logs**(*dataSource=None*, *targets=NOTHING*, *title=''*, *cacheTimeout=None*, *description=None*, *editable=True*, *error=False*, *height=None*, *gridPos=None*, *hideTimeOverride=False*, *id=None*, *interval=None*, *links=NOTHING*, *maxDataPoints=100*, *minSpan=None*, *repeat=NOTHING*, *span=None*, *thresholds=NOTHING*, *thresholdType='absolute'*, *timeFrom=None*, *timeShift=None*, *transparent=False*, *transformations=NOTHING*, *extraJson=None*, *showLabels=False*, *showCommonLabels=False*, *showTime=False*, *wrapLogMessages=False*, *sortOrder='Descending'*, *dedupStrategy='none'*, *enableLogDetails=False*, *overrides=NOTHING*, *prettifyLogMessage=False*)

Bases: *Panel*

Generates Logs panel json structure Grafana doc on Logs panel: https://grafana.com/docs/grafana/latest/panels/visualizations/logs-panel/

Parameters

- **showLabels** – Show or hide the unique labels column, which shows only non-common labels
- **showCommonLabels** – Show or hide the common labels.
- **showTime** – Show or hide the log timestamp column
- **wrapLogMessages** – Toggle line wrapping

- **sortOrder** – Display results in 'Descending' or 'Ascending' time order. The default is Descending, showing the newest logs first.
- **dedupStrategy** – One of none, exact, numbers, signature. Default is none
- **enableLogDetails** – Set this to True to see the log details view for each log row.
- **overrides** – To override the base characteristics of certain data
- **prettifyLogMessage** – Set this to true to pretty print all JSON logs. This setting does not affect logs in any format other than JSON.

to_json_data()

grafanalib.core.**LowerThan**(*value*)

class grafanalib.core.**Mapping**(*name*, *value*)

Bases: object

to_json_data()

class grafanalib.core.**News**(*dataSource=None*, *targets=NOTHING*, *title=''*, *cacheTimeout=None*, *description=None*, *editable=True*, *error=False*, *height=None*, *gridPos=None*, *hideTimeOverride=False*, *id=None*, *interval=None*, *links=NOTHING*, *maxDataPoints=100*, *minSpan=None*, *repeat=NOTHING*, *span=None*, *thresholds=NOTHING*, *thresholdType='absolute'*, *timeFrom=None*, *timeShift=None*, *transparent=False*, *transformations=NOTHING*, *extraJson=None*, *feedUrl=''*, *showImage=True*, *useProxy=False*)

Bases: *Panel*

Generates News panel json structure

Parameters

- **feedUrl** – URL to query, only RSS feed formats are supported (not Atom).
- **showImage** – Controls if the news item social (og:image) image is shown above text content
- **useProxy** – If the feed is unable to connect, consider a CORS proxy

to_json_data()

grafanalib.core.**NoValue**()

class grafanalib.core.**Notification**(*uid*)

Bases: object

to_json_data()

class grafanalib.core.**NumberColumnStyleType**(*colorMode=None*, *colors=NOTHING*, *thresholds=NOTHING*, *decimals=2*, *unit='short'*)

Bases: object

TYPE = 'number'

to_json_data()

grafanalib.core.**OutsideRange**(*from_value*, *to_value*)

class grafanalib.core.**Panel**(*dataSource=None*, *targets=NOTHING*, *title=''*, *cacheTimeout=None*, *description=None*, *editable=True*, *error=False*, *height=None*, *gridPos=None*, *hideTimeOverride=False*, *id=None*, *interval=None*, *links=NOTHING*, *maxDataPoints=100*, *minSpan=None*, *repeat=NOTHING*, *span=None*, *thresholds=NOTHING*, *thresholdType='absolute'*, *timeFrom=None*, *timeShift=None*, *transparent=False*, *transformations=NOTHING*, *extraJson=None*)

Bases: `object`

Generic panel for shared defaults

Parameters

- **cacheTimeout** – metric query result cache ttl
- **dataSource** – Grafana datasource name
- **description** – optional panel description
- **editable** – defines if panel is editable via web interfaces
- **height** – defines panel height
- **hideTimeOverride** – hides time overrides
- **id** – panel id
- **interval** – defines time interval between metric queries
- **links** – additional web links
- **maxDataPoints** – maximum metric query results, that will be used for rendering
- **minSpan** – minimum span number
- **repeat** – Template's name to repeat Graph on
- **span** – defines the number of spans that will be used for panel
- **targets** – list of metric requests for chosen datasource
- **thresholds** – single stat thresholds
- **thresholdType** – type of threshold, absolute or percentage
- **timeFrom** – time range that Override relative time
- **title** – of the panel
- **transparent** – defines if panel should be transparent
- **transformations** – defines transformations applied to the table
- **extraJson** – raw JSON additions or overrides added to the JSON output of this panel, can be used for using unsupported features

panel_json(*overrides*)

class grafanalib.core.**Percent**(*num=100*)

Bases: `object`

to_json_data()

class grafanalib.core.**PieChart**(*dataSource=None*, *targets=NOTHING*, *title=''*, *cacheTimeout=None*, *description=None*, *editable=True*, *error=False*, *height=None*, *gridPos=None*, *hideTimeOverride=False*, *id=None*, *interval=None*, *links=NOTHING*, *maxDataPoints=100*, *minSpan=None*, *repeat=NOTHING*, *span=None*, *thresholdType='absolute'*, *timeFrom=None*, *timeShift=None*, *transparent=False*, *transformations=NOTHING*, *extraJson=None*, *aliasColors=NOTHING*, *format='none'*, *legendType='Right side'*, *overrides=NOTHING*, *pieType='pie'*, *percentageDecimals=0*, *showLegend=True*, *showLegendValues=True*, *showLegendPercentage=False*, *thresholds=''*)

Bases: *Panel*

Generates Pie Chart panel json structure

This panel was deprecated in Grafana 8.0, please use PieChartv2 instead

Grafana doc on Pie Chart: https://grafana.com/grafana/plugins/grafana-piechart-panel

Parameters

- **aliasColors** – dictionary of color overrides
- **format** – defines value units
- **legendType** – defines where the legend position
- **overrides** – To override the base characteristics of certain data
- **pieType** – defines the shape of the pie chart (pie or donut)
- **percentageDecimals** – Number of decimal places to show if percentages shown in legned
- **showLegend** – defines if the legend should be shown
- **showLegendValues** – defines if the legend should show values
- **showLegendPercentage** – Show percentages in the legend
- **thresholds** – defines thresholds

to_json_data()

class grafanalib.core.**PieChartv2**(*dataSource=None*, *targets=NOTHING*, *title=''*, *cacheTimeout=None*, *description=None*, *editable=True*, *error=False*, *height=None*, *gridPos=None*, *hideTimeOverride=False*, *id=None*, *interval=None*, *links=NOTHING*, *maxDataPoints=100*, *minSpan=None*, *repeat=NOTHING*, *span=None*, *thresholds=NOTHING*, *thresholdType='absolute'*, *timeFrom=None*, *timeShift=None*, *transparent=False*, *transformations=NOTHING*, *extraJson=None*, *custom={}*, *colorMode='palette-classic'*, *legendDisplayMode='list'*, *legendPlacement='bottom'*, *legendValues=[]*, *mappings=NOTHING*, *overrides=[]*, *pieType='pie'*, *reduceOptionsCalcs=['lastNotNull']*, *reduceOptionsFields=''*, *reduceOptionsValues=False*, *tooltipMode='single'*, *unit=''*)

Bases: *Panel*

Generates Pie Chart panel json structure Grafana docs on Pie Chart: https://grafana.com/docs/grafana/latest/visualizations/pie-chart-panel/

Parameters

- **custom** – Custom overides

- **colorMode** – Color mode palette-classic (Default),
- **legendDisplayMode** – Display mode of legend: list, table or hidden
- **legendPlacement** – Location of the legend in the panel: bottom or right
- **legendValues** – List of value to be shown in legend eg. [‘value’, ‘percent’]
- **mappings** – To assign colors to boolean or string values, use Value mappings
- **overrides** – Overrides
- **pieType** – Pie chart type pie (Default), donut
- **reduceOptionsCalcs** – Reducer function / calculation
- **reduceOptionsFields** – Fields that should be included in the panel
- **reduceOptionsValues** – Calculate a single value per column or series or show each row
- **tooltipMode** – Tooltip mode single (Default), multi, none
- **unit** – units

to_json_data()

class grafanalib.core.**Pixels**(*num*)

Bases: object

to_json_data()

class grafanalib.core.**RGB**(*r*, *g*, *b*)

Bases: object

to_json_data()

class grafanalib.core.**RGBA**(*r*, *g*, *b*, *a*)

Bases: object

to_json_data()

class grafanalib.core.**RangeMap**(*start*, *end*, *text*)

Bases: object

to_json_data()

class grafanalib.core.**Repeat**(*direction=None*, *variable=None*, *maxPerRow=None*)

Bases: object

Panel repetition settings.

Parameters

- **direction** – The direction into which to repeat (‘h’ or ‘v’)
- **variable** – The name of the variable over whose values to repeat
- **maxPerRow** – The maximum number of panels per row in horizontal repetition

to_json_data()

class grafanalib.core.**Row**(*panels=NOTHING*, *collapse=False*, *editable=True*, *height=NOTHING*, *showTitle=None*, *title=None*, *repeat=None*)

Bases: `object`

Legacy support for old row, when not used with gridpos

to_json_data()

class grafanalib.core.**RowPanel**(*dataSource=None*, *targets=NOTHING*, *title=''*, *cacheTimeout=None*, *description=None*, *editable=True*, *error=False*, *height=None*, *gridPos=None*, *hideTimeOverride=False*, *id=None*, *interval=None*, *links=NOTHING*, *maxDataPoints=100*, *minSpan=None*, *repeat=NOTHING*, *span=None*, *thresholds=NOTHING*, *thresholdType='absolute'*, *timeFrom=None*, *timeShift=None*, *transparent=False*, *transformations=NOTHING*, *extraJson=None*, *panels=NOTHING*, *collapsed=False*)

Bases: *Panel*

Generates Row panel json structure.

Parameters

- **title** – title of the panel
- **collapsed** – set True if row should be collapsed
- **panels** – list of panels in the row, only to be used when collapsed=True

to_json_data()

class grafanalib.core.**SeriesOverride**(*alias*, *bars=False*, *lines=True*, *yaxis=1*, *fill=1*, *zindex=0*, *dashes=False*, *dashLength=None*, *spaceLength=None*, *color=None*, *fillBelowTo=None*)

Bases: `object`

To override properties of e.g. Graphs.

Parameters

- **alias** – Name of the metric to apply to
- **bars** – Whether to show data point bars
- **lines** – Whether to keep graph lines
- **yaxis** – Whether to move axis of the metric to the right (=2) or not (=1)
- **fill** – Fill strength (0…10)
- **color** – Whether to change color to
- **fillBelowTo** – Alias of the other metric to fill below
- **zindex** – Move things to front or background (-3…3)
- **dashed** – Whether to dash the line
- **dashLength** – Length of dashes (1..20)
- **spaceLength** – Length of spaces betwee dashed
- **zindex** – Move things to front or background

to_json_data()

class grafanalib.core.**SingleStat**(*dataSource=None*, *targets=NOTHING*, *title=''*, *description=None*, *editable=True*, *error=False*, *height=None*, *gridPos=None*, *hideTimeOverride=False*, *id=None*, *interval=None*, *links=NOTHING*, *maxDataPoints=100*, *repeat=NOTHING*, *span=None*, *thresholdType='absolute'*, *timeFrom=None*, *timeShift=None*, *transparent=False*, *transformations=NOTHING*, *extraJson=None*, *cacheTimeout=None*, *colors=NOTHING*, *colorBackground=False*, *colorValue=False*, *decimals=None*, *format='none'*, *gauge=NOTHING*, *mappingType=1*, *mappingTypes=NOTHING*, *minSpan=None*, *nullText=None*, *nullPointMode='connected'*, *postfix=''*, *postfixFontSize='50%'*, *prefix=''*, *prefixFontSize='50%'*, *rangeMaps=NOTHING*, *sparkline=NOTHING*, *thresholds=''*, *valueFontSize='80%'*, *valueName='avg'*, *valueMaps=NOTHING*)

Bases: *Panel*

Generates Single Stat panel json structure

This panel was deprecated in Grafana 7.0, please use Stat instead

Grafana doc on singlestat: https://grafana.com/docs/grafana/latest/features/panels/singlestat/

Parameters

- **cacheTimeout** – metric query result cache ttl
- **colors** – the list of colors that can be used for coloring panel value or background. Additional info on coloring in docs: https://grafana.com/docs/grafana/latest/features/panels/singlestat/#coloring
- **colorBackground** – defines if grafana will color panel background
- **colorValue** – defines if grafana will color panel value
- **decimals** – override automatic decimal precision for legend/tooltips
- **format** – defines value units
- **gauge** – draws and additional speedometer-like gauge based
- **mappingType** – defines panel mapping type. Additional info can be found in docs: https://grafana.com/docs/grafana/latest/features/panels/singlestat/#value-to-text-mapping
- **mappingTypes** – the list of available mapping types for panel
- **nullText** – defines what to show if metric query result is undefined
- **nullPointMode** – defines how to render undefined values
- **postfix** – defines postfix that will be attached to value
- **postfixFontSize** – defines postfix font size
- **prefix** – defines prefix that will be attached to value
- **prefixFontSize** – defines prefix font size
- **rangeMaps** – the list of value to text mappings
- **sparkline** – defines if grafana should draw an additional sparkline. Sparkline grafana documentation: https://grafana.com/docs/grafana/latest/features/panels/singlestat/#spark-lines
- **thresholds** – single stat thresholds
- **valueFontSize** – defines value font size

- **valueName** – defines value type. possible values are: min, max, avg, current, total, name, first, delta, range
- **valueMaps** – the list of value to text mappings

to_json_data()

class grafanalib.core.**SparkLine**(*fillColor=NOTHING*, *full=False*, *lineColor=NOTHING*, *show=False*)

Bases: object

to_json_data()

class grafanalib.core.**SqlTarget**(*expr=''*, *format='time_series'*, *hide=False*, *legendFormat=''*, *interval=''*, *intervalFactor=2*, *metric=''*, *refId=''*, *step=10*, *target=''*, *instant=False*, *datasource=None*, *rawSql=''*, *rawQuery=True*)

Bases: *Target*

Metric target to support SQL queries

to_json_data()

Override the Target to_json_data to add additional fields. rawSql: this will contain the actual SQL queries rawQuery: this is set to True by default as in case of False

the rawSql would be unused

class grafanalib.core.**Stat**(*dataSource=None*, *targets=NOTHING*, *title=''*, *cacheTimeout=None*, *description=None*, *editable=True*, *error=False*, *height=None*, *gridPos=None*, *hideTimeOverride=False*, *id=None*, *interval=None*, *links=NOTHING*, *maxDataPoints=100*, *minSpan=None*, *repeat=NOTHING*, *span=None*, *thresholdType='absolute'*, *timeFrom=None*, *timeShift=None*, *transparent=False*, *transformations=NOTHING*, *extraJson=None*, *alignment='auto'*, *colorMode='value'*, *decimals=None*, *format='none'*, *graphMode='area'*, *mappings=NOTHING*, *noValue='none'*, *orientation='auto'*, *overrides=NOTHING*, *reduceCalc: str = 'mean'*, *fields=''*, *textMode='auto'*, *thresholds=''*)

Bases: *Panel*

Generates Stat panel json structure

Grafana doc on stat: https://grafana.com/docs/grafana/latest/panels/visualizations/stat-panel/

Parameters

- **alignment** – defines value & title positioning: keys 'auto' 'centre'
- **colorMode** – defines if Grafana will color panel background: keys "value" "background"
- **decimals** – number of decimals to display
- **format** – defines value units
- **graphMode** – defines if Grafana will draw graph: keys 'area' 'none'
- **noValue** – define the default value if no value is found
- **mappings** – the list of values to text mappings This should be a list of StatMapping objects https://grafana.com/docs/grafana/latest/panels/field-configuration-options/#value-mapping
- **orientation** – Stacking direction in case of multiple series or fields: keys 'auto' 'horizontal' 'vertical'
- **overrides** – To override the base characteristics of certain timeseries data

- **reduceCalc** – algorithm for reduction to a single value: keys 'mean' 'lastNotNull' 'last' 'first' 'firstNotNull' 'min' 'max' 'sum' 'total'
- **fields** – should be included in the panel
- **textMode** – define Grafana will show name or value: keys: 'auto' 'name' 'none' 'value' 'value_and_name'
- **thresholds** – single stat thresholds

to_json_data()

class grafanalib.core.**StatMapping**(*text*, *mapValue=''*, *startValue=''*, *endValue=''*, *id=None*)

Bases: object

Deprecated Grafana v8 Generates json structure for the value mapping for the Stat panel:

Parameters

- **text** – Sting that will replace input value
- **value** – Value to be replaced
- **startValue** – When using a range, the start value of the range
- **endValue** – When using a range, the end value of the range
- **id** – panel id

to_json_data()

class grafanalib.core.**StatRangeMapping**(*text*, *startValue=''*, *endValue=''*, *id=None*)

Bases: object

Deprecated Grafana v8 Generates json structure for the range mappings for the StatPanel:

Parameters

- **text** – Sting that will replace input value
- **startValue** – When using a range, the start value of the range
- **endValue** – When using a range, the end value of the range
- **id** – panel id

to_json_data()

class grafanalib.core.**StatRangeMappings**(*text*, *startValue=0*, *endValue=0*, *color=''*, *index=None*)

Bases: object

Generates json structure for the range mappings for the StatPanel:

Parameters

- **text** – Sting that will replace input value
- **startValue** – When using a range, the start value of the range
- **endValue** – When using a range, the end value of the range
- **color** – How to color the text if mapping occurs
- **index** – index

to_json_data()

class grafanalib.core.**StatValueMapping**(*text*, *mapValue=''*, *id=None*)

Bases: object

Deprecated Grafana v8 Generates json structure for the value mappings for the StatPanel:

Parameters

- **text** – Sting that will replace input value
- **mapValue** – Value to be replaced
- **id** – panel id

to_json_data()

class grafanalib.core.**StatValueMappingItem**(*text*, *mapValue=''*, *color=''*, *index=None*)

Bases: object

Generates json structure for the value mapping item for the StatValueMappings class:

Parameters

- **text** – String that will replace input value
- **mapValue** – Value to be replaced
- **color** – How to color the text if mapping occurs
- **index** – index

to_json_data()

class grafanalib.core.**StatValueMappings**(**mappings:* StatValueMappingItem)

Bases: object

Generates json structure for the value mappings for the StatPanel:

Parameters
mappingItems – List of StatValueMappingItem objects

mappings=[

core.StatValueMappings(
core.StatValueMappingItem('Offline', '0', 'red'), # Value must a string
core.StatValueMappingItem('Online', '1', 'green')

),

],

to_json_data()

class grafanalib.core.**StateTimeline**(*dataSource=None*, *targets=NOTHING*, *title=''*, *cacheTimeout=None*, *description=None*, *editable=True*, *error=False*, *height=None*, *gridPos=None*, *hideTimeOverride=False*, *id=None*, *interval=None*, *links=NOTHING*, *maxDataPoints=100*, *minSpan=None*, *repeat=NOTHING*, *span=None*, *thresholds=NOTHING*, *thresholdType='absolute'*, *timeFrom=None*, *timeShift=None*, *transparent=False*, *transformations=NOTHING*, *extraJson=None*, *alignValue='left'*, *colorMode='thresholds'*, *fillOpacity=70*, *legendDisplayMode='list'*, *legendPlacement='bottom'*, *lineWidth=0*, *mappings=NOTHING*, *overrides=NOTHING*, *mergeValues=True*, *rowHeight=0.9*, *showValue='auto'*, *tooltipMode='single'*)

Bases: *Panel*

Generates State Timeline panel json structure Grafana docs on State Timeline panel: https://grafana.com/docs/grafana/latest/visualizations/state-timeline/

Parameters

- **alignValue** – Controls value alignment inside state regions, default left
- **colorMode** – Default thresholds
- **fillOpacity** – Controls the opacity of state regions, default 0.9
- **legendDisplayMode** – refine how the legend appears, list, table or hidden
- **legendPlacement** – bottom or top
- **lineWidth** – Controls line width of state regions
- **mappings** – To assign colors to boolean or string values, use Value mappings
- **overrides** – To override the base characteristics of certain data
- **mergeValues** – Controls whether Grafana merges identical values if they are next to each other, default True
- **rowHeight** – Controls how much space between rows there are. 1 = no space = 0.5 = 50% space
- **showValue** – Controls whether values are rendered inside the state regions. Auto will render values if there is sufficient space.
- **tooltipMode** – Default single

to_json_data()

class grafanalib.core.**Statusmap**(*dataSource=None*, *targets=NOTHING*, *title=''*, *cacheTimeout=None*, *description=None*, *editable=True*, *error=False*, *height=None*, *gridPos=None*, *hideTimeOverride=False*, *id=None*, *interval=None*, *links=NOTHING*, *maxDataPoints=100*, *minSpan=None*, *repeat=NOTHING*, *span=None*, *thresholds=NOTHING*, *thresholdType='absolute'*, *timeFrom=None*, *timeShift=None*, *transparent=False*, *transformations=NOTHING*, *extraJson=None*, *alert=None*, *cards={'cardHSpacing': 2, 'cardMinWidth': 5, 'cardRound': None, 'cardVSpacing': 2}*, *color=NOTHING*, *isNew=True*, *legend=NOTHING*, *nullPointMode='null as zero'*, *tooltip=NOTHING*, *xAxis=NOTHING*, *yAxis=NOTHING*)

Bases: *Panel*

Generates json structure for the flant-statusmap-panel visualisation plugin (https://grafana.com/grafana/plugins/flant-statusmap-panel/).

Parameters

- **alert** – Alert
- **cards** – A statusmap card object: keys 'cardRound', 'cardMinWidth', 'cardHSpacing', 'cardVSpacing'
- **color** – A StatusmapColor object
- **isNew** – isNew
- **legend** – Legend object

- **nullPointMode** – null
- **tooltip** – Tooltip object
- **xAxis** – XAxis object
- **yAxis** – YAxis object

to_json_data()

class grafanalib.core.**StatusmapColor**(*cardColor='#b4ff00'*, *colorScale='sqrt'*, *colorScheme='GnYlRd'*, *exponent=0.5*, *mode='spectrum'*, *thresholds=[]*, *max=None*, *min=None*)

Bases: `object`

A Color object for Statusmaps

Parameters

- **cardColor** – colour
- **colorScale** – scale
- **colorScheme** – scheme
- **exponent** – exponent
- **max** – max
- **min** – min
- **mode** – mode
- **thresholds** – threshold

to_json_data()

class grafanalib.core.**StringColumnStyleType**(*decimals=2*, *colorMode=None*, *colors=NOTHING*, *thresholds=NOTHING*, *preserveFormat=False*, *sanitize=False*, *unit='short'*, *mappingType=1*, *valueMaps=NOTHING*, *rangeMaps=NOTHING*)

Bases: `object`

TYPE = 'string'

to_json_data()

class grafanalib.core.**Svg**(*dataSource=None*, *targets=NOTHING*, *title=''*, *cacheTimeout=None*, *description=None*, *editable=True*, *error=False*, *gridPos=None*, *hideTimeOverride=False*, *id=None*, *interval=None*, *links=NOTHING*, *maxDataPoints=100*, *minSpan=None*, *repeat=NOTHING*, *span=None*, *thresholds=NOTHING*, *thresholdType='absolute'*, *timeFrom=None*, *timeShift=None*, *transparent=False*, *transformations=NOTHING*, *extraJson=None*, *format='none'*, *jsCodeFilePath=''*, *jsCodeInitFilePath=''*, *height=None*, *svgFilePath=''*)

Bases: *Panel*

Generates SVG panel json structure Grafana doc on SVG: https://grafana.com/grafana/plugins/marcuscalidus-svg-panel

Parameters

- **format** – defines value units

- **jsCodeFilePath** – path to javascript file to be run on dashboard refresh
- **jsCodeInitFilePath** – path to javascript file to be run after the first initialization of the SVG
- **reduceCalc** – algorithm for reduction to a single value, keys 'mean' 'lastNotNull' 'last' 'first' 'firstNotNull' 'min' 'max' 'sum' 'total'
- **svgFilePath** – path to SVG image file to be displayed

static read_file(*file_path*)

to_json_data()

class grafanalib.core.**Table**(*dataSource=None*, *targets=NOTHING*, *title=''*, *cacheTimeout=None*, *description=None*, *editable=True*, *error=False*, *height=None*, *gridPos=None*, *hideTimeOverride=False*, *id=None*, *interval=None*, *links=NOTHING*, *maxDataPoints=100*, *minSpan=None*, *repeat=NOTHING*, *thresholds=NOTHING*, *thresholdType='absolute'*, *timeFrom=None*, *timeShift=None*, *transparent=False*, *transformations=NOTHING*, *extraJson=None*, *align='auto'*, *colorMode='thresholds'*, *columns=NOTHING*, *displayMode='auto'*, *fontSize='100%'*, *filterable=False*, *mappings=NOTHING*, *overrides=NOTHING*, *showHeader=True*, *span=6*)

Bases: *Panel*

Generates Table panel json structure

Now supports Grafana v8+ Grafana doc on table: https://grafana.com/docs/grafana/latest/visualizations/table/

Parameters

- **align** – Align cell contents; auto (default), left, center, right
- **colorMode** – Default thresholds
- **columns** – Table columns for Aggregations view
- **displayMode** – By default, Grafana automatically chooses display settings, you can choose; color-text, color-background, color-background-solid, gradient-gauge, lcd-gauge, basic, json-view
- **fontSize** – Defines value font size
- **filterable** – Allow user to filter columns, default False
- **mappings** – To assign colors to boolean or string values, use Value mappings
- **overrides** – To override the base characteristics of certain data
- **showHeader** – Show the table header

to_json_data()

classmethod with_styled_columns(*columns*, *styles=None*, ***kwargs*)

Styled columns is not support in Grafana v8 Table

class grafanalib.core.**Target**(*expr=''*, *format='time_series'*, *hide=False*, *legendFormat=''*, *interval=''*, *intervalFactor=2*, *metric=''*, *refId=''*, *step=10*, *target=''*, *instant=False*, *datasource=None*)

Bases: `object`

Metric to show.

Parameters

target – Graphite way to select data

to_json_data()

class grafanalib.core.**Template**(*name*, *query*, *default=None*, *dataSource=None*, *label=None*, *allValue=None*, *includeAll=False*, *multi=False*, *options=NOTHING*, *regex=None*, *useTags=False*, *tagsQuery=None*, *tagValuesQuery=None*, *refresh=1*, *type='query'*, *hide=0*, *sort=1*, *auto=False*, *autoCount=30*, *autoMin='10s'*)

Bases: object

Template create a new 'variable' for the dashboard, defines the variable name, human name, query to fetch the values and the default value.

Parameters

- **default** – the default value for the variable
- **dataSource** – where to fetch the values for the variable from
- **label** – the variable's human label
- **name** – the variable's name
- **query** – the query users to fetch the valid values of the variable
- **refresh** – Controls when to update values in the dropdown
- **allValue** – specify a custom all value with regex, globs or lucene syntax.
- **includeAll** – Add a special All option whose value includes all options.
- **regex** – Regex to filter or capture specific parts of the names return by your data source query.
- **multi** – If enabled, the variable will support the selection of multiple options at the same time.
- **type** – The template type, can be one of: query (default), interval, datasource, custom, constant, adhoc.
- **hide** – Hide this variable in the dashboard, can be one of: SHOW (default), HIDE_LABEL, HIDE_VARIABLE
- **auto** – Interval will be dynamically calculated by dividing time range by the count specified in auto_count.
- **autoCount** – Number of intervals for dividing the time range.
- **autoMin** – Smallest interval for auto interval generator.

to_json_data()

class grafanalib.core.**Templating**(*list=NOTHING*)

Bases: object

to_json_data()

class grafanalib.core.**Text**(*dataSource=None*, *targets=NOTHING*, *title=''*, *cacheTimeout=None*, *description=None*, *editable=True*, *height=None*, *gridPos=None*, *hideTimeOverride=False*, *id=None*, *interval=None*, *links=NOTHING*, *maxDataPoints=100*, *minSpan=None*, *repeat=NOTHING*, *span=None*, *thresholds=NOTHING*, *thresholdType='absolute'*, *timeFrom=None*, *timeShift=None*, *transparent=False*, *transformations=NOTHING*, *extraJson=None*, *content=''*, *error=False*, *mode='markdown'*)

Bases: *Panel*

Generates a Text panel.

to_json_data()

class grafanalib.core.**Threshold**(*color*, *index*, *value*, *line=True*, *op='gt'*, *yaxis='left'*)

Bases: object

Threshold for for panels

Parameters

- **color** – Color of threshold
- **index** – Index of color in panel
- **line** – Display Threshold line, defaults to True
- **value** – When to use this color will be null if index is 0
- **op** – EVAL_LT for less than or EVAL_GT for greater than to indicate what the threshold applies to.
- **yaxis** – Choose left or right for panels

Care must be taken in the order in which the Threshold objects are specified, Grafana expects the value to increase.

Example::

thresholds = [
Threshold('green', 0, 0.0), Threshold('red', 1, 80.0)]

to_json_data()

class grafanalib.core.**Time**(*start*, *end*)

Bases: object

to_json_data()

class grafanalib.core.**TimePicker**(*refreshIntervals*, *timeOptions*, *hidden=False*)

Bases: object

Time Picker

Parameters

- **refreshIntervals** – dashboard auto-refresh interval options
- **timeOptions** – dashboard time range options
- **hidden** – hide the time picker from dashboard

to_json_data()

class grafanalib.core.**TimeRange**(*from_time*, *to_time*)

Bases: `object`

A time range for an alert condition.

A condition has to hold for this length of time before triggering.

Parameters

- **from_time** (*str*) – Either a number + unit (s: second, m: minute, h: hour, etc) e.g. `"5m"` for 5 minutes, or `"now"`.
- **to_time** (*str*) – Either a number + unit (s: second, m: minute, h: hour, etc) e.g. `"5m"` for 5 minutes, or `"now"`.

to_json_data()

class grafanalib.core.**TimeSeries**(*dataSource=None*, *targets=NOTHING*, *title=''*, *cacheTimeout=None*, *description=None*, *editable=True*, *error=False*, *height=None*, *gridPos=None*, *hideTimeOverride=False*, *id=None*, *interval=None*, *links=NOTHING*, *maxDataPoints=100*, *minSpan=None*, *repeat=NOTHING*, *span=None*, *thresholds=NOTHING*, *thresholdType='absolute'*, *timeFrom=None*, *timeShift=None*, *transparent=False*, *transformations=NOTHING*, *extraJson=None*, *axisPlacement='auto'*, *axisLabel=''*, *barAlignment=0*, *colorMode='palette-classic'*, *drawStyle='line'*, *fillOpacity=0*, *gradientMode='none'*, *legendDisplayMode='list'*, *legendPlacement='bottom'*, *legendCalcs=[]*, *lineInterpolation='linear'*, *lineWidth=1*, *mappings=NOTHING*, *overrides=NOTHING*, *pointSize=5*, *scaleDistributionType='linear'*, *scaleDistributionLog=2*, *spanNulls=False*, *showPoints='auto'*, *stacking={}*, *tooltipMode='single'*, *unit=''*, *thresholdsStyleMode='off'*)

Bases: *Panel*

Generates Time Series panel json structure added in Grafana v8

Grafana doc on time series: https://grafana.com/docs/grafana/latest/panels/visualizations/time-series/

Parameters

- **axisPlacement** – auto(Default), left. right, hidden
- **axisLabel** – axis label string
- **barAlignment** – bar alignment -1 (left), 0 (centre, default), 1
- **colorMode** – Color mode palette-classic (Default),
- **drawStyle** – how to display your time series data line (Default), bars, points
- **fillOpacity** – fillOpacity
- **gradientMode** – gradientMode
- **legendDisplayMode** – refine how the legend appears in your visualization list (Default), table, hidden
- **legendPlacement** – bottom (Default), right
- **legendCalcs** – which calculations should be displayed in the legend. Defaults to an empty list. Possible values are: allIsNull, allIsZero, changeCount, count, delta, diff, diffperc, distinctCount, firstNotNull, max, mean, min, logmin, range, step, total. For more information see

- **lineInterpolation** – line interpolation linear (Default), smooth, stepBefore, stepAfter
- **lineWidth** – line width, default 1
- **mappings** – To assign colors to boolean or string values, use Value mappings
- **overrides** – To override the base characteristics of certain timeseries data
- **pointSize** – point size, default 5
- **scaleDistributionType** – axis scale linear or log
- **scaleDistributionLog** – Base of if logarithmic scale type set, default 2
- **spanNulls** – connect null values, default False
- **showPoints** – show points auto (Default), always, never
- **stacking** – dict to enable stacking, {"mode": "normal", "group": "A"}
- **thresholds** – single stat thresholds
- **tooltipMode** – When you hover your cursor over the visualization, Grafana can display tooltips single (Default), multi, none
- **unit** – units
- **thresholdsStyleMode** – thresholds style mode off (Default), area, line, line+area

to_json_data()

class grafanalib.core.**Tooltip**(*msResolution=True*, *shared=True*, *sort=0*, *valueType='cumulative'*)

Bases: object

to_json_data()

class grafanalib.core.**ValueMap**(*text*, *value*, *op='='*)

Bases: object

Generates json structure for a value mapping item.

Parameters

- **op** – comparison operator
- **value** – value to map to text
- **text** – text to map the value to

to_json_data()

grafanalib.core.**WithinRange**(*from_value*, *to_value*)

class grafanalib.core.**Worldmap**(*dataSource=None*, *targets=NOTHING*, *title=''*, *cacheTimeout=None*, *description=None*, *editable=True*, *error=False*, *height=None*, *gridPos=None*, *hideTimeOverride=False*, *id=None*, *interval=None*, *links=NOTHING*, *maxDataPoints=100*, *minSpan=None*, *repeat=NOTHING*, *span=None*, *thresholdType='absolute'*, *timeFrom=None*, *timeShift=None*, *transparent=False*, *transformations=NOTHING*, *extraJson=None*, *circleMaxSize=30*, *circleMinSize=2*, *decimals=0*, *geoPoint='geohash'*, *locationData='countries'*, *locationName=''*, *hideEmpty=False*, *hideZero=False*, *initialZoom=1*, *jsonUrl=''*, *jsonpCallback=''*, *mapCenter='(0°, 0°)'*, *mapCenterLatitude=0*, *mapCenterLongitude=0*, *metric='Value'*, *mouseWheelZoom=False*, *stickyLabels=False*, *thresholds='0,100,150'*, *thresholdColors=['#73BF69', '#73BF69', '#FADE2A', '#C4162A']*, *unitPlural=''*, *unitSingle=''*, *unitSingular=''*, *aggregation='total'*)

Bases: *Panel*

Generates Worldmap panel json structure Grafana doc on Worldmap: https://grafana.com/grafana/plugins/grafana-worldmap-panel/

Parameters

- **aggregation** – metric aggregation: min, max, avg, current, total
- **circleMaxSize** – Maximum map circle size
- **circleMinSize** – Minimum map circle size
- **decimals** – Number of decimals to show
- **geoPoint** – Name of the geo_point/geohash column. This is used to calculate where the circle should be drawn.
- **locationData** – Format of the location data, options in *WORLDMAP_LOCATION_DATA*
- **locationName** – Name of the Location Name column. Used to label each circle on the map. If it is empty then the geohash value is used.
- **metric** – Name of the metric column. This is used to give the circle a value - this determines how large the circle is.
- **mapCenter** – Where to centre the map, default center (0°, 0°). Options: North America, Europe, West Asia, SE Asia, Last GeoHash, custom
- **mapCenterLatitude** – If mapCenter=custom set the initial map latitude
- **mapCenterLongitude** – If mapCenter=custom set the initial map longitude
- **hideEmpty** – Hide series with only nulls
- **hideZero** – Hide series with only zeros
- **initialZoom** – Initial map zoom
- **jsonUrl** – URL for JSON location data if *json_endpoint* or *jsonp endpoint* used
- **jsonpCallback** – Callback if *jsonp endpoint* used
- **mouseWheelZoom** – Zoom map on scroll of mouse wheel
- **stickyLabels** – Sticky map labels
- **thresholds** – String of thresholds eg. '0,10,20'
- **thresholdsColors** – List of colors to be used in each threshold

- **unitPlural** – Units plural
- **unitSingle** – Units single
- **unitSingular** – Units singular

to_json_data()

class grafanalib.core.**XAxis**(*mode='time'*, *name=None*, *values=NOTHING*, *show=True*)

Bases: object

X Axis

Parameters

- **mode** – Mode of axis can be time, series or histogram
- **name** – X axis name
- **value** – list of values eg. ["current"] or ["avg"]
- **show** – show X axis

to_json_data()

class grafanalib.core.**YAxes**(*left=NOTHING*, *right=NOTHING*)

Bases: object

The pair of Y axes on a Grafana graph.

Each graph has two Y Axes, a left one and a right one.

to_json_data()

class grafanalib.core.**YAxis**(*decimals=None*, *format=None*, *label=None*, *logBase=1*, *max=None*, *min=None*, *show=True*)

Bases: object

A single Y axis.

Grafana graphs have two Y axes: one on the left and one on the right.

Parameters

- **decimals** – Defines how many decimals are displayed for Y value. (default auto)
- **format** – The display unit for the Y value
- **label** – The Y axis label. (default "")
- **logBase** – The scale to use for the Y value, linear, or logarithmic. (default linear)
- **max** – The maximum Y value
- **min** – The minimum Y value
- **show** – Show or hide the axis

to_json_data()

class grafanalib.core.**ePict**(*dataSource=None*, *targets=NOTHING*, *title=''*, *cacheTimeout=None*, *description=None*, *editable=True*, *error=False*, *height=None*, *gridPos=None*, *hideTimeOverride=False*, *id=None*, *interval=None*, *links=NOTHING*, *maxDataPoints=100*, *minSpan=None*, *repeat=NOTHING*, *span=None*, *thresholds=NOTHING*, *thresholdType='absolute'*, *timeFrom=None*, *timeShift=None*, *transparent=False*, *transformations=NOTHING*, *extraJson=None*, *bgURL=''*, *autoScale=True*, *boxes=[]*)

Bases: *Panel*

Generates ePict panel json structure. https://grafana.com/grafana/plugins/larona-epict-panel/

Parameters

- **autoScale** – Whether to auto scale image to panel size.
- **bgURL** – Where to load the image from.
- **boxes** – The info boxes to be placed on the image.

to_json_data()

class grafanalib.core.**ePictBox**(*angle=0*, *backgroundColor='#000'*, *blinkHigh=False*, *blinkLow=False*, *color='#000'*, *colorHigh='#000'*, *colorLow='#000'*, *colorMedium='#000'*, *colorSymbol=False*, *customSymbol=''*, *decimal=0*, *fontSize=12*, *hasBackground=False*, *hasOrb=False*, *hasSymbol=False*, *isUsingThresholds=False*, *orbHideText=False*, *orbLocation='Left'*, *orbSize=13*, *prefix=''*, *prefixSize=10*, *selected=False*, *serie=''*, *suffix=''*, *suffixSize=10*, *symbol=''*, *symbolDefHeight=32*, *symbolDefWidth=32*, *symbolHeight=32*, *symbolHideText=False*, *symbolWidth=32*, *text='N/A'*, *thresholds=''*, *url=''*, *xpos=0*, *ypos=0*)

Bases: `object`

ePict Box.

Parameters

- **angle** – Rotation angle of box
- **backgroundColor** – Dito
- **blinkHigh** – Blink if below threshold
- **blinkLow** – Blink if above threshold
- **color** – Text color
- **colorHigh** – High value color
- **colorLow** – Low value color
- **colorMedium** – In between value color
- **colorSymbol** – Whether to enable background color for symbol
- **customSymbol** – URL to custom symbol (will set symbol to "custom" if set)
- **decimal** – Number of decimals
- **fontSize** – Dito
- **hasBackground** – Whether to enable background color for text
- **hasOrb** – Whether an orb should be displayed
- **hasSymbol** – Whether a (custom) symbol should be displayed
- **isUsingThresholds** – Whether to enable thresholds.
- **orbHideText** – Whether to hide text next to orb
- **orbLocation** – Orb location (choose from 'Left', 'Right', 'Top' or 'Bottom')
- **orbSize** – Dito

- **prefix** – Value prefix to be displayed (e.g. °C)
- **prefixSize** – Dito
- **selected** – Dont know
- **serie** – Which series to use data from
- **suffix** – Value suffix to be displayed
- **suffixSize** – Dito
- **symbol** – Automatically placed by the plugin format: *<base64>*, check manually.
- **symbolDefHeight** – Dont know
- **symbolDefWidth** – Dont know
- **symbolHeight** – Dito
- **symbolHideText** – Whether to hide value text next to symbol
- **symbolWidth** – Dito
- **text** – Dont know
- **thresholds** – Coloring thresholds: Enter 2 comma-separated numbers. 20,60 will produce: value <= 20 -> green; value between 20 and 60 -> yellow; value >= 60 -> red. If set, it will also set isUsingThresholds to True
- **url** – URL to open when clicked on
- **xpos** – X in (0, X size of image)
- **ypos** – Y in (0, Y size of image)

to_json_data()

grafanalib.core.**is_valid_max_per_row**(*instance*, *attribute*, *value*)

grafanalib.core.**is_valid_target**(*instance*, *attribute*, *value*)

Check if a given attribute is a valid Target

grafanalib.core.**is_valid_triggers**(*instance*, *attribute*, *value*)

Validator for AlertRule triggers

grafanalib.core.**is_valid_triggersv9**(*instance*, *attribute*, *value*)

Validator for AlertRule triggers for Grafana v9

grafanalib.core.**is_valid_xaxis_mode**(*instance*, *attribute*, *value*)

grafanalib.core.**single_y_axis**(**kwargs*)

Specify that a graph has a single Y axis.

Parameters are those passed to *YAxis*. Returns a *YAxes* object (i.e. a pair of axes) that can be used as the yAxes parameter of a graph.

grafanalib.core.**to_y_axes**(*data*)

Backwards compatibility for 'YAxes'.

In grafanalib 0.1.2 and earlier, Y axes were specified as a list of two elements. Now, we have a dedicated *YAxes* type.

This function converts a list of two *YAxis* values to a *YAxes* value, silently passes through *YAxes* values, warns about doing things the old way, and errors when there are invalid values.

2.4 grafanalib.elasticsearch module

Helpers to create Elasticsearch-specific Grafana queries.

class grafanalib.elasticsearch.**AverageMetricAgg**(*field=''*, *id=0*, *hide=False*, *inline=''*)

Bases: object

An aggregator that provides the average. value among the values.

https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-metrics-avg-aggregation.html

Parameters

- **field** – name of elasticsearch field to provide the maximum for
- **id** – id of the metric
- **hide** – show/hide the metric in the final panel display
- **inline** – script to apply to the data, using '_value'

to_json_data()

class grafanalib.elasticsearch.**BucketScriptAgg**(*fields={}*, *id=0*, *hide=False*, *script=''*)

Bases: object

An aggregator that applies a bucket script to the results of previous aggregations. https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-pipeline-bucket-script-aggregation.html

Parameters

- **fields** – dictionary of field names mapped to aggregation IDs to be used in the bucket script e.g. { "field1":1 }, which allows the output of aggregate ID 1 to be referenced as params.field1 in the bucket script
- **script** – script to apply to the data using the variables specified in 'fields'
- **id** – id of the aggregator
- **hide** – show/hide the metric in the final panel display

to_json_data()

class grafanalib.elasticsearch.**CardinalityMetricAgg**(*field=''*, *id=0*, *hide=False*, *inline=''*)

Bases: object

An aggregator that provides the cardinality. value among the values.

https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-metrics-cardinality-aggregation.html

Parameters

- **field** – name of elasticsearch field to provide the maximum for
- **id** – id of the metric
- **hide** – show/hide the metric in the final panel display
- **inline** – script to apply to the data, using '_value'

to_json_data()

class grafanalib.elasticsearch.**CountMetricAgg**(*id=0*, *hide=False*, *inline=''*)

Bases: `object`

An aggregator that counts the number of values.

https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-metrics-valuecount-aggregation.html

It's the default aggregator for elasticsearch queries. :param hide: show/hide the metric in the final panel display :param id: id of the metric :param inline: script to apply to the data, using '_value'

to_json_data()

class grafanalib.elasticsearch.**DateHistogramGroupBy**(*id=0*, *field='time_iso8601'*, *interval='auto'*, *minDocCount=0*)

Bases: `object`

A bucket aggregator that groups results by date.

https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-datehistogram-aggregation.html

Parameters

- **id** – ascending unique number per GroupBy clause
- **field** – name of the elasticsearch field to group by
- **interval** – interval to group by
- **minDocCount** – min. amount of records in the timespan to return a result

to_json_data()

class grafanalib.elasticsearch.**DerivativeMetricAgg**(*field=''*, *hide=False*, *id=0*, *pipelineAgg=1*, *unit=''*)

Bases: `object`

An aggregator that takes the derivative of another metric aggregator.

https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-pipeline-derivative-aggregation.html

Parameters

- **field** – id of elasticsearch metric aggregator to provide the derivative of
- **hide** – show/hide the metric in the final panel display
- **id** – id of the metric
- **pipelineAgg** – pipeline aggregator id
- **unit** – derivative units

to_json_data()

class grafanalib.elasticsearch.**ElasticsearchAlertCondition**(*evaluator=None*, *timeRange=None*, *operator='and'*, *reducerType='last'*, *target=None*, **, type='query'*)

Bases: *AlertCondition*

Override alert condition to support Elasticseach target.

See AlertCondition for more information.

Parameters

- **target** (Target) – Metric the alert condition is based on.
- **evaluator** (Evaluator) – How we decide whether we should alert on the metric. e.g. GreaterThan(5) means the metric must be greater than 5 to trigger the condition. See GreaterThan, LowerThan, WithinRange, OutsideRange, NoValue.
- **timeRange** (TimeRange) – How long the condition must be true for before we alert.
- **operator** – One of OP_AND or OP_OR. How this condition combines with other conditions.
- **reducerType** – RTYPE_*
- **type** – CTYPE_*

class grafanalib.elasticsearch.**ElasticsearchTarget**(*alias=None*, *bucketAggs=NOTHING*, *metricAggs=NOTHING*, *query=''*, *refId=''*, *timeField='@timestamp'*)

Bases: object

Generates Elasticsearch target JSON structure.

Grafana docs on using Elasticsearch: http://docs.grafana.org/features/datasources/elasticsearch/ Elasticsearch docs on querying or reading data: https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html

Parameters

- **alias** – legend alias
- **bucketAggs** – Bucket aggregators
- **metricAggs** – Metric Aggregators
- **query** – query
- **refId** – target reference id
- **timeField** – name of the elasticsearch time field

auto_bucket_agg_ids()

Give unique IDs all bucketAggs without ID.

Returns a new ElasticsearchTarget that is the same as this one, except all of the bucketAggs have their id property set. Any panels which had an id property set will keep that property, all others will have auto-generated IDs provided for them.

If the bucketAggs don't have unique ID associated with it, the generated graph will be broken.

to_json_data()

class grafanalib.elasticsearch.**Filter**(*label=''*, *query=''*)

Bases: object

A Filter for a FilterGroupBy aggregator.

https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-filter-aggregation.html

Parameters

- **label** – label for the metric that is shown in the graph
- **query** – the query to filter by

to_json_data()

class grafanalib.elasticsearch.**FiltersGroupBy**(*id=0*, *filters=NOTHING*)

Bases: object

A bucket aggregator that groups records by a filter expression.

https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-filter-aggregation.html

Parameters

- **id** – ascending unique number per GroupBy clause
- **filters** – list of Filter objects

to_json_data()

class grafanalib.elasticsearch.**MaxMetricAgg**(*field=''*, *id=0*, *hide=False*, *inline=''*)

Bases: object

An aggregator that provides the max. value among the values.

https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-metrics-max-aggregation.html

Parameters

- **field** – name of elasticsearch field to provide the maximum for
- **hide** – show/hide the metric in the final panel display
- **id** – id of the metric
- **inline** – script to apply to the data, using '_value'

to_json_data()

class grafanalib.elasticsearch.**MinMetricAgg**(*field=''*, *id=0*, *hide=False*, *inline=''*)

Bases: object

An aggregator that provides the min. value among the values. https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-metrics-min-aggregation.html :param field: name of elasticsearch field to provide the maximum for :param hide: show/hide the metric in the final panel display :param id: id of the metric :param inline: script to apply to the data, using '_value'

to_json_data()

class grafanalib.elasticsearch.**PercentilesMetricAgg**(*field=''*, *id=0*, *hide=False*, *inline=''*, *percents=NOTHING*, *settings={}*)

Bases: object

A multi-value metrics aggregation that calculates one or more percentiles over numeric values extracted from the aggregated documents https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-metrics-percentile-aggregation.html :param field: name of elasticsearch field to provide the maximum for :param hide: show/hide the metric in the final panel display :param id: id of the metric :param inline: script to apply to the data, using '_value' :param percents: list of percentiles, like [95,99]

to_json_data()

class grafanalib.elasticsearch.**SumMetricAgg**(*field=''*, *id=0*, *hide=False*, *inline=''*)

Bases: object

An aggregator that provides the sum of the values. https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-metrics-sum-aggregation.html :param field: name of elasticsearch field to provide the sum over :param hide: show/hide the metric in the final panel display :param id: id of the metric :param inline: script to apply to the data, using '_value'

to_json_data()

class grafanalib.elasticsearch.**TermsGroupBy**(*field*, *id=0*, *minDocCount=1*, *order='desc'*, *orderBy='_term'*, *size=0*)

Bases: object

A multi-bucket aggregator based on field values.

https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-terms-aggregation.html

Parameters

- **id** – ascending unique number per GroupBy clause
- **field** – name of the field to group by
- **minDocCount** – min. amount of matching records to return a result
- **order** – ORDER_ASC or ORDER_DESC
- **orderBy** – term to order the bucket Term value: '_term', Doc Count: '_count' or to use metric function use the string value "2"
- **size** – how many buckets are returned

to_json_data()

2.5 grafanalib.formatunits module

Grafana unit formats (https://github.com/grafana/grafana/blob/main/packages/grafana-data/src/valueFormats/categories.ts)

To use: from grafanalib import formatunits as UNITS

format = UNITS.BYTES

2.6 grafanalib.influxdb module

Helpers to create InfluxDB-specific Grafana queries.

class grafanalib.influxdb.**InfluxDBTarget**(*alias=''*, *format='time_series'*, *datasource=''*, *measurement=''*, *query=''*, *rawQuery=True*, *refId=''*)

Bases: object

Generates InfluxDB target JSON structure.

Grafana docs on using InfluxDB: https://grafana.com/docs/features/datasources/influxdb/ InfluxDB docs on querying or reading data: https://v2.docs.influxdata.com/v2.0/query-data/

Parameters

- **alias** – legend alias
- **format** – Bucket aggregators
- **datasource** – Influxdb name (for multiple datasource with same panel)
- **measurement** – Metric Aggregators
- **query** – query
- **rawQuery** – target reference id
- **refId** – target reference id

to_json_data()

2.7 grafanalib.opentsdb module

Support for OpenTSDB.

class grafanalib.opentsdb.**OpenTSDBFilter**(*value*, *tag*, *type='literal_or'*, *groupBy=False*)

Bases: object

to_json_data()

class grafanalib.opentsdb.**OpenTSDBTarget**(*metric*, *refId=''*, *aggregator='sum'*, *alias=None*, *isCounter=False*, *counterMax=None*, *counterResetValue=None*, *disableDownsampling=False*, *downsampleAggregator='sum'*, *downsampleFillPolicy='none'*, *downsampleInterval=None*, *filters=NOTHING*, *shouldComputeRate=False*, *currentFilterGroupBy=False*, *currentFilterKey=''*, *currentFilterType='literal_or'*, *currentFilterValue=''*)

Bases: object

Generates OpenTSDB target JSON structure.

Grafana docs on using OpenTSDB: http://docs.grafana.org/features/datasources/opentsdb/ OpenTSDB docs on querying or reading data: http://opentsdb.net/docs/build/html/user_guide/query/index.html

Parameters

- **metric** – OpenTSDB metric name
- **refId** – target reference id
- **aggregator** – defines metric aggregator. The list of opentsdb aggregators: http://opentsdb.net/docs/build/html/user_guide/query/aggregators.html#available-aggregators
- **alias** – legend alias. Use patterns like $tag_tagname to replace part of the alias for a tag value.
- **isCounter** – defines if rate function results should be interpret as counter
- **counterMax** – defines rate counter max value
- **counterResetValue** – defines rate counter reset value
- **disableDownsampling** – defines if downsampling should be disabled. OpenTSDB docs on downsampling: http://opentsdb.net/docs/build/html/user_guide/query/index.html#downsampling

- **downsampleAggregator** – defines downsampling aggregator
- **downsampleFillPolicy** – defines downsampling fill policy
- **downsampleInterval** – defines downsampling interval
- **filters** – defines the list of metric query filters. OpenTSDB docs on filters: http://opentsdb.net/docs/build/html/user_guide/query/index.html#filters
- **shouldComputeRate** – defines if rate function should be used. OpenTSDB docs on rate function: http://opentsdb.net/docs/build/html/user_guide/query/index.html#rate
- **currentFilterGroupBy** – defines if grouping should be enabled for current filter
- **currentFilterKey** – defines current filter key
- **currentFilterType** – defines current filter type
- **currentFilterValue** – defines current filter value

`to_json_data()`

2.8 grafanalib.prometheus module

Helpers for Prometheus-driven graphs.

`grafanalib.prometheus.PromGraph(data_source, title, expressions, **kwargs)`

Create a graph that renders Prometheus data.

Parameters

- **data_source** (*str*) – The name of the data source that provides Prometheus data.
- **title** – The title of the graph.
- **expressions** – List of tuples of (legend, expr), where 'expr' is a Prometheus expression. Or a list of dict where keys are Target's args.
- **kwargs** – Passed on to Graph.

2.9 grafanalib.validators module

`grafanalib.validators.is_color_code(instance, attribute, value)`

A validator that raises a `ValueError` if attribute value is not valid color code. Value considered as valid color code if it starts with # char followed by hexadecimal.

`grafanalib.validators.is_in(choices)`

A validator that raises a `ValueError` if the attribute value is not in a provided list.

Parameters

choices – List of valid choices

`grafanalib.validators.is_interval(instance, attribute, value)`

A validator that raises a `ValueError` if the attribute value is not matching regular expression.

`grafanalib.validators.is_list_of(etype)`

A validator that raises a `ValueError` if the attribute value is not in a provided list.

Parameters

choices – List of valid choices

2.10 grafanalib.weave module

Weave-specific dashboard configuration.

Unlike 'core', which has logic for building generic Grafana dashboards, this has our Weave-specific preferences.

`grafanalib.weave.`**`PercentUnitAxis`**(*label=None*)

A Y axis that shows a percentage based on a unit value.

`grafanalib.weave.`**`QPSGraph`**(*data_source*, *title*, *expressions*, ***kwargs*)

Create a graph of QPS, broken up by response code.

Data is drawn from Prometheus.

Parameters

- **`title`** – Title of the graph.
- **`expressions`** – List of Prometheus expressions. Must be 5.
- **`kwargs`** – Passed on to Graph.

`grafanalib.weave.`**`stacked`**(*graph*)

Turn a graph into a stacked graph.

2.11 grafanalib.zabbix module

class `grafanalib.zabbix.`**`ZabbixAggregateByFunction`**(*added=False*, *interval='1m'*, *function='avg'*)

Bases: `object`

Takes all timeseries and consolidate all its points falled in given interval into one point using function, which can be one of: avg, min, max, median. https://alexanderzobnin.github.io/grafana-zabbix/reference/functions/#aggregateBy

`to_json_data`()

class `grafanalib.zabbix.`**`ZabbixAverageFunction`**(*added=False*, *interval='1m'*)

Bases: `object`

Deprecated, use aggregateBy(interval, avg) instead. https://alexanderzobnin.github.io/grafana-zabbix/reference/functions/#average

`to_json_data`()

class `grafanalib.zabbix.`**`ZabbixBottomFunction`**(*added=False*, *number=5*, *function='avg'*)

Bases: `object`

`to_json_data`()

class `grafanalib.zabbix.`**`ZabbixColor`**(*color*, *priority*, *severity*, *show=True*)

Bases: `object`

`to_json_data`()

class `grafanalib.zabbix.`**`ZabbixDeltaFunction`**(*added=False*)

Bases: `object`

Convert absolute values to delta, for example, bits to bits/sec https://alexanderzobnin.github.io/grafana-zabbix/reference/functions/#delta

to_json_data()

class grafanalib.zabbix.**ZabbixGroupByFunction**(*added=False*, *interval='1m'*, *function='avg'*)

Bases: object

Takes each timeseries and consolidate its points falled in given interval into one point using function, which can be one of: avg, min, max, median. https://alexanderzobnin.github.io/grafana-zabbix/reference/functions//#groupBy

to_json_data()

class grafanalib.zabbix.**ZabbixMaxFunction**(*added=False*, *interval='1m'*)

Bases: object

Deprecated, use aggregateBy(interval, max) instead. https://alexanderzobnin.github.io/grafana-zabbix/reference/functions/#max

to_json_data()

class grafanalib.zabbix.**ZabbixMedianFunction**(*added=False*, *interval='1m'*)

Bases: object

Deprecated, use aggregateBy(interval, median) instead. https://alexanderzobnin.github.io/grafana-zabbix/reference/functions/#median

to_json_data()

class grafanalib.zabbix.**ZabbixMinFunction**(*added=False*, *interval='1m'*)

Bases: object

Deprecated, use aggregateBy(interval, min) instead. https://alexanderzobnin.github.io/grafana-zabbix/reference/functions/#min

to_json_data()

class grafanalib.zabbix.**ZabbixScaleFunction**(*added=False*, *factor=100*)

Bases: object

Takes timeseries and multiplies each point by the given factor. https://alexanderzobnin.github.io/grafana-zabbix/reference/functions//#scale

to_json_data()

class grafanalib.zabbix.**ZabbixSetAliasByRegexFunction**(*regexp*, *added=False*)

Bases: object

Returns part of the metric name matched by regex. https://alexanderzobnin.github.io/grafana-zabbix/reference/functions/#setAliasByRegex

to_json_data()

class grafanalib.zabbix.**ZabbixSetAliasFunction**(*alias*, *added=False*)

Bases: object

Returns given alias instead of the metric name. https://alexanderzobnin.github.io/grafana-zabbix/reference/functions/#setAlias

to_json_data()

class grafanalib.zabbix.**ZabbixSumSeriesFunction**(*added=False*)

Bases: object

This will add metrics together and return the sum at each datapoint. This method required interpolation of each timeseries so it may cause high CPU load. Try to combine it with groupBy() function to reduce load. https://alexanderzobnin.github.io/grafana-zabbix/reference/functions/#sumSeries

to_json_data()

class grafanalib.zabbix.**ZabbixTarget**(*application=''*, *expr=''*, *functions=NOTHING*, *group=''*, *host=''*, *intervalFactor=2*, *item=''*, *itService=''*, *mode=0*, *options=NOTHING*, *refId=''*, *slaProperty=NOTHING*, *textFilter=''*, *useCaptureGroups=False*)

Bases: object

Generates Zabbix datasource target JSON structure.

Grafana-Zabbix is a plugin for Grafana allowing to visualize monitoring data from Zabbix and create dashboards for analyzing metrics and realtime monitoring.

Grafana docs on using Zabbix plugin: https://alexanderzobnin.github.io/grafana-zabbix/

Parameters

- **application** – zabbix application name
- **expr** – zabbix arbitary query
- **functions** – list of zabbix aggregation functions
- **group** – zabbix host group
- **host** – hostname
- **intervalFactor** – defines interval between metric queries
- **item** – regexp that defines which metric to query
- **itService** – zabbix it service name
- **mode** – query mode type
- **options** – additional query options
- **refId** – target reference id
- **slaProperty** – zabbix it service sla property. Zabbix returns the following availability information about IT service Status - current status of the IT service SLA - SLA for the given time interval OK time - time the service was in OK state, in seconds Problem time - time the service was in problem state, in seconds Down time - time the service was in scheduled downtime, in seconds
- **textFilter** – query text filter. Use regex to extract a part of the returned value.
- **useCaptureGroups** – defines if capture groups should be used during metric query

to_json_data()

class grafanalib.zabbix.**ZabbixTargetField**(*filter=''*)

Bases: object

to_json_data()

class grafanalib.zabbix.**ZabbixTargetOptions**(*showDisabledItems=False*)

Bases: object

to_json_data()

class grafanalib.zabbix.**ZabbixTimeShiftFunction**(*added=False*, *interval='24h'*)

Bases: object

Draws the selected metrics shifted in time. If no sign is given, a minus sign (-) is implied which will shift the metric back in time. If a plus sign (+) is given, the metric will be shifted forward in time. https://alexanderzobnin.github.io/grafana-zabbix/reference/functions/#timeShift

to_json_data()

class grafanalib.zabbix.**ZabbixTopFunction**(*added=False*, *number=5*, *function='avg'*)

Bases: object

to_json_data()

class grafanalib.zabbix.**ZabbixTrendValueFunction**(*added=False*, *type='avg'*)

Bases: object

Specifying type of trend value returned by Zabbix when trends are used (avg, min or max). https://alexanderzobnin.github.io/grafana-zabbix/reference/functions/#trendValue

to_json_data()

class grafanalib.zabbix.**ZabbixTrigger**(*application=''*, *group=''*, *host=''*, *trigger=''*)

Bases: object

to_json_data()

class grafanalib.zabbix.**ZabbixTriggersPanel**(*dataSource*, *title*, *ackEventColor=NOTHING*, *ageField=True*, *customLastChangeFormat=False*, *description=''*, *fontSize=NOTHING*, *height=Pixels(num=250)*, *hideHostsInMaintenance=False*, *hostField=True*, *hostTechNameField=False*, *id=None*, *infoField=True*, *lastChangeField=True*, *lastChangeFormat=''*, *limit=10*, *links=NOTHING*, *markAckEvents=False*, *minSpan=None*, *okEventColor=NOTHING*, *pageSize=10*, *repeat=None*, *scroll=True*, *severityField=False*, *showEvents=NOTHING*, *showTriggers='all triggers'*, *sortTriggersBy=NOTHING*, *span=None*, *statusField=False*, *transparent=False*, *triggerSeverity=(('#B7DBAB', 'Not classified'), ('#82B5D8', 'Information'), ('#E5AC0E', 'Warning'), ('#C15C17', 'Average'), ('#BF1B00', 'High'), ('#890F02', 'Disaster'))*, *triggers=NOTHING*)

Bases: object

Parameters

- **dataSource** – query datasource name
- **title** – panel title
- **ackEventColor** – acknowledged triggers color
- **customLastChangeFormat** – defines last change field data format. See momentjs docs for time format: http://momentjs.com/docs/#/displaying/format/

- **description** – additional panel description
- **fontSize** – panel font size
- **height** – panel height in Pixels
- **hideHostsInMaintenance** – defines if triggers form hosts in maintenance should be shown
- **hostField** – defines if host field should be shown
- **hostTechNameField** – defines if field with host technical name should be shown
- **id** – panel identificator
- **infoField** – defines if field with host info should be shown
- **lastChangeField** – defines if field with last change time should be shown
- **limit** – defines number of queried triggers
- **links** – list of dashboard links
- **markAckEvents** – defines if acknowledged triggers should be colored with different color
- **minSpan** – defines panel minimum spans
- **okEventColor** – defines color for triggers with Ok status
- **pageSize** – defines number of triggers per panel page
- **scroll** – defines if scroll should be shown
- **severityField** – defines if severity field should be shown
- **showEvents** – defines event type to query (Ok, Problems, All)
- **showTriggers** – defines trigger type to query (all, acknowledged, unacknowledged)
- **sortTriggersBy** – defines trigger sort policy
- **span** – defines span number for panel
- **statusField** – defines if status field should be shown
- **transparent** – defines if panel should be transparent
- **triggerSeverity** – defines colors for trigger severity,
- **triggers** – trigger query

to_json_data()

grafanalib.zabbix.**convertZabbixSeverityColors**(*colors*)

grafanalib.zabbix.**zabbixMetricTarget**(*application*, *group*, *host*, *item*, *functions=[]*)

grafanalib.zabbix.**zabbixServiceTarget**(*service*, *sla={'name': 'Status', 'property': 'status'}*)

grafanalib.zabbix.**zabbixTextTarget**(*application*, *group*, *host*, *item*, *text*, *useCaptureGroups=False*)

2.12 Module contents

Routines for building Grafana dashboards.

CHAPTER

THREE

GRAFANALIB

CHAPTER

FOUR

CONTRIBUTING TO GRAFANALIB

Thank you for contributing to grafanalib! Here are some notes to help you get your PR merged as quickly as possible, and to help us remember how to review things properly.

If something comes up during a code review or on a ticket that you think should be part of these guidelines, please say so, or even file a PR to make this doc better!

4.1 Code of conduct

We have a *code of conduct*, and we enforce it. Please take a look!

4.2 Coding guidelines

- Python 3 all the way
- Must be flake8 compliant
- We use attrs everywhere
- Avoid inheritance as much as possible
- Avoid mutation as much as possible—keep things purely functional
- Docstrings are great, let's have more of those
- Link to official Grafana docs in comments as much as possible

4.2.1 Conventions

- Classes are `StudlyCaps`
- Attributes are `camelCased`
- Methods are `snake_cased`
- Local variables are `snake_cased`
- We're kind of fussy about indentation: 4 spaces everywhere, follow the examples in core.py if you're uncertain
- Triple Double quotes `"""` for docstrings
- Double quotes `""` for human readable message or when string used for interpolation
- Single quotes `''` for symbol like strings

4.2.2 Testing

Lots of grafanalib is just simple data structures, so we aren't fastidious about test coverage.

However, tests are strongly encouraged for anything with non-trivial logic. Please try to use hypothesis for your tests.

```
$ make all
```

4.2.3 Gotchas

- Do **not** use mutable values as default values for attributes. Mutable values include lists (e.g. `default=[RED, GREEN]`) and other grafanalib objects (e.g. `default=Annotations()`). Instead, use attr.Factory. e.g. `default=attr.Factory(Annotations)` or `default=attr.Factory(lambda: [RED, GREEN])`.

4.3 Submitting a PR

- We are very grateful for all PRs, and deeply appreciate the work and effort involved!
- We try to review PRs as quickly as possible, but it might take a couple of weeks to get around to reviewing your PR—sorry, we know that sucks
- Please add an entry to the *CHANGELOG* in your PR
- It helps a lot if the PR description provides some context on what you are trying to do and why you think it's a good idea
- The smaller the PR, the more quickly we'll be able to review it

4.4 Filing a bug

- Please say what you saw, what you expected to see, and how someone else can reproduce the bug
- If it comes with a test case, even better!

CHAPTER

FIVE

COMMUNITY CODE OF CONDUCT

Weaveworks follows the CNCF Community Code of Conduct v1.0.

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting a Weaveworks project maintainer, or *Alexis Richardson <alexis@weave.works>*.

CHAPTER

SIX

RELEASE PROCESS

6.1 Pre-release

- Pick a new version number (e.g. `X.Y.Z`)
- Update CHANGELOG with that number
- Update setup.py with that number

6.2 Smoke-testing

- Run

```
$ python setup.py install --user
```

- Check `~/.local/bin/generate-dashboard` for the update version.
- Try the example on README.

6.3 Releasing

- Head to https://github.com/weaveworks/grafanalib/releases/new and create the release there.
- Wait for GitHub Actions to complete the build and release.
- Confirm on https://pypi.org/project/grafanalib/ that the release made it there.

6.4 Follow-up

- Run

```
$ pip intall grafanalib -U
```

- Check if the upgrade worked and the test above still passes.

CHAPTER

SEVEN

CHANGELOG

7.1 0.7.0 (2022-10-02)

- Added Grafana 8.x new Alert Rule
- Added Grafana 9.x new Alert Rule
- Added ePict plugin.
- Added ae3e plotly panel support
- Added datasource parameter to Influxdb targets
- Added missing units for Boolean, Data, Data Rate, Date & Time, Energy, Length, Mass, and Misc
- Fix typo in unit constant `GIGA_WATT` (was `GAGA_WATT`)
- Fix typo in unit constant `NORMAL_CUBIC_METER` (was `NORMAIL_CUBIC_METER`)

7.2 0.6.3 (2022-03-30)

- Added Azure Monitor Target
- Added `legendCalcs` parameter to TimeSeries Panel
- Added `hide` parameter to ElasticsearchTarget
- Added ExpressionTarget support for ElasticSearch data sources

7.3 0.6.2 (2022-02-24)

- Added percentage type for thresholds
- Added `datasource` parameter to CloudWatch targets
- Added support for auto panels ids to AlertList panel
- Added `SeriesOverride` options (dashes and Z-index)
- Added support for fields value in Stat panel
- Added `alertName` parameter to AlertList panel
- Added `thresholdsStyleMode` parameter to TimeSeries panel
- Added Histogram panel support

- Dashboard upload script updated to support overwriting dashboards

7.4 0.6.1 (2021-11-23)

- Added new SqlTarget to core to be able to define SQL queries as well
- Added missing attributes to the Logs panel
- Added Cloudwatch Logs Insights Target
- Added overrides to panels
- Extend `SeriesOverride` options

7.4.1 Changes

- Fix Text panel (and add tests)

  ATTENTION: This might break panels generated for Grafana <8.0.6

7.5 0.6.0 (2021-10-26)

- Added Discrete panel (https://grafana.com/grafana/plugins/natel-discrete-panel/)
- Added support for colors in stat mapping panel with StatValueMappings & StatRangeMappings
- Added missing auto interval properties in Template
- Added param to RowPanel to collapse the row
- Added StateTimeline panel which was added in Grafana v8
- Added support for timeseries panel added in Grafana v8
- Added MinMetricAgg and PercentilesMetricAgg to Elasticsearch
- Added support for News panel
- Added support for Pie Chart v2 from Grafana v8

7.5.1 Changes

- Refine expectations of is_color_code
- Deprecated StatMapping, StatValueMapping & StatRangeMapping
- Change YAxis min value default from 0 to None
- Support for Table panel for Grafana v8 may have broken backwards compatibility in old Table panel
- Breaking change, support for styled columns in tables removed, no longer used in Grafana v8 new Table
- Move development to `main` branch on GitHub. If you have work tracking the `master` you will need to update this.

7.6 0.5.14 (2021-09-14)

- Added colour overrides to pie chart panel
- Added missing attributes from xAxis class
- Added transformations for the Panel class (https://grafana.com/docs/grafana/latest/panels/reference-transformation-functions/)
- Added Worldmap panel (https://grafana.com/grafana/plugins/grafana-worldmap-panel/)
- Added missing fill gradient to Graph panel
- Added missing align to graph panel
- Added missing show percentage attribute to Pie chart panel
- Added `extraJson` attribute to the Panel class for overriding the panel with raw JSON
- Added inline script support for Elasticsearch metrics
- Selected needs to be set as a bool value for templating to work.

7.7 0.5.13 (2021-05-17)

- Added a test for the Alert class.

7.7.1 Changes

- Bugfix: changed 'target' validator in AlertNotification to accept CloudwatchMetricsTarget
- Moved the alertRuleTag field from Graph to Alert.

7.8 0.5.12 (2021-04-24)

- Added hide parameter to CloudwatchMetricsTarget class
- Added table-driven example dashboard and upload script

7.8.1 Changes

- bugfix load_dashboard add support for old python version 2.x, 3.3 and 3.4
- Fix default target datasource to work with newer versions of Grafana
- Removed re-defined maxDataPoints field from multiple panels
- Fix the AlertList class and add a test for it

Thanks to all those who have contributed to this release.

7.9 0.5.11 (2021-04-06)

- Added timeField field for the Elasticsearch target to allow the alert to change its state
- Added nameFilter field for the AlertList panel
- Added dashboardTags field for the AlertList panel

Thanks a lot for your contributions to this release, @dafna-starkware

7.10 0.5.10 (2021-03-21)

- Added Logs panel (https://grafana.com/docs/grafana/latest/panels/visualizations/logs-panel/)
- Added Cloudwatch metrics datasource (https://grafana.com/docs/grafana/latest/datasources/cloudwatch/)
- Added option to hide dashboard time picker
- Added Notification for Alert
- Added alertRuleTags field to the graph panel
- Added support for thresholds to graph panel
- Added support for Elasticsearch alert condition
- Added support for using gridPos for dashboard panels
- Added support for Humio Data Source. (https://grafana.com/grafana/plugins/humio-datasource/)

7.10.1 Changes

- Replace deprecated attr.assoc with attr.evolve

7.11 0.5.9 (2020-12-18)

- Added Alert Threshold enabled/disabled to Graphs.
- Added constants for all Grafana value formats
- Added support for repetitions to Stat Panels
- Added textMode option to Stat Panels
- Add Panel object for all panels to inherit from
- Add Dashboard list panel (https://grafana.com/docs/grafana/latest/panels/visualizations/dashboard-list-panel/)

7.11.1 Changes

- Change supported python versions from 3.6 to 3.9
- Added hide parameter to Target
- Updated dependencies (docs, build, CI)
- Consistent coding style

7.12 0.5.8 (2020-11-02)

This release adds quite a few new classes to grafanalib, ElasticSearch support was improved and support for InfluxDB data sources was added.

We would also very much like to welcome James Gibson as new maintainer of grafanalib. Thanks a lot for stepping up to this role!

7.12.1 Changes

- Added more YAxis formats, added Threshold and SeriesOverride types
- dataLinks support in graphs
- Add Elasticsearch bucket script pipeline aggregator
- Added ability to hide metrics for Elasticsearch MetricAggs
- Add derivative metric aggregation for Elasticsearch
- Add `Stat` class (and `StatMapping`, `StatValueMapping`, `StatRangeMapping`) to support the Stat panel
- Add `Svg` class to support the SVG panel
- Add `PieChart` class for creating Pie Chart panels
- Add *transparent* setting to classes that were missing it (Heatmap, PieChart)
- Add InfluxDB data source
- Add `auto_ref_ids` to `Graph`

Thanks a lot for your contributions to this release, @DWalker487, @JamesGibo, @daveworth, @dholbach, @fauust, @larsderidder, @matthewmrichter.

7.13 0.5.7 (2020-05-11)

7.13.1 Changes

- Fix crasher instatiating elasticsearch panels.
- Remove unused `tools/` directory.

Thanks a lot for your contributions to this release, @DWalker487, @dholbach and @matthewmrichter.

7.14 0.5.6 (2020-05-05)

7.14.1 Changes

- Add `Heatmap` class (and `HeatmapColor`) to support the Heatmap panel (#170)
- Add `BarGuage` for creating bar guages panels in grafana 6
- Add `GuagePanel` for creating guages in grafana 6
- Add data links support to `Graph`, `BarGuage`, and `GuagePanel` panels
- Removed gfdatasource - feature is built in to Grafana since v5.
- Generate API docs for readthedocs.org
- Fix AlertList panel generation
- Add both upper and lower case *"time"* pattern for time_series column format in Table class
- Drop testing of Python 2.7, it has been EOL'ed and CI was broken due to this.
- Automatically test documentation examples.
- Point to dev meeting resources.
- Add description attribute to Dashboard.
- Add support for custom variables.
- Point out documentation on readthedocs more clearly.
- Add average metric aggregation for elastic search
- Bugfix to query ordering in Elasticsearch TermsGroupBy
- Added all parameters for StringColumnStyle
- Add Elasticsearch Sum metric aggregator
- Add `Statusmap` class (and `StatusmapColor`) to support the Statusmap panel plugin
- Bugfix to update default `Threshold` values for `GaugePanel` and `BarGauge`
- Use Github Actions for CI.
- Fix test warnings.
- Update `BarGauge` and `GaugePanel` default Threshold values.
- Update release instructions.

Thanks a lot to the contributions from @DWalker487, @bboreham, @butlerx, @dholbach, @franzs, @jaychitalia95, @matthewmrichter and @number492 for this release!

7.15 0.5.5 (2020-02-17)

It's been a while since the last release and we are happy to get this one into your hands. 0.5.5 is a maintenance release, most importantly it adds support for Python >= 3.5.

We are very delighted to welcome Matt Richter on board as maintainer.

7.15.1 Changes

- Automate publishing to PyPI with GitHub Actions
- Update README.rst to make the example work
- Bump Dockerfile to use Alpine 3.10 as base
- Fix up `load_source()` call which doesn't exist in Python 3.5
- Update versions of Python tested
- Repair tests
- pin to attrs 19.2 and fix deprecated arguments

Many thanks to contributors @bboreham, @dholbach, @ducksecops, @kevingessner, @matthewmrichter, @uritau.

7.16 0.5.4 (2019-08-30)

7.16.1 Changes

- Add 'diff', 'percent_diff' and 'count_non_null' as RTYPE
- Support for changing sort value in Template Variables.
- Sort tooltips by value in Weave/Stacked-Charts
- Add `for` parameter for alerts on Grafana 6.X
- Add `STATE_OK` for alerts
- Add named values for the Template.hide parameter
- Add cardinality metric aggregator for ElasticSearch
- Add Threshold and Series Override types
- Add more YAxis formats

Many thanks to contributors @kevingessner, @2easy, @vicmarbev, @butlerx.

7.17 0.5.3 (2018-07-19)

7.17.1 Changes

- Minor markup tweaks to the README

7.18 0.5.2 (2018-07-19)

7.18.1 Fixes

- `PromGraph` was losing all its legends. It doesn't any more. (#130)

7.18.2 Changes

- Add `AlertList` panel support
- Add support for mixed data sources
- Add `ExternalLink` class for dashboard-level links to other pages
- Template now supports 'type' and 'hide' attributes
- Legend now supports `sort` and `sortDesc` attributes
- Tables now support `timeFrom` attribute
- Update README.rst with information on how to get help.

7.19 0.5.1 (2018-02-27)

7.19.1 Fixes

- Fix for crasher bug that broke `SingleStat`, introduced by #114

7.20 0.5.0 (2018-02-26)

7.20.1 New features

- grafanalib now supports Python 2.7. This enables it to be used within Bazel.
- Partial support for graphs against Elasticsearch datasources (https://github.com/weaveworks/grafanalib/pull/99)

7.20.2 Extensions

- Constants for days, hours, and minutes (https://github.com/weaveworks/grafanalib/pull/98)
- Groups and tags can now be used with templates (https://github.com/weaveworks/grafanalib/pull/97)

7.21 0.4.0 (2017-11-23)

Massive release!

It's Thanksgiving today, so more than ever I want to express my gratitude to all the people who have contributed to this release!

- @aknuds1
- @atopuzov
- @bboreham
- @fho
- @filippog
- @gaelL
- @lalinsky
- @leth
- @lexfrei
- @mikebryant

7.21.1 New features

- Support for `Text` panels (https://github.com/weaveworks/grafanalib/pull/63)
- `PromGraph` is now more powerful. If you want to pass extra parameters like `intervalFactor` to your targets, you can do so by listing targets as dictionaries, rather than tuples. (https://github.com/weaveworks/grafanalib/pull/66)
- Support for absolute links to drill-down in graphs (https://github.com/weaveworks/grafanalib/pull/86)

7.21.2 Changes

- Breaking change to `weave.QPSGraph()` - added `data_source` parameter and removed old hard-coded setting. (https://github.com/weaveworks/grafanalib/pull/77)

7.21.3 Extensions

Generally adding more parameters to existing things:

- Graphs can now have descriptions or be transparent (https://github.com/weaveworks/grafanalib/pull/62 https://github.com/weaveworks/grafanalib/pull/89)
- New formats: "bps" and "Bps" (https://github.com/weaveworks/grafanalib/pull/68)
- Specify the "Min step" for a `Target` using the `interval` attribute.
- Specify the number of decimals shown on the `YAxis` with the `decimals` attribute
- Specify multiple `Dashboard` inputs, allowing dashboards to be parametrized by data source. (https://github.com/weaveworks/grafanalib/pull/83)
- Templates * `label` is now optional (https://github.com/weaveworks/grafanalib/pull/92) * `allValue` and `includeAll` attributes now available (https://github.com/weaveworks/grafanalib/pull/67) * `regex` and `multi` attributes now available (https://github.com/weaveworks/grafanalib/pull/82)
- Rows can now repeat (https://github.com/weaveworks/grafanalib/pull/82)
- Add missing `NULL_AS_NULL` constant
- Specify the "Instant" for a `Target` using the `instant` attribute.

7.21.4 Fixes

- The `showTitle` parameter in `Row` is now respected (https://github.com/weaveworks/grafanalib/pull/80)

7.22 0.3.0 (2017-07-27)

7.22.1 New features

- OpenTSDB datasource support (https://github.com/weaveworks/grafanalib/pull/27)
- Grafana Zabbix plugin support (https://github.com/weaveworks/grafanalib/pull/31, https://github.com/weaveworks/grafanalib/pull/36)
- `Dashboard` objects now have an `auto_panel_id` method which will automatically supply unique panel (i.e. graph) IDs for any panels that don't have one set. Dashboard config files no longer need to track their own `GRAPH_ID` counter.
- Support for `SingleStat` panels (https://github.com/weaveworks/grafanalib/pull/22)
- `single_y_axis` helper for the common case of a graph that has only one Y axis

7.22.2 Improvements

- `PromGraph` now lives in `grafanalib.prometheus`, and takes a `data_source` parameter
- Additional fields for `Legend` (https://github.com/weaveworks/grafanalib/pull/25)
- Additional fields for `XAxis` (https://github.com/weaveworks/grafanalib/pull/28)
- Get an error when you specify the wrong number of Y axes

7.22.3 Changes

- New `YAxes` type for specifying Y axes. Using a list of two `YAxis` objects is deprecated.

7.23 0.1.2 (2017-01-02)

- Add support for Grafana Templates (https://github.com/weaveworks/grafanalib/pull/9)

7.24 0.1.1 (2016-12-02)

- Include README on PyPI page

7.25 0.1.0 (2016-12-02)

Initial release.

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

g

INDEX

H

I

L

M

N

O

P

Q

R

S

T

V

W

X

Y

Z