grafana-pcp Documentation

Release 3.0.3

Performance Co-Pilot

Feb 24, 2021

Getting started

Performance Co-Pilot (PCP) provides a framework and services to support system-level performance monitoring and management. It presents a unifying abstraction for all of the performance data in a system, and many tools for interrogating, retrieving, and processing that data.

CHAPTER 1

Features

- analysis of historical PCP metrics using pmseries query language
- analysis of real-time PCP metrics using pmwebapi live services
- enhanced Berkeley Packet Filter (eBPF) tracing using bpftrace scripts
- dashboards for detecting potential performance issues and show possible solutions with the checklist dashboards, using the USE method [2]
- full-text search in metric names, descriptions, instances [1]
- support for Grafana Alerting [1]
- support for derived metrics (allows the usage of arithmetic operators and statistical functions inside a query) [2]
- automated configuration of metric units [1,2,3]
- automatic rate and time utilization conversion
- heatmap, table [2,3] and flame graph [3] support
- auto-completion of metric names [1,2], qualifier keys and values [1], and bpftrace probes, builtin variables and functions [3]
- display of semantics, units and help texts of metrics [2] and bpftrace builtins [3]
- legend templating support with `$metric`, `$metric0`, `$instance`, `$some_label`, `$some_dashboard_variable`
- container support [1,2]
- support for custom endpoint and hostspec per panel [2,3]
- support for repeated panels
- sample dashboards for all data sources

[1] PCP Redis [2] PCP Vector [3] PCP bpftrace

CHAPTER 2

Getting started

2.1 Quickstart

2.1.1 Installation (Fedora)

```
$ sudo dnf install grafana-pcp
$ sudo systemctl restart grafana-server
$ sudo systemctl start pmproxy
```

For other distributions, please refer to the *Installation Guide*.

After Grafana and grafana-pcp are installed, you can enable the plugin: Open the Grafana configuration, go to Plugins, select *Performance Co-Pilot*, and click the *Enable* button.

2.1.2 Data Sources

Before using grafana-pcp, you need to configure the data sources. Open the Grafana configuration, go to Data Sources and add the *PCP Redis*, *PCP Vector* and/or *PCP bpftrace* datasources.

The only required configuration field for each data source is the URL to pmproxy. In most cases, the default setting of `http://localhost:44322` can be used. All other fields can be left to their default values.

Note: Make sure the URL text box actually contains a value (font color should be white) and not the placeholder value (light grey text).

Note: The Redis and bpftrace data sources need additional configuration on the collector host. See *PCP Redis* and *PCP bpftrace*.

2.1.3 Dashboards

After installing grafana-pcp and configuring the data sources, you're ready to open the pre-installed dashboards or create new ones. Each data source comes with a few pre-installed dashboards, showing most of the respective functionality. Further information on each data source and the functionality can be found in the *Data Sources* section.

2.2 Installation

2.2.1 Distribution Package

Distribution Package is the recommended method of installing grafana-pcp.

Fedora

```
$ sudo dnf install grafana-pcp
$ sudo systemctl restart grafana-server
```

2.2.2 GitHub Release

If there is no package available for your distribution, you can install a release from GitHub. Replace X.Y.Z with the version of grafana-pcp you wish to install.

```
$ wget https://github.com/performancecopilot/grafana-pcp/releases/download/vX.Y.Z/
↪performancecopilot-pcp-app-X.Y.Z.zip
$ sudo unzip -d /var/lib/grafana/plugins performancecopilot-pcp-app-X.Y.Z.zip
$ sudo systemctl restart grafana-server
```

2.2.3 Container

You can also run Grafana with grafana-pcp in a container, using podman or docker. Keep in mind that with the default configuration, every container has its own isolated network, and you won't be able to reach pmproxy through localhost. Replace X.Y.Z with the version of grafana-pcp you wish to install.

```
$ podman run -e GF_INSTALL_PLUGINS="https://github.com/performancecopilot/grafana-pcp/
↪releases/download/vX.Y.Z/performancecopilot-pcp-app-X.Y.Z.zip;performancecopilot-
↪pcp-app" -p 3000:3000 grafana/grafana
```

```
$ docker run -e GF_INSTALL_PLUGINS="https://github.com/performancecopilot/grafana-pcp/
↪releases/download/vX.Y.Z/performancecopilot-pcp-app-X.Y.Z.zip;performancecopilot-
↪pcp-app" -p 3000:3000 grafana/grafana
```

2.2.4 From Source

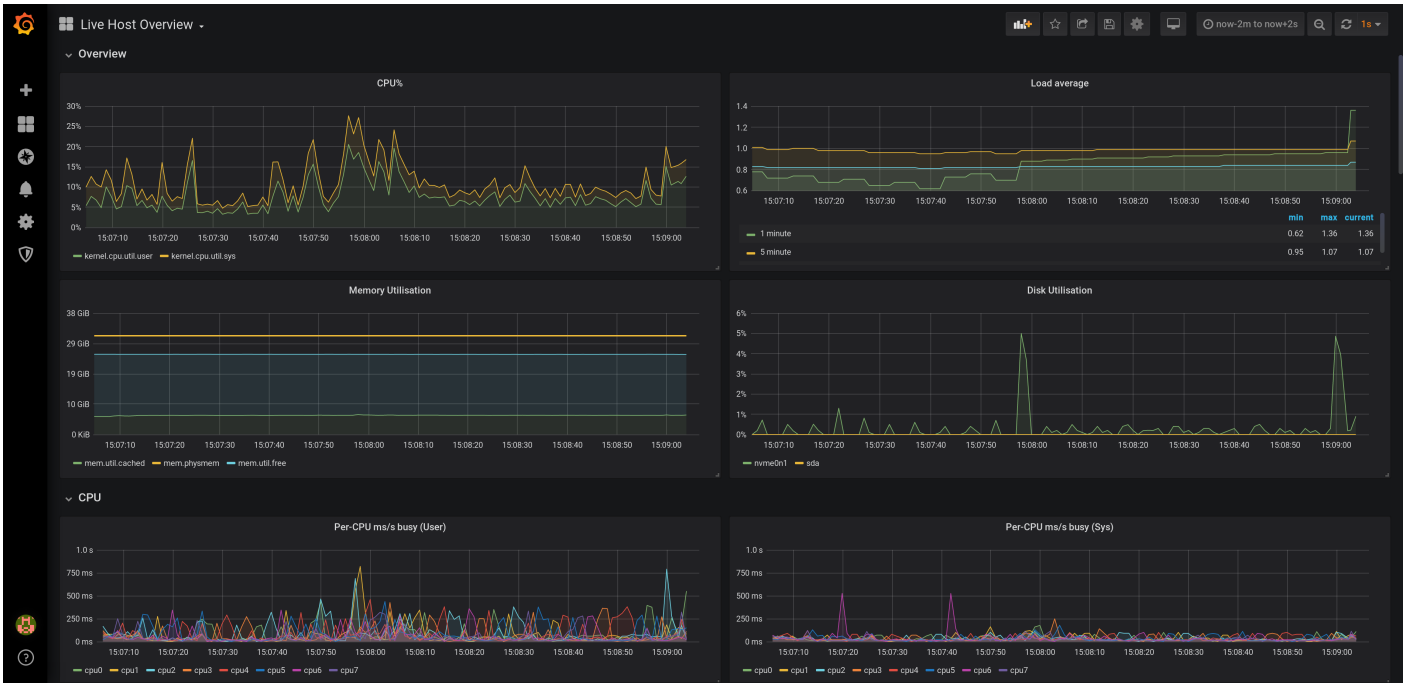
The yarn package manager, Go compiler, jsonnet and jsonnet bundler are required to build grafana-pcp.

```
$ git clone https://github.com/performancecopilot/grafana-pcp.git
$ make dist
$ sudo ln -s $(pwd) /var/lib/grafana/plugins
$ sudo systemctl restart grafana-server
```

To list all available Makefile targets, run `make help`.

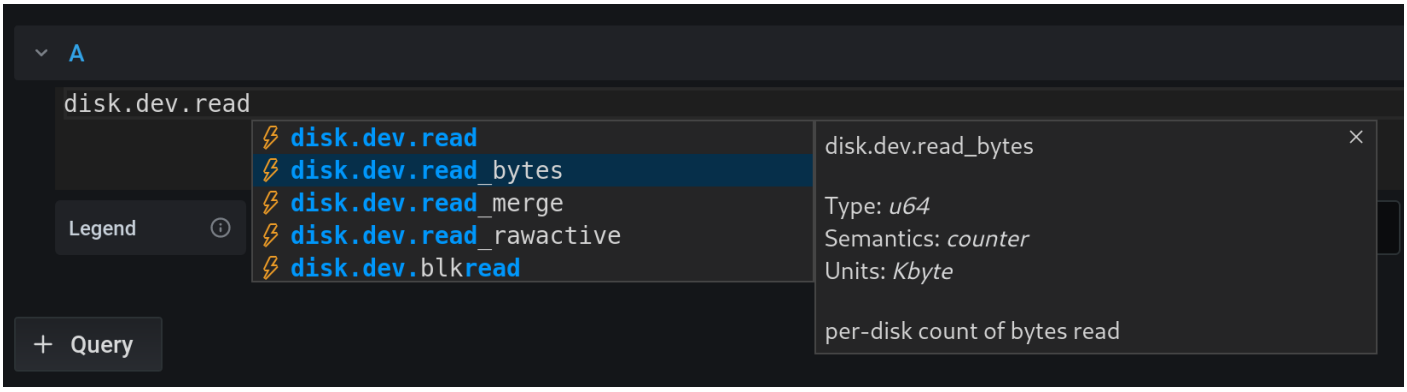
2.3 Screenshots

2.3.1 PCP Vector



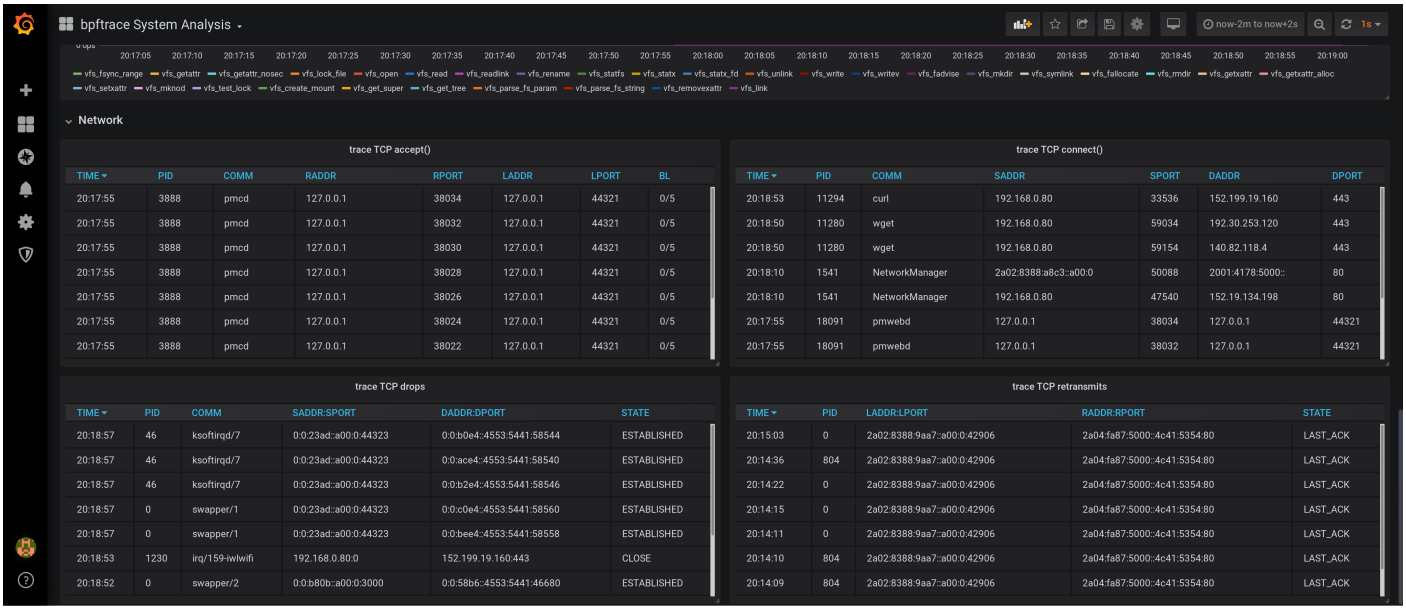

PCP Vector: Container Overview (CGroups v2)
container
container-app1 + container-app2
container-app1
container-app2
CPU
Memory
Disk IOPS


PCP Vector Checklist
Overview
Choose
CPU Utilization [%]
Storage Utilization [%]
Memory Utilization [%]
Network TX Utilization [%]
Network RX Utilization [%]

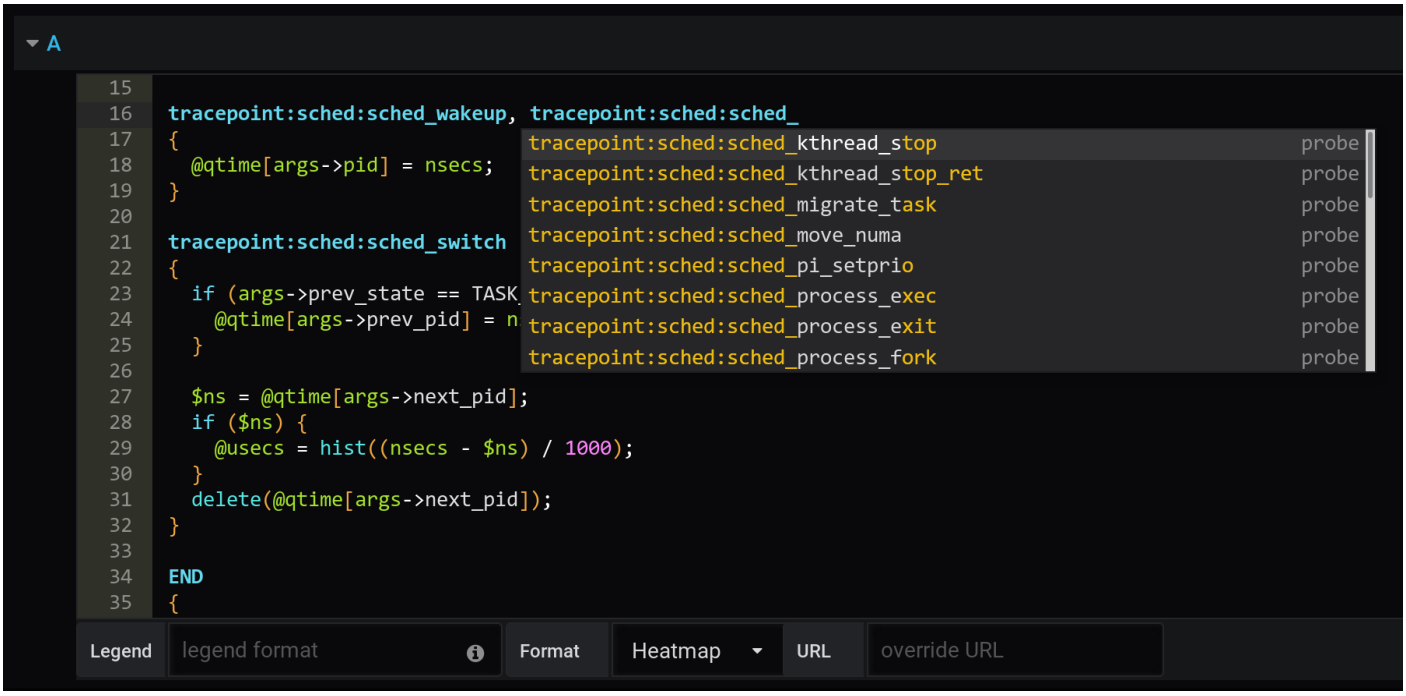


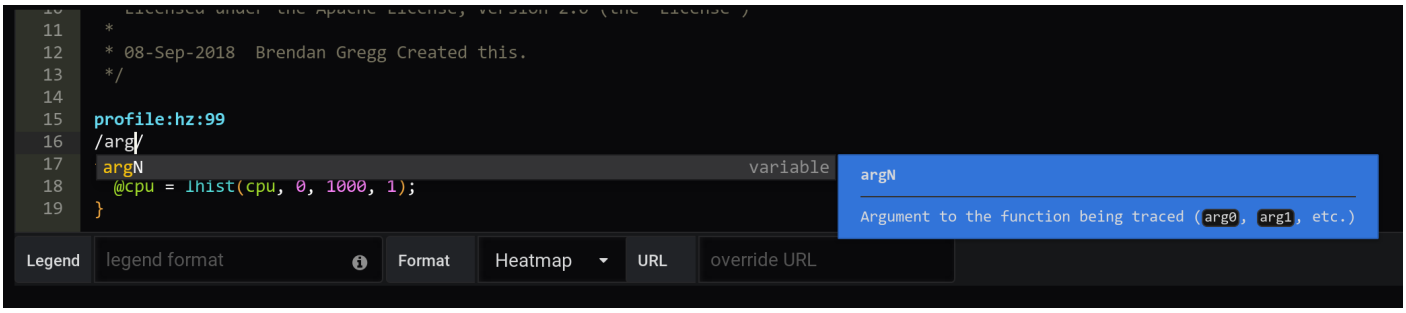
2.3.2 PCP bpftrace

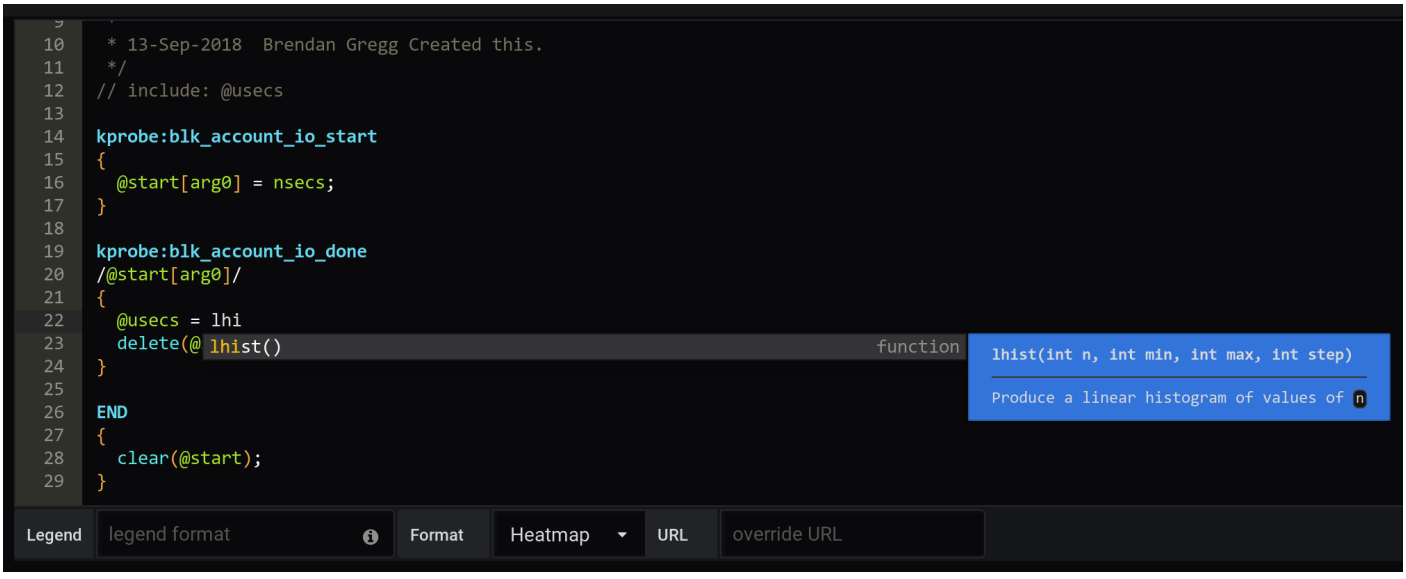

bpftrace System Analysis
Disk
Block I/O latency
66 ms
33 ms
16 ms
8 ms
4 ms
2 ms
1 ms
511 µs
255 µs
127 µs
63 µs
31 µs
15 µs
20:05:20 20:05:25 20:05:30 20:05:35 20:05:40 20:05:45 20:05:50 20:05:55 20:06:00 20:06:05 20:06:10 20:06:15 20:06:20 20:06:25 20:06:30 20:06:35 20:06:40 20:06:45 20:06:50 20:06:55 20:07:00 20:07:05 20:07:10 20:07:15
Filesystem
virtual file system calls
40K ops
30K ops
20K ops
10K ops
0 ops
vfs_fsync_range vfs_getattr vfs_getattr_nosec vfs_lock_file vfs_open vfs_read vfs_readlink vfs_rename vfs_statfs vfs_statx vfs_statx_fd vfs_unlink vfs_write vfs_writev vfs_fadvise vfs_mkdir vfs_symlink vfs_fallocate vfs_rmdir
Network


bpftrace System Analysis
vfs_fsync_range vfs_getattr vfs_getattr_nosec vfs_lock_file vfs_open vfs_read vfs_readlink vfs_rename vfs_statfs vfs_statx vfs_statx_fd vfs_unlink vfs_write vfs_writev vfs_fadvise vfs_mkdir vfs_symlink vfs_fallocate vfs_rmdir vfs_getxattr vfs_getxattr_alloc
vfs_setxattr vfs_mknod vfs_test_lock vfs_create_mount vfs_get_super vfs_get_tree vfs_parse_fs_param vfs_parse_fs_string vfs_removexattr vfs_link
Network
trace TCP accept()
TIME PID COMM RADDR RPORT LADDR LPORT BL
20:17:55 3888 pmcd 127.0.0.1 38034 127.0.0.1 44321 0/5
20:17:55 3888 pmcd 127.0.0.1 38032 127.0.0.1 44321 0/5
20:17:55 3888 pmcd 127.0.0.1 38030 127.0.0.1 44321 0/5
20:17:55 3888 pmcd 127.0.0.1 38028 127.0.0.1 44321 0/5
20:17:55 3888 pmcd 127.0.0.1 38026 127.0.0.1 44321 0/5
20:17:55 3888 pmcd 127.0.0.1 38024 127.0.0.1 44321 0/5
20:17:55 3888 pmcd 127.0.0.1 38022 127.0.0.1 44321 0/5
trace TCP connect()
TIME PID COMM SADDR SPORT DADDR DPORT
20:18:53 11294 curl 192.168.0.80 33536 152.199.19.160 443
20:18:50 11280 wget 192.168.0.80 59034 192.30.253.120 443
20:18:50 11280 wget 192.168.0.80 59154 140.82.118.4 443
20:18:10 1541 NetworkManager 2a02:8388:a8c3::a00:0 50088 2001:4178:5000:: 80
20:18:10 1541 NetworkManager 192.168.0.80 47540 152.19.134.198 80
20:17:55 18091 pmwebd 127.0.0.1 38034 127.0.0.1 44321
20:17:55 18091 pmwebd 127.0.0.1 38032 127.0.0.1 44321
trace TCP drops
TIME PID COMM SADDR:SPORT DADDR:DPORT STATE
20:18:57 46 ksoftirqd/7 0:0:23ad::a00:0:44323 0:0:b0e4::4553:5441:58544 ESTABLISHED
20:18:57 46 ksoftirqd/7 0:0:23ad::a00:0:44323 0:0:ace4::4553:5441:58540 ESTABLISHED
20:18:57 46 ksoftirqd/7 0:0:23ad::a00:0:44323 0:0:b2e4::4553:5441:58546 ESTABLISHED
20:18:57 0 swapper/1 0:0:23ad::a00:0:44323 0:0:c0e4::4553:5441:58560 ESTABLISHED
20:18:57 0 swapper/1 0:0:23ad::a00:0:44323 0:0:bee4::4553:5441:58558 ESTABLISHED
20:18:53 1230 irq/159-iwlwifi 192.168.0.80:0 152.199.19.160:443 CLOSE
20:18:52 0 swapper/2 0:0:b80b::a00:0:3000 0:0:58b6::4553:5441:46680 ESTABLISHED
trace TCP retransmits
TIME PID LADDR:LPORT RADDR:RPORT STATE
20:15:03 0 2a02:8388:9aa7::a00:0:42906 2a04:fa87:5000::4c41:5354:80 LAST_ACK
20:14:36 804 2a02:8388:9aa7::a00:0:42906 2a04:fa87:5000::4c41:5354:80 LAST_ACK
20:14:22 0 2a02:8388:9aa7::a00:0:42906 2a04:fa87:5000::4c41:5354:80 LAST_ACK
20:14:15 0 2a02:8388:9aa7::a00:0:42906 2a04:fa87:5000::4c41:5354:80 LAST_ACK
20:14:11 0 2a02:8388:9aa7::a00:0:42906 2a04:fa87:5000::4c41:5354:80 LAST_ACK
20:14:10 804 2a02:8388:9aa7::a00:0:42906 2a04:fa87:5000::4c41:5354:80 LAST_ACK
20:14:09 804 2a02:8388:9aa7::a00:0:42906 2a04:fa87:5000::4c41:5354:80 LAST_ACK

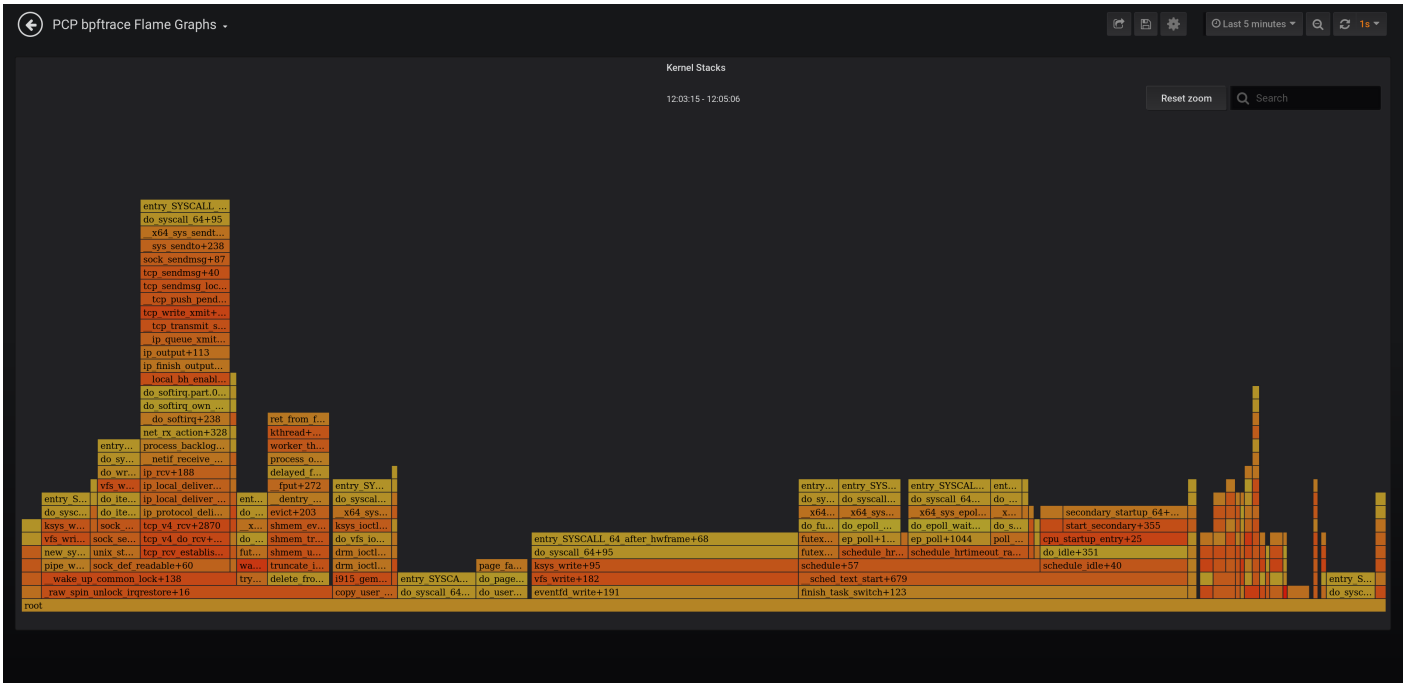
bpftrace code editor



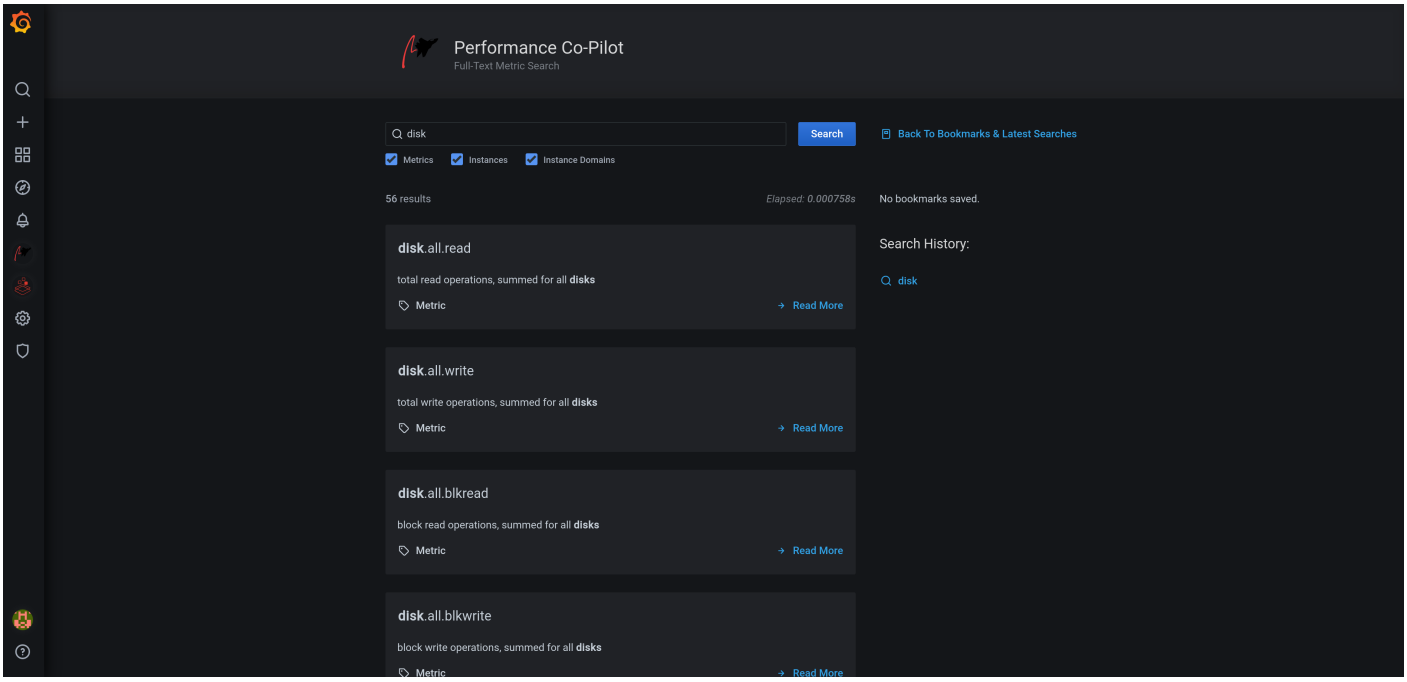




bpftrace flame graphs



Metric Search



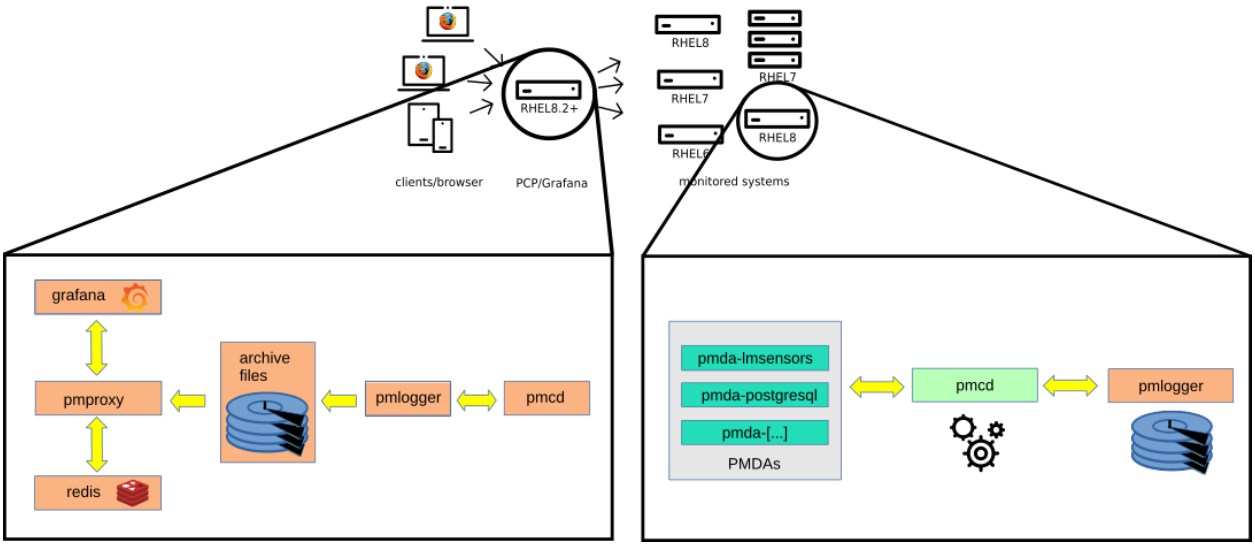


Fig. 1: © Christian Horn

2.4 Architecture

2.4.1 Monitored Hosts

Monitored hosts run the **Performance Metrics Collector Daemon (PMCD)**, which communicates with one or many **Performance Metrics Domain Agents (PMDAs)** on the same host. Each **PMDA** is responsible for gathering metrics of one specific domain - e.g., the kernel, services (e.g., PostgreSQL), or other instrumented applications. The **pmlogger** daemon records metrics from **pmcd** and stores them in archive files on the hard drive.

Since **PCP 5** metrics can also be stored in the redis database, which allows multi-host performance analysis, the **pmproxy** daemon discovers new archives (created by **pmlogger**) and stores them in a redis database.

2.4.2 Dashboards

Performance Co-Pilot metrics can be analyzed with Grafana dashboards, using the **grafana-pcp** plugin. There are two modes available:

- historical metrics across multiple hosts using the *PCP Redis* datasource
- live, on-host metrics using the *PCP Vector* datasource

The *PCP Redis* datasource sends *pmseries* queries to **pmproxy**, which in turn queries the redis database for metrics. The *PCP Vector* datasource connects to **pmproxy**, which in turn requests live metrics directly from a local or remote **PMCD**. In this case, metrics are stored temporarily in the browser, and metric values are lost when the browser tab is refreshed. The *PCP Redis datasource* is required for persistence.

2.5 Change Log

2.5.1 3.0.3 (2021-02-24)

- **test**: fix e2e tests by using another CSS selector

- **chore**: update dependencies
- **docs**: add container guide and screenshot

2.5.2 3.0.2 (2021-01-22)

- **checklist**: replace the storage metrics `disk.dm.*` with `disk.dev.*` (enables usage without device mapper)

2.5.3 3.0.1 (2020-12-22)

Enhancements / Bug Fixes

- **redis**: add auto-completions for new pmseries(1) language functions
- **redis, vector**: show error messages returned by the REST API
- **vector, bpftrace**: fix error messages regarding missing metrics
- **vector**: register derived metrics for every context
- **vector**: handle missing metric metadata responses
- **checklist**: fix metric name in storage warning dialog
- **test**: fix PCP Redis datasource test on 32bit architectures
- **build**: update dependencies

2.5.4 3.0.0 (2020-11-23)

Highlights of v3.0

- **redis**: support for Grafana Alerting
- **redis**: full-text search in metric names, descriptions, instances
- **vector**: support derived metrics, which allows the usage of arithmetic operators and statistical functions inside a query (pmRegisterDerived(3))
- **vector**: configurable hostspec (access remote PMCDs through a central pmproxy)
- **vector**: automatically configure the unit of the panel
- **dashboards**: detect potential performance issues and show possible solutions with the checklist dashboards, using the USE method
- **dashboards**: new MS SQL server dashboard (Louis Imershein)
- **dashboards**: new eBPF/BCC dashboard
- **dashboards**: new container overview dashboard with CGroups v2

Breaking Changes in v3.0

- **dashboards**: All dashboards are now located in the *Dashboards* tab at the datasource settings pages and are not imported automatically
- **redis**: Using `label_values(metric, label)` in a Grafana variable query is deprecated due to performance reasons. `label_values(label)` is still supported.

New Features

- **redis**: added instance.name and dashboard variables support in query editor
- **redis**: heatmap support
- **dashboards**: updated PCP Redis Metric Preview dashboards: added metric drop-down
- **dashboards**: added MS SQL server dashboard for Vector (Louis Imershein)
- **chore**: sign plugin

Enhancements / Bug Fixes

- **redis**: implement workaround if two values for the same instance and timestamp are received
- **redis**: send one instance labels request instead of one per instance
- **redis**: refresh instances only once per series
- **redis**: improved error messages
- **vector**: (internal) option to disable time utilization conversion
- **vector**: show error message when access mode is set to server & url override is set
- **vector**: disable redis backfill for now (pmseries and pmapi instance id's don't match)
- **bpftrace**: interpret all fields of CSV output as strings
- **dashboards**: moved dashboards to the datasource level: dashboards of interest can be imported using the dashboards tab of each datasource settings page
- **dashboards**: fix KB/s unit in dashboards, should be KiB/s
- **dashboards**: add installation instructions to BCC and bpftrace dashboards
- **dashboards**: update titles and add units to checklist dashboards
- **search**: fix datasource detection
- **search**: propagate error messages to the user
- **poller**: use timeout instead of interval to prevent overlapping timers
- **poller**: deregister targets immediately if endpoint changed
- **chore**: update build dependencies
- **test**: add unit tests to all datasources
- **test**: add End-to-End tests
- **docs**: update authentication guide to use scram-sha-256

2.5.5 3.0.0-beta1 (2020-10-12)

New Features

- **redis**: support for Grafana Alerting
- **redis**: full-text search in metric names, descriptions, instances
- **vector**: support derived metrics, which allows the usage of arithmetic operators and statistical functions inside a query, see pmRegisterDerived(3)

- **vector**: set background metric poll interval according to current dashboard refresh interval, do not stop polling while in background
- **vector**: automatically configure the unit of the panel
- **vector**: redis backfilling: if redis is available, initialize the graph with historical data
- **vector**: configurable hostspec (access remote PMCDs through a central pmproxy)
- **vector**: access context, metric, instancedomain and instance labels
- **dashboards**: checklist dashboard: detects potential performance issues and shows possible solutions to resolve them
- **dashboards**: eBPF/BCC dashboard
- **dashboards**: container overview dashboard with CGroups v2

Enhancements / Bug Fixes

- **build**: convert dashboards to jsonnet/grafonnet
- **all**: use latest Grafana UI components based on React (Grafana previously used Angular)

Redis datasource installation

Unfortunately it is not possible to sign community plugins at the moment. Therefore the PCP Redis datasource plugin needs to be allowed explicitely in the Grafana configuration file:

```
allow_loading_unsigned_plugins = pcp-redis-datasource
```

Restart Grafana server, and check the logs if the plugin loaded successfully.

Deprecated features

- **redis**: Using `label_values(metric, label)` in a Grafana variable query is deprecated due to performance reasons. `label_values(label)` is still supported.

2.5.6 2.0.2 (2020-02-25)

- **vector, redis**: remove autocompletion cache (PCP metrics can be added and removed dynamically)

2.5.7 2.0.1 (2020-02-17)

- **build**: fix production build (implement workaround for systemjs/systemjs#2117, grafana/grafana#21785)

2.5.8 2.0.0 (2020-02-17)

- **vector, bpftrace**: fix version checks on dashboard load (prevent multiple pmcd.version checks on dashboard load)
- **vector, bpftrace**: change datasource check box to red if URL is inaccessible
- **redis:** add tests

- **flame graphs:** support multidimensional eBPF maps (required to display e.g. the process name)
- **dashboards**: remove BCC metrics from Vector host overview (because the BCC PMDA isn't installed by default)
- **misc**: update dependencies

2.5.9 2.0.0-beta1 (2019-12-12)

- support Grafana 6.5+, drop support for Grafana < 6.5

2.5.10 1.0.7 (2020-01-29)

- **redis:** fix timespec (fixes empty graphs for large time ranges)

2.5.11 1.0.6 (2020-01-07)

- **redis:** support wildcards in metric names (e.g. `disk.dev.*`)
- **redis:** fix label support
- **redis:** fix legends

2.5.12 1.0.5 (2019-12-16)

- **redis:** set default sample interval to `60s` (fixes empty graph borders)
- **build:** upgrade `copy-webpack-plugin` to mitigate XSS vulnerability in the `serialize-javascript` transitive dependency
- **build:** remove deprecated `uglify-webpack-plugin`

2.5.13 1.0.4 (2019-12-11)

Enhancements

- **flame graphs:** clean flame graph stacks every 5s (reduces CPU load)
- **general:** implement PCP version checks

Bug Fixes

- **build:** remove `weak` dependency (doesn't work with Node.js 12)
- **build:** upgrade `terser-webpack-plugin` to mitigate XSS vulnerability in the `serialize-javascript` transitive dependency

2.5.14 1.0.3 (2019-11-22)

- fix flame graph dependency (`flamegraph.destroy` error in javascript console)

2.5.15 1.0.2 (2019-11-12)

- handle counter wraps (overflows)
- convert time based counters to time utilization

2.5.16 1.0.1 (2019-10-24)

Flame Graphs

- aggregate stack counts by selected time range in the Grafana UI
- add an option to hide idle stacks

Vector

- fix container dropdown in the query editor
- remove container setting from the datasource settings page

Redis

- fix value transformations (e.g., rate conversion of counters)

All

- request more datapoints from the datasource to fill the borders of the graph panel

2.5.17 1.0.0 (2019-10-11)

bpftrace

- support for Flame Graphs
- context-sensitive auto-completion for bpftrace probes, builtin variables, and functions incl. help texts
- parse the output of bpftrace scripts (e.g., using `printf()`) as CSV and display it in the Grafana table panel
- sample dashboards (BPFtrace System Analysis, BPFtrace Flame Graphs)

Vector

- table output: show instance name in the left column
- table output: support non-matching instance names (cells of metrics which don't have the specific instance will be blank)

Vector & bpftrace

- if the metric/script gets changed in the query editor, immediately stop polling the old metric/deregister the old script
- improve pmwebd compatibility

miscellaneous

- help texts for all datasources (visible with the **[?]** button in the query editor)
- renamed PCP Live to PCP Vector
- logos for all datasources
- improved error handling

2.5.18 0.0.7 (2019-08-16)

- The initial release of grafana-pcp

Features

- retrieval of Performance Co-Pilot metrics from pmseries (PCP Redis), pmproxy, and pmwebd (PCP Live)
- automatic rate conversion of counter metrics
- auto-completion of metric names 1,2, qualifier keys, and values 2
- display of semantics, units, and help texts of metrics 1
- legend templating support with `$metric`, `$metric0`, `$instance`, `$some_label`
- container support
- support for repeating panels
- support for custom endpoint URL and container setting per query, with templating support 1
- heatmap and table support 1
- sample dashboards for PCP Redis and PCP Live

1 PCP Live 2 PCP Redis

Known Bugs

- the bpftrace datasource is work-in-progress and will be ready with the next release (approx. 1-2 weeks)

Thanks to Jason Koch for the initial pcp-live datasource implementation and the host overview dashboard.

2.6 Overview

2.6.1 PCP Redis

This data source queries the fast, scalable time series capabilities provided by the pmseries functionality. It is intended to query **historical** data across **multiple hosts** and supports filtering based on labels.

2.6.2 PCP Vector

The PCP Vector data source shows **live, on-host metrics** from the real-time pmwebapi interfaces. It is intended for an individual host, on-demand performance monitoring, and includes container support.

2.6.3 PCP bpftrace

The PCP bpftrace data source supports system introspection using bpftrace scripts. It connects to the bpftrace PMDA and runs bpftrace scripts on the host.

2.7 Authentication

Performance Co-Pilot supports the following authentication mechanisms through the SASL authentication framework: `plain`, `login`, `digest-md5`, `scram-sha-256` and `gssapi`. This guide shows how to setup authentication using the `scram-sha-256` authentication mechanism and a local user database.

Note: Authentication methods `login`, `digest-md5` and `scram-sha-256` require PCP 5.1.0 or later.

2.7.1 Requisites

Install the following package, which provides support for the `scram-sha-256` authentication method:

```
$ sudo dnf install -y cyrus-sasl-scram
```

2.7.2 Configuring PMCD

First, open the `/etc/sasl2/pmcd.conf` file and specify the supported authentication mechanism and the path to the user database:

```
mech_list: scram-sha-256
sasldb_path: /etc/pcp/passwd.db
```

Then create a new unix user (in this example `pcptestuser`) and add it to the user database:

```
$ sudo useradd -r pcptestuser
$ sudo saslpasswd2 -a pmcd pcptestuser
```

Note: For every user in the user database, a unix user with the same name must exist. The passwords of the unix user and the `/etc/pcp/passwd.db` database are not synchronized, and (only) the password of the `saslpasswd2` command is used for authentication.

Make sure that the permissions of the user database are correct (readable only by root and the pcp user):

```
$ sudo chown root:pcp /etc/pcp/passwd.db
$ sudo chmod 640 /etc/pcp/passwd.db
```

Finally, restart pmcd and pmproxy:

```
$ sudo systemctl restart pmcd pmproxy
```

2.7.3 Test Authentication

To test if the authentication is set up correctly, execute the following command:

```
$ pminfo -f -h "pcp://127.0.0.1?username=pcptestuser" disk.dev.read
```

2.7.4 Configuring the Grafana Datasource

Go to the Grafana datasource settings, enable **Basic auth**, and enter the username and password. Click the *Save & Test* button to check if the authentication is working.

Note: Due to security reasons, the access mode *Browser* is **not supported** with authentication.

2.8 PCP Redis

2.8.1 Introduction

This data source provides a native interface between Grafana and Performance Co-Pilot (PCP), allowing PCP metric data to be presented in Grafana panels, such as graphs, tables, heatmaps, etc. Under the hood, the data source makes REST API query requests to the PCP pmproxy service, which can be running either locally or on a remote host. The pmproxy daemon can be local or remote and uses the Redis time-series database (local or remote) for persistent storage.

2.8.2 Setup Redis and PCP daemons

```
$ sudo dnf install redis
$ sudo systemctl start redis pmlogger pmproxy
```

2.8.3 Query Language

Syntax: `[metric.name] '{metadata qualifiers}'`

Examples:

```
kernel.all.load
kernel.all.load{hostname == "web01"}
network.interface.in.bytes{agent == "linux"}
```

Documentation of the pmseries query language can be found in the man page of pmseries.

2.8.4 Query Formats

Time Series

Returns the data as time series. If there are multiple series for a metric, all series will be shown as separate targets (i.e., a line in a line graph). For metrics with instance domains, each instance is shown as a separate target. If there are multiple queries defined, all values will be combined in the same graph.

Table

Transforms the data for the table panel. Two or more queries are required, and it will transform every metric into a column, and every instance into a row. The latest values of the currently selected timeframe will be displayed.

2.8.5 Legend Format Templating

The following variables can be used in the legend format box:

Variable	Description	Example
`$expr`	query expression	`rate(disk.dm.avactive)`
`$metric`	metric name	`disk.dev.read`
`$metric0`	last part of metric name	`read`
`$instance`	instance name	`sda`
`$some_label`	label value	anything

2.8.6 Query Functions

The following functions are available for dashboard variables of type *Query*:

Function	Description	Example
`metrics([pattern])`	returns all metrics matching a glob pattern (if no pattern is defined, all metrics are returned)	`metrics(disk.*)`
`label_names([pattern])`	returns all label names matching a glob pattern (if no pattern is defined, all metrics are returned)	`label_names(host*)`
`label_values(label)`	returns all label values for the specified label	`label_values(hostname)`

2.9 PCP Vector

2.9.1 Query Formats

Time Series

Returns the data as time series. For metrics with instance domains, each instance is shown as a separate target (i.e., line in a line graph). If there are multiple queries defined, all values will be combined in the same graph.

Heatmap

Transforms the data for the heatmap panel. Instance names have to be in the following format: `<lower_bound>-<upper_bound>`, for example, `512-1023` (the bcc PMDA produces histograms in this format).

The following settings have to be set in the heatmap panel options:

Setting	Value
Format	**Time Series Buckets**
Bucket bound	**Upper**

Table

Transforms the data for the table panel. Two or more queries are required, and it will transform every metric into a column, and every instance into a row. The latest values of the currently selected timeframe will be displayed.

2.9.2 Legend Format Templating

The following variables can be used in the legend format box:

Variable	Description	Example
`$expr`	query expression	`rate(disk.dm.avactive)`
`$metric`	metric name	`disk.dev.read`
`$metric0`	last part of metric name	`read`
`$instance`	instance name	`sda`
`$some_label`	label value	anything

2.10 PCP bpftrace

2.10.1 bpftrace PMDA installation

```
$ sudo dnf install pcp-pmda-bpftrace
$ cd /var/lib/pcp/pmdas/bpftrace
$ sudo ./Install
```

2.10.2 Query Formats

Time Series

Shows bpftrace variables as time series. For bpftrace maps, each key is shown as a separate target (i.e. line in a line graph), for example `@counts[comm] = count()`. If there are multiple variables (or scripts) defined, all values will be combined in the same graph.

Heatmap

Transforms bpftrace histograms into heatmaps.

The following settings have to be set in the heatmap panel options:

Setting	Value
Format	**Time Series Buckets**
Bucket bound	**Upper**

Table

Transforms CSV output of bpftrace scripts into a table. The first line must be the column names.

2.10.3 Legend Format Templating

The following variables can be used in the legend format box:

Variable	Description
`$metric0`	bpftrace variable name
`$instance`	bpftrace map key

2.10.4 More Information

bpftrace PMDA README

2.11 Multiple Vector Hosts

In cloud environments, it is often desired to use the Vector datasource to connect to multiple remote hosts without configuring a new data source for each host. This guide shows a setup for this use case using Grafana templates.

2.11.1 Setup the Vector data source

Open the Grafana configuration, go to Data Sources, and add the *PCP Vector* datasource. Leave the URL field empty and select **Access: Browser**. Click the save button. A red alert will appear, with the text *To use this data source, please configure the URL in the query editor.*

2.11.2 Create a new dashboard variable

Create a new dashboard (plus icon in the left navigation - *Create* - *Dashboard*) and open the dashboard settings (wheel icon on the right, top navigation bar). Navigate to *Variables* and create a new variable with the following settings:

Setting	Value
Name	host
Type	Text box

Leave the other fields to their default values. Save the new variable, go back to the dashboard, enter a hostname (for example, `localhost`) in the text box, and press enter.

2.11.3 Create a new graph

Add a new graph to the dashboard, select the *PCP Vector* datasource, enter a PCP metric name (for example `disk.dev.read_bytes`) in the big textbox, and enter `http://$host:44322` in the URL field. If you haven't already, select the time range to *last 5 minutes* and select the auto-refresh interval (top right corner) to 5 seconds, for example.

Now Grafana connects to `http://localhost:44322` for this panel (if you have entered `localhost` in the host textbox). By changing the value of the host text box, you can change the remote host.

2.11.4 Setting the host by query parameter

You can also set the host by an URL query parameter. Add `&var-host=example.com` to the current query, or update the `var-host` query parameter in case it is already present in the current query string.

2.12 Monitoring Containers

2.12.1 Importing the dashboards

grafana-pcp includes the following (optional) dashboards:

- PCP Vector: Container Overview (CGroups v1)
- PCP Vector: Container Overview (CGroups v2)

You can import the corresponding dashboard on the *PCP Vector* datasource settings page.

Note: grafana-pcp before version 3.0.0 includes a single dashboard called **PCP Vector: Container Overview** which supports CGroups v1 only and is installed by default (i.e. no import is required).

2.12.2 Usage

You can choose one or multiple containers in the *container* drop-down field at the top of the dashboard:

2.12.3 Common Problems

My container doesn't show up

- make sure that the docker and/or podman PMDAs are installed
- currently PCP only supports containers started by the root user (there is an open feature request to change this)

2.13 Troubleshooting

2.13.1 Common Problems

When I try to add a datasource in Grafana, I get: "HTTP Error 502: Bad Gateway, please check the datasource and pmproxy settings. To use this data source, please configure the URL in the query editor."

- check if pmproxy is running: `systemctl status pmproxy`
- make sure that pmproxy was built with time-series (libuv) support enabled. You can find out if so in *$PCP_LOG_DIR/pmproxy/pmproxy.log*