Red Hat Enterprise Linux 10

Monitoring and managing system status and performance

Optimizing system throughput, latency, and power consumption

Last Updated: 2025-06-05

Red Hat Enterprise Linux 10 Monitoring and managing system status and performance

Optimizing system throughput, latency, and power consumption

Legal Notice

Abstract

Monitor and optimize the throughput, latency, and power consumption of Red Hat Enterprise Linux 10 in different scenarios.

Table of Contents

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your feedback on our documentation. Let us know how we can improve it.

Submitting feedback through Jira (account required)

1. Log in to the Jira website.
2. Click **Create** in the top navigation bar
3. Enter a descriptive title in the **Summary** field.
4. Enter your suggestion for improvement in the **Description** field. Include links to the relevant parts of the documentation.
5. Click **Create** at the bottom of the dialogue.

CHAPTER 1. OPTIMIZING SYSTEM PERFORMANCE WITH TUNED

TuneD is a system tuning service designed to optimize system performance and power consumption by using predefined or custom profiles. It includes predefined profiles suited for different workloads, such as high throughput, low latency, and power saving.

1.1. TUNED PROFILES DISTRIBUTED WITH RHEL

During installation, TuneD automatically selects the most suitable profile based on the system type. For example, the throughput-performance is selected for compute nodes, virtual-guest is selected for virtual machines, and balanced is selected for general systems, ensuring optimal performance tailored to the environment. Profiles are divided mainly into two categories: power-saving profiles that reduce power consumption with minimal performance impact, and performance-boosting profiles that optimize system resources for improved speed and responsiveness.

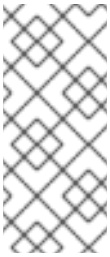
Following is the list of notable profiles distributed with RHEL to select based on the system load:

balanced

It is the default power-saving profile and is intended to be a compromise between performance and power consumption. It uses auto-scaling and auto-tuning whenever possible.

powersave

It is a profile for maximum power saving performance and can throttle the performance to minimize the actual power consumption. It enables USB autosuspend, Wi-Fi power saving, and Aggressive Link Power Management (ALPM) power savings for SATA host adapters, and also schedules multi-core power savings for systems with a low wakeup rate and activates the **ondemand** governor. It enables AC97 audio power saving or, depending on your system, HDA-Intel power savings with a 10 seconds timeout. If your system contains a supported Radeon graphics card with enabled KMS, the profile configures it to automatic power saving. It changes the **energy_performance_preference** attribute to the **powersave** or **power energy** setting. It also changes the **scaling_governor** policy attribute to either the **ondemand** or **powersave** CPU governor.



NOTE

In some cases, the balanced profile is more efficient than powersave. For tasks such as video transcoding, running at full power completes the job faster, allowing the machine to idle and switch to efficient power-saving modes sooner. Throttling the machine reduces power during the task but extends its duration, potentially increasing overall energy use. Thus, the balanced profile is often a better choice.

throughput-performance

A server profile optimized for high throughput that disables power savings mechanisms and enables **sysctl** settings that improve the throughput performance of the disk and network IO.

accelerator-performance

A profile that contains the same tuning as the **throughput-performance** profile. Additionally, it locks the CPU to low C states so that the latency is less than 100us. This improves the performance of certain accelerators, such as GPUs.

latency-performance

A server profile optimized for low latency and disables power savings mechanisms and enables **sysctl** settings that improve latency. CPU governor is set to performance and the CPU is locked to the low C states (by PM QoS).

network-latency

A profile for low latency network tuning. It is based on the **latency-performance** profile. It additionally disables transparent huge pages and NUMA balancing, and tunes several other network-related sysctl parameters.

hpc-compute

A profile optimized for high-performance computing. It is based on the **latency-performance** profile.

network-throughput

A profile for throughput network tuning. It is based on the **throughput-performance** profile. It additionally increases kernel network buffers.

virtual-guest

A profile designed for Red Hat Enterprise Linux virtual machines and VMWare guests based on the **throughput-performance** profile that, among other tasks, decreases virtual memory swappiness and increases disk readahead values. It does not disable disk barriers.

virtual-host

A profile designed for virtual hosts based on the **throughput-performance** profile that, among other tasks, decreases virtual memory swappiness, increases disk readahead values, and enables a more aggressive value of dirty pages writeback.

oracle

A profile optimized for Oracle database loads based on the **throughput-performance** profile. It additionally disables transparent huge pages and modifies other performance-related kernel parameters. This profile is provided by the tuned-profiles-oracle package.

desktop

A profile optimized for desktops, based on the **balanced** profile. It additionally enables scheduler autogroups for better response of interactive applications.

optimize-serial-console

A profile that tunes down I/O activity to the serial console by reducing the printk value. This should make the serial console more responsive. This profile is intended to be used as an overlay on other profiles. For example:

```
# tuned-adm profile throughput-performance optimize-serial-console
```

mssql

A profile provided for Microsoft SQL Server. It is based on the **throughput-performance** profile.

intel-sst

A profile optimized for systems with user-defined **Intel Speed Select Technology** configurations. This profile is intended to be used as an overlay on other profiles. For example:

```
# tuned-adm profile cpu-partitioning intel-sst
```

Additional resources

- **tuned-profiles** man page

1.2. REAL-TIME TUNED PROFILES DISTRIBUTED WITH RHEL

RHEL ships some profiles specifically for systems running the real-time kernel. Without a special kernel build, they do not configure the system to be real-time. On RHEL, the profiles are available with additional repositories.

realtime

Use on bare metal real-time systems. This is provided by the **tuned-profiles-realtime** package, which is available from the RT or NFV repositories.

realtime-virtual-host

Use in a virtualization host configured for real-time. This is provided by the **tuned-profiles-nfv-host** package, which is available from the NFV repository.

realtime-virtual-guest

Use in a virtualization guest configured for real-time. This is provided by the **tuned-profiles-nfv-guest** package, which is available from the NFV repository.

1.3. INSTALLING AND ENABLING TUNED

As a system administrator, you can use the **TuneD** daemon to optimize the performance profile of your system for a variety of use cases. You must install and enable TuneD to start using it.

After installation, the **/usr/lib/tuned/profiles** directory stores distribution-specific profiles. Each profile is stored in its own subdirectory. The profile includes the profile configuration file, named **tuned.conf** and optionally other files, such as helper scripts. Also, to customize a profile, copy the profile directory into **/etc/tuned/profiles**, which stores the files related to the custom profiles. In cases where profiles with the same name exist, the custom profile located in **/etc/tuned/profiles** takes precedence.

Prerequisites

- You have administrative privileges.

Procedure

1. Install the TuneD package:

```
# dnf install tuned
```

2. Enable and start the TuneD service:

```
# systemctl enable --now tuned
```

3. Optional: Install TuneD profiles for real-time systems:
   For the TuneD profiles for real-time systems enable the **rhel-9** repository.

```
# subscription-manager repos --enable=rhel-10-for-x86_64-nfv-beta-rpms
# dnf install tuned-profiles-realtime tuned-profiles-nfv
```

Verification

- Verify that the profile is active and applied:

```
$ tuned-adm active
Current active profile: throughput-performance
$ tuned-adm verify
```

```
Verification succeeded, current system settings match the preset profile. See tuned log file
('/var/log/tuned/tuned.log') for details.
```

Additional resources

- **tuned.conf(5)** man page

1.4. MANAGING TUNED PROFILES

You can manage TuneD profiles in your Red Hat Enterprise Linux system, including listing available profiles, setting a specific profile, and disabling TuneD when necessary. Proper management of profiles optimizes system performance for specific workloads, such as using the cpu-partitioning profile for low-latency applications.

Prerequisites

- You have installed and enabled TuneD.
- You have administrative privileges.

Procedure

1. List all the available predefined TuneD profiles:

```
$ tuned-adm list
```

2. Optional: To let TuneD recommend the most suitable profile for your system, use the following command:

```
# tuned-adm recommend
throughput-performance
```

3. Activate the profile:

```
# tuned-adm profile selected-profile
```

Alternatively, you can activate a combination of multiple profiles:

```
# tuned-adm profile selected-profile1 selected-profile2
```

4. View the current active TuneD profile on your system:

```
# tuned-adm active
Current active profile: selected-profile
```

5. Reboot the system to apply the changes:

```
# reboot
```

6. Disable all tunings temporarily:

```
# tuned-adm off
The tunings are applied again after the TuneD service restarts.
```

7. Optional: Stop and disable the TuneD service permanently:

```
# systemctl disable --now tuned
```

1.5. PROFILE INHERITANCE AND VARIABLES FOR PROFILE CUSTOMIZATION

TuneD supports profile inheritance, variable usage, and built-in functions to create new or modify existing profiles efficiently.

The pre-installed and custom profiles are available in the following directories:

- Pre-installed profiles: Located in **/usr/lib/tuned/profiles/**
- Custom profiles: Created in **/etc/tuned/profiles/**

Profile Inheritance

TuneD profiles can inherit settings from other profiles by using the include option in the [main] section. Profile inheritance means a child profile inherits all settings from its parent but can override or add new parameters based on the custom requirements. For example, to create a profile based on the balanced profile, which additionally sets ALPM to the min_power instead of the medium_power, which is originally set by the balanced profile, use:

```
[main]
include=balanced

[scsi_host]
alpm=min_power
```

Variables' usage while customizing profiles

You can define variables while customizing profiles. Variables help to simplify configurations by reducing repetitive definitions. You can define your own variables by creating the **[variables]** section in a profile and by using:

```
[variables]
variable_name=value
```

To expand the value of a variable in a profile, use:

```
${variable_name}
```

You can also store the variables in different files and use these variables while customizing profiles by adding the filepath to the variable file. For example, you can add the following lines to configuration file to consider variables from the separate file:

```
tuned.conf:
[variables]
include=/etc/tuned/my-variables.conf
```

```
[bootloader]
cmdline=isolcpus=${isolated_cores}
```

Where **isolated_cores=1,2** is added in **my-variable.conf** file as follows:

my-variable.conf:

```
isolated_cores=1,2
```

1.6. BUILT-IN FUNCTIONS FOR PROFILE CUSTOMIZATION

You can use the built-in functions in TuneD profiles to dynamically expand at runtime when a profile is activated. Use built-in functions with TuneD variables to modify and process values dynamically within a profile. Additionally, you can extend TuneD with custom functions by creating and integrating custom Python functions as plugins.

Syntax to start a built-in function:

```
${f:function_name:argument_1:argument_2}
```

Also, to retrieve the directory path where the profile and the tuned.conf file are located, use the **PROFILE_DIR** variable, which requires the following syntax:

```
${i:PROFILE_DIR}
```

Example of isolating CPU cores by using a built-in function:

```
[variables]
non_isolated_cores=0,3-5

[bootloader]
cmdline=isolcpus=${f:cpulist_invert:${non_isolated_cores}}
```

In this example, the **${non_isolated_cores}** variable expands to **0,3-5**. The **cpulist_invert** function inverts the CPU list. On a system with 6 CPUs, **0,3-5** inverts to **1,2**, resulting in the kernel booting with the **isolcpus=1,2** option.

Table 1.1. Available built-in functions:

Function name	Description
assertion	Compare two arguments. If they do not match, the function logs text from the first argument and aborts profile loading.
assertion_non_equal	Compare two arguments. If they match, the function logs text from the first argument and aborts profile loading.

Function name	Description
calc_isolated_cores	Calculates and returns isolated cores. The argument specifies how many cores per socket to reserve for housekeeping. If not specified, one core per socket is reserved for housekeeping and the rest are isolated.
check_net_queue_count	Checks whether the user has specified a queue count for net devices. If not, it returns the number of housekeeping CPUs.
cpuinfo_check	Checks regular expressions against /proc/cpuinfo. Accepts arguments in the form: REGEX1, STR1, REGEX2, STR2, ...[, STR_FALLBACK]. If REGEX1 matches something in /proc/cpuinfo, it expands to STR1; if REGEX2 matches, it expands to STR2. It stops on the first match. If no regular expression matches, it expands to STR_FALLBACK or an empty string if no fallback is provided.
cpulist2devs	Converts a CPU list to device strings.
cpulist2hex	Converts a CPU list to a hexadecimal CPU mask.
cpulist2hex_invert	Converts a CPU list to a hexadecimal CPU mask and inverts it.
cpulist_invert	Inverts a list of CPUs to make its complement. For example, on a system with 4 CPUs (0-3), the inversion of the list 0,2,3 is 1.
cpulist_online	Checks whether the CPUs from the list are online. Returns the list containing only online CPUs.
cpulist_pack	Packs a CPU list in the form of 1,2,3,5 to 1-3,5.
cpulist_present	Checks whether the CPUs from the list are present. Returns the list containing only present CPUs.
cpulist_unpack	Unpacks a CPU list in the form of 1-3,4 to 1,2,3,4.
exec	Executes a process and returns its output.
hex2cpulist	Converts a hexadecimal CPU mask to a CPU list.

Function name	Description
intel_recommended_pstate	Checks the processor code name and returns the recommended **intel_pstate** CPUFreq driver mode. "Active" is returned for newer generation processors not in the **PROCESSOR_NAME** list.
iscpu_check	Checks regular expressions against the output of **iscpu**. Accepts arguments in the form: REGEX1, STR1, REGEX2, STR2, ...[, STR_FALLBACK]. If REGEX1 matches something in the output, it expands to STR1; if REGEX2 matches, it expands to STR2. It stops on the first match. If no regular expression matches, it expands to **STR_FALLBACK** or an empty string if no fallback is provided.
package2cpus	Provides a CPU device list for a package (socket).
package2uncores	Provides an uncore device list for a package (socket).
regex_search_ternary	Ternary regular expression operator. Takes arguments in the form: STR1, REGEX, STR2, STR3. If REGEX matches STR1 (**re.search** is used), STR2 is returned; otherwise, STR3 is returned.
log	Expands to the concatenation of arguments and logs the result, useful for debugging.
kb2s	Converts KB to disk sectors.
s2kb	Converts disk sectors to KB.
strip	Creates a string from all passed arguments and deletes both leading and trailing white spaces.
virt_check	Checks whether TuneD is running inside a virtual machine (VM) or on bare metal. Inside a VM, the function returns the first argument. On bare metal, the function returns the second argument, even in case of an error.

1.7. TUNED PLUGINS

TuneD profiles use plugins to monitor or optimize different devices on the system. TuneD uses two types of plugins:

Monitoring plugins

Used for gathering system data such as, CPU load, disk I/O, and network traffic. The output of the monitoring plugins can be used by tuning plugins for dynamic tuning. Monitoring plugins are

automatically instantiated whenever their metrics are needed by any of the enabled tuning plugins. The available monitoring plugins are:

disk

Gets disk load (number of IO operations) per device and measurement interval.

net

Gets network load (number of transferred packets) per network card and measurement interval.

load

Gets CPU load per CPU and measurement interval.

Tuning plugins

Each tuning plugin tunes an individual subsystem and takes several parameters that are populated from the TuneD profiles. Each subsystem can have multiple devices, such as multiple CPUs or network cards, handled by individual instances of the tuning plugins. Specific settings for individual devices are also supported. The available tuning plugins are:

acpi

Configures the ACPI driver. Use the **platform_profile** option to set the ACPI platform profile **sysfs** attribute. It is a generic **power**/**performance** preference API for other drivers. Multiple profiles can be specified, separated by |. The first available profile is selected.

audio

Sets the autosuspend timeout for audio codecs to the value specified by the timeout option. Currently, the **snd_hda_intel** and **snd_ac97_codec** codecs are supported. The value 0 means that autosuspend is disabled. You can also enforce the controller reset by setting the Boolean option reset_controller to **true**.

bootloader

Adds options to the kernel command line. This plugin supports only the GRUB boot loader. Customized non-standard location of the GRUB configuration file can be specified by the **grub2_cfg_file** option. The kernel options are added to the current GRUB configuration and its templates. The system needs to be rebooted for the kernel options to take effect.

cpu

Manages CPU governor and power settings by setting the CPU governor to the value specified by the governor option and dynamically changing the Power Management Quality of Service (PM QoS) CPU Direct Memory Access (DMA) latency according to the CPU load.

disk

Manages disk settings such as **apm**, **scheduler_quantum**, **readahead**, **readahead_multiply**, **spindown**.

eeepc_she

Dynamically sets the front-side bus (FSB) speed according to the CPU load.

irq

Handles individual interrupt requests (IRQ) as devices and multiple plugin instances can be defined, each addressing different devices or irqs. The device names used by the plugin are **irq<n>**, where **<n>** is the IRQ number. The special device **DEFAULT** controls values written to **/proc/irq/default_smp_affinity**, which applies to all non-active IRQs.

irqbalance

Manages settings for **irqbalance**. The plugin configures CPUs which should be skipped when rebalancing IRQs in **/etc/sysconfig/irqbalance**. It then restarts irqbalance only if it was previously running.

modules

Applies custom kernel modules options. It can set parameters to kernel modules and creates **/etc/modprobe.d/tuned.conf** file. The syntax is ***module=option1=value1 option2=value2...*** where ***module*** is the module name and ***optionx=valuex*** are module options which might be present.

mounts

Enables or disables barriers for mounted filesystems.

net

Configures Wake-on-LAN and interface speed by using the same syntax as the ethtool utility. Also, dynamically changes the interface speed according to the interface utilization.

rtentsk

Avoids inter-processor interruptions caused by enabling or disabling static keys. It has no options. When included, **TuneD** keeps an open socket with timestamping enabled, thus keeping the static key enabled.

selinux

Tunes SELinux options. SELinux decisions, such as allowing or denying access, are cached. This cache is known as the Access Vector Cache (AVC). When using these cached decisions, SELinux policy rules need to be checked less, which increases performance. The **avc_cache_threshold** option allows adjusting the maximum number of AVC entries.

systemd

Tunes systemd options. The **cpu_affinity** option allows setting CPUAffinity in **/etc/systemd/system.conf**. This configures the CPU affinity for the service manager and the default CPU affinity for all forked off processes. Add a comma-separated list of CPUs with optional CPU ranges specified by the minus sign (**-**).

scsi_host

Tunes SCSI host settings (for example, **ALPM**).

scheduler

Offers a variety of options for the tuning of scheduling priorities, CPU core isolation, and process, thread, and IRQ affinities.

script

Runs external script or binary when loading or unloading a profile.

service

Handles various sysvinit, sysv-rc, openrc, and systemd services specified by the plugin options. Supported service handling commands are start, stop, enable, and disable.

sysctl

Modifies kernel parameters. Use this plugin only if you want to change system settings that are not covered by other plugins available in TuneD. The syntax is *name=value*, where *name* is the same as the name provided by the **sysctl** utility.

sysfs

Sets various **sysfs** settings specified by the plugin options. The syntax is *name=value*, where *name* is the **sysfs** path to use.

usb

Adjusts USB autosuspend timeout. The value **0** means that autosuspend is disabled.

uncore

Limits the maximum and minimum uncore frequency. The options **max_freq_khz**, **min_freq_khz** correspond to **sysfs** files exposed by Intel uncore frequency driver. Their values can be specified in kHz or as a percentage of their configurable range.

video

Sets various powersave levels on video cards. Currently, only the **Radeon** cards are supported. The **powersave** level can be specified by using the **radeon_powersave** option. Supported values are, default, auto, low, mid, high, dynpm, dpm-battery, dpm-balanced, and dpm-perfomance.

vm

Enables or disables transparent huge pages. Valid values for the transparent_hugepages option are: **always, never, madvise**.

Additional resources

- Optimizing CPU Scheduling with the TuneD Scheduler Plugin (Red Hat Knowledgebase)

1.8. CREATING A NEW TUNED PROFILE

You can create a new TuneD profile to customize your requirements for performance optimization.

Prerequisites

- The TuneD service is running. See Installing and Enabling TuneD for details.

Procedure

1. In the /etc/tuned/profiles directory, create a new directory named the same as the profile that you want to create:

```
# mkdir /etc/tuned/profiles/my-profile
```

2. In the new directory, create a file named **tuned.conf**.

3. Edit the file and add a [main] section and plug in definitions in it, according to your requirements. For example, see the configuration of the **balanced** profile:

```
[main]
summary=General non-specialized TuneD profile

[cpu]
governor=conservative
energy_perf_bias=normal

[audio]
timeout=10

[video]
radeon_powersave=dpm-balanced, auto

[scsi_host]
alpm=medium_power
```

4. Activate the profile.

```
# tuned-adm profile my-profile
```

Verification

- View the profile is active and applied:

```
$ tuned-adm active
Current active profile: my-profile
$ tuned-adm verify
Verification succeeded, current system settings match the preset profile. See tuned log file
('/var/log/tuned/tuned.log') for details.
```

1.9. MODIFYING AN EXISTING TUNED PROFILE

You can modify the parameters of an existing profile to suit your custom requirements. A modified child profile can be created based on an existing TuneD profile.

Prerequisites

- The TuneD service is running. See Installing and Enabling TuneD for details.

Procedure

1. In the **/etc/tuned/profiles** directory, create a new directory named the same as the profile that you want to create:

```
# mkdir /etc/tuned/profiles/modified-profile
```

2. In the new directory, create a file named **tuned.conf**, and set the [main] section as follows:

```
[main]
include=parent-profile
```

   Replace parent-profile with the name of the profile you are modifying, for example: **throughput-performance**

3. Include your profile modifications. For example, to lower swappiness in the **throughput-performance** profile, change the value of **vm.swappiness** to **5**, instead of the default **10**, use:

```
[main]
include=throughput-performance

[sysctl]
vm.swappiness=5
```

4. Activate the profile.

```
# tuned-adm profile modified-profile
```

Verification

- View the profile is active and applied:

```
$ tuned-adm active
Current active profile: my-profile
$ tuned-adm verify
```

```
Verification succeeded, current system settings match the preset profile. See tuned log file
('/var/log/tuned/tuned.log') for details.
```

Additional resources

- **tuned.conf(5)** man page

1.10. ENABLING TUNED NO-DAEMON MODE

TuneD can be run in no-daemon mode, which does not require any resident memory. In this mode, TuneD applies the settings and then exits. However, note that this mode disables certain features, including D-Bus support, hot plug support, and rollback functionality for settings.

Prerequisites

- You have root privileges or appropriate permissions to modify the system configuration.

Procedure

1. Edit the **/etc/tuned/tuned-main.conf** file with the following changes:

```
sudo vi /etc/tuned/tuned-main.conf
Add the following line to the configuration file.
daemon = 0
```

2. Save and close the file.

Verification

- To verify the settings, you can check the status of TuneD or view the configuration file again.

```
# tuned-adm active
```

CHAPTER 2. SETTING UP PCP

Performance Co-Pilot (PCP) is a suite of tools, services, and libraries for monitoring, visualizing, storing, and analyzing system-level performance measurements. You can add performance metrics using Python, Perl, C, and C interfaces. Analysis tools can use the Python, C, C client APIs directly, and rich web applications can explore all available performance data using a JSON interface. You can analyze data patterns by comparing live results with archived data.

Features of PCP

- Light-weight distributed architecture useful during the centralized analysis of complex systems.
- Ability to monitor and manage real-time data.
- Ability to log and retrieve historical data.

PCP has the following components

- The Performance Metric Collector Daemon (pmcd) collects performance data from the installed Performance Metric Domain Agents (PMDA). PMDAs can be individually loaded or unloaded on the system and are controlled by the PMCD on the same host.
- Various client tools, such as **pminfo** or **pmstat**, can retrieve, display, archive, and process this data on the same host or over the network.
- The **pcp** and **pcp-system-tools** packages provide the command-line tools and core functionality.
- The **pcp-gui** package provides the graphical application pmchart.
- The **grafana-pcp** package provides powerful web-based visualisation and alerting with Grafana.

2.1. INSTALLING AND ENABLING PCP

Install the required packages and enable the PCP monitoring services to start using it. You can also automate the PCP installation by using the **pcp-zeroconf** package. For more information about installing PCP by using **pcp-zeroconf**, Setting up PCP with pcp-zeroconf.

Procedure

1. Install the pcp package:

```
# yum install pcp
```

2. Enable and start the pmcd service on the host machine:

```
# systemctl enable pmcd
# systemctl start pmcd
```

Verification

- Verify if the PMCD process is running on the host:

```
# pcp
Performance Co-Pilot configuration on arm10.local:

 platform: Linux arm10.local 6.12.0-55.13.1.el10_0.aarch64 #1 SMP PREEMPT_DYNAMIC
Mon May 19 07:29:57 UTC 2025 aarch64
 hardware: 4 cpus, 1 disk, 1 node, 3579MB RAM
 timezone: JST-9
 services: pmcd
    pmcd: Version 6.3.7-1, 12 agents, 6 clients
    pmda: root pmcd proc pmproxy xfs linux nfsclient mmv kvm jbd2
          dm openmetrics
```

Additional resources

- **pmcd(1)** and **pcp(1)** man pages on your system
- System services and tools distributed with PCP

2.2. DEPLOYING A MINIMAL PCP SETUP

The minimal PCP setup collects performance statistics on Red Hat Enterprise Linux. The setup involves adding the minimum number of packages on a production system needed to gather data for further analysis. You can analyze the resulting **tar.gz** file and the archive of the pmlogger output by using various PCP tools and compare them with other sources of performance information.

Prerequisites

- PCP is installed. For more information, see Installing and enabling PCP .

Procedure

1. Update the pmlogger configuration:

```
# pmlogconf -r /var/lib/pcp/config/pmlogger/config.default
```

2. Start the pmcd and pmlogger services:

```
# systemctl start pmcd.service
# systemctl start pmlogger.service
```

3. Execute the required operations to record the performance data.
4. Save the output and save it to a tar.gz file named based on the host name and the current date and time:

```
# cd /var/log/pcp/pmlogger/
# tar -czf $(hostname).$(date +%F-%Hh%M).pcp.tar.gz $(hostname)
```

5. Extract this file and analyze the data using PCP tools.

Additional resources

- **pmlogconf(1)**, **pmlogger(1)**, and **pmcd(1)** man pages on your system

- System services and tools distributed with PCP

2.3. SYSTEM SERVICES AND TOOLS DISTRIBUTED WITH PCP

The basic package **pcp** includes the system services and basic tools. You can install additional tools that are provided with the **pcp-system-tools**, **pcp-gui**, and **pcp-devel** packages.

Roles of system services distributed with PCP

pmcd

The Performance Metric Collector Daemon.

pmie

The Performance Metrics Inference Engine.

pmlogger

The performance metrics logger.

pmproxy

The realtime and historical performance metrics proxy, time series query and REST API service.

Tools distributed with base PCP package

pcp

Displays the current status of a Performance Co-Pilot installation.

pcp-check

Activates, or deactivates core and optional components, such as **pmcd**, **pmlogger**, **pmproxy**, and PMDAs.

pcp-vmstat

Provides a high-level system performance overview every 5 seconds. Displays information about processes, memory, paging, block IO, traps, and CPU activity.

pmconfig

Displays the values of configuration parameters.

pmdiff

Compares the average values for every metric in either one or two archives, in a given time window, for changes that are likely to be of interest when searching for performance regressions.

pmdumplog

Displays control, metadata, index, and state information from a Performance Co-Pilot archive file.

pmfind

Finds PCP services on the network.

pmie

An inference engine that periodically evaluates a set of arithmetic, logical, and rule expressions. The metrics are collected either from a live system, or from a Performance Co-Pilot archive file.

pmieconf

Displays or sets configurable pmie variables.

pmiectl

Manages non-primary instances of pmie.

pminfo

Displays information about performance metrics. The metrics are collected either from a live system, or from a Performance Co-Pilot archive file.

pmlc

Interactively configures active pmlogger instances.

pmlogcheck

Identifies invalid data in a Performance Co-Pilot archive file.

pmlogconf

Creates and modifies a pmlogger configuration file.

pmlogctl

Manages non-primary instances of pmlogger.

pmloglabel

Verifies, modifies, or repairs the label of a Performance Co-Pilot archive file.

pmlogsummary

Calculates statistical information about performance metrics stored in a Performance Co-Pilot archive file.

pmprobe

Determines the availability of performance metrics.

pmsocks

Allows access to a Performance Co-Pilot hosted through a firewall.

pmstat

Periodically displays a brief summary of system performance.

pmstore

Modifies the values of performance metrics.

pmseries

Fast, scalable time series querying, using the facilities of PCP and a distributed key-value data store such as Valkey.

pmtrace

Provides a command-line interface to the trace PMDA.

pmval

An updating display of the current value of any performance metric.

Tools distributed with the separately installed pcp-system-tools package

pcp-atop

Shows the system-level occupation of the most critical hardware resources from the performance point of view: CPU, memory, disk, and network.

pcp-atopsar

Generates a system-level activity report over a variety of system resource utilization. The report is generated from a raw logfile previously recorded using pmlogger or the -w option of pcp-atop.

pcp-dmcache

Displays information about configured Device Mapper Cache targets, such as: device IOPs, cache and metadata device utilization, as well as hit and miss rates and ratios for both reads and writes for each cache device.

pcp-dstat

Displays metrics of one system at a time. To display metrics of multiple systems, use --host option.

pcp-free

Reports on free and used memory in a system.

pcp-htop

Displays all processes running on a system along with their command line arguments in a manner similar to the top command, but allows you to scroll vertically and horizontally as well as interact using a mouse. You can also view processes in a tree format and select and act on multiple processes at once.

pcp-ipcs

Displays information about the inter-process communication (IPC) facilities that the calling process has read access for.

pcp-mpstat

Reports CPU and interrupt-related statistics.

pcp-numastat

Displays NUMA allocation statistics from the kernel memory allocator.

pcp-pidstat

Displays information about individual tasks or processes running on the system, such as CPU percentage, memory and stack usage, scheduling, and priority. Reports live data for the local host by default.

pcp-shping

Samples and reports on the shell-ping service metrics exported by the pmdashping Performance Metrics Domain Agent (PMDA).

pcp-ss

Displays socket statistics collected by the pmdasockets PMDA.

pcp-tapestat

Reports I/O statistics for tape devices.

pcp-uptime

Displays how long the system has been running, how many users are currently logged on, and the system load averages for the past 1, 5, and 15 minutes.

pcp-verify

Inspects various aspects of a Performance Co-Pilot collector installation and reports on whether it is configured correctly for certain modes of operation.

pcp-iostat

Reports I/O statistics for SCSI devices (by default) or device-mapper devices (with the -x device-mapper option).

pmrep

Reports on selected, easily customizable, performance metrics values.

Tools distributed with the separately installed pcp-gui package

pmchart

Plots performance metrics values available through the facilities of the PCP.

pmdumptext

Outputs the values of performance metrics collected live or from a PCP archive.

Tools distributed with the separately installed pcp-devel package

Tools distributed with the separately installed pcp-devel package

pmclient

Displays high-level system performance metrics by using the Performance Metrics Application Programming Interface (PMAPI).

pmdbg

Displays available Performance Co-Pilot debug control flags and their values.

pmerr

Displays available Performance Co-Pilot error codes and their corresponding error messages.

pcp-xsos

Gives a fast summary report for a system by using a single sample taken from either a PCP archive or live metric values from that system.

Other tools distributed as a separate packages

pcp-geolocate

Discovers collector system geographical labels.

pcp2openmetrics

A customizable performance metrics exporter tool from PCP to Open Metrics format. You can select any available performance metric, live or archived, system and application for exporting by using either command line arguments or a configuration file.

2.4. PCP DEPLOYMENT ARCHITECTURES

PCP supports multiple deployment architectures, based on the scale of the PCP deployment, and offers many options to accomplish advanced setups. Available scaling deployment setup variants, determined by sizing factors and configuration options, include the following:

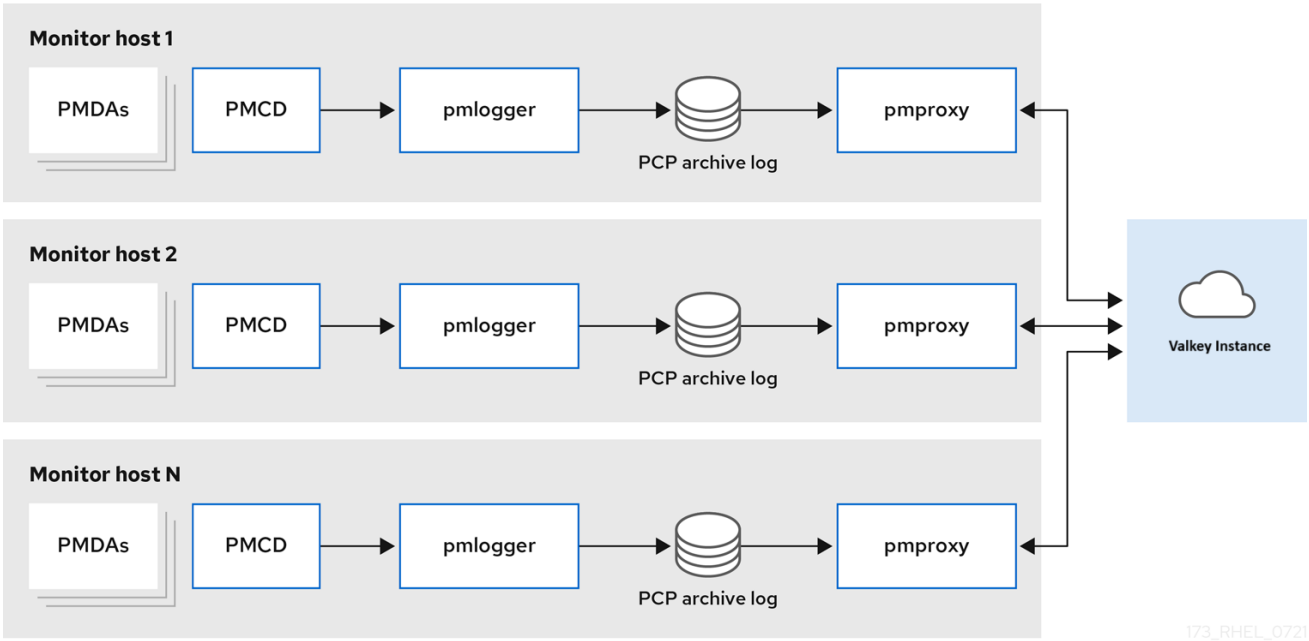
Localhost

Each service runs locally on the monitored machine. Starting a service without any configuration changes results in a default standalone deployment on the localhost. This setup does not support scaling beyond a single node. However, Valkey can also run in a highly available and scalable clustered mode, where data is shared across multiple hosts. You can also deploy a Valkey cluster in the cloud or use a managed Valkey cluster from a cloud provider.

Decentralized

The only difference between localhost and decentralized setup is the centralized Valkey service. In this model, the host executes pmlogger service on each monitored host and retrieves metrics from a local pmcd instance. A local pmproxy service then exports the performance metrics to a central Valkey instance.

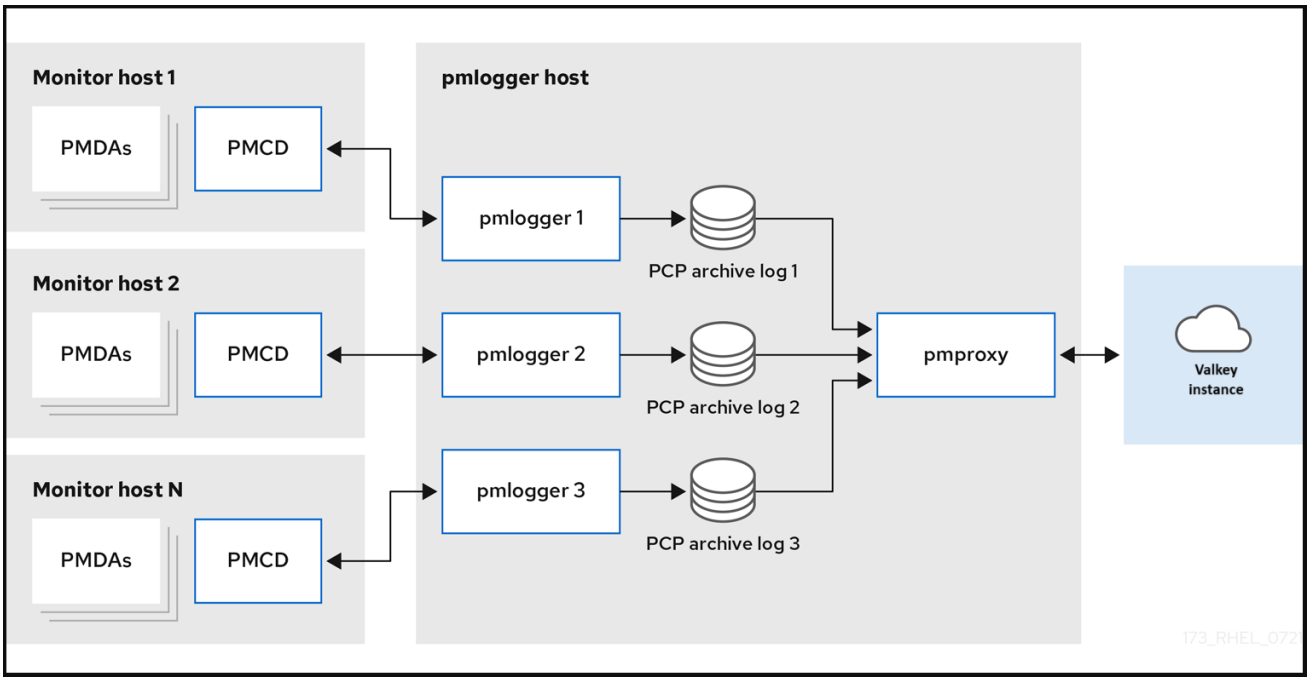
Figure 2.1. Decentralized logging



Centralized logging - pmlogger farm

When the resource usage on the monitored hosts is constrained, another deployment option is a pmlogger farm, which is also known as centralized logging. In this setup, a single logger host executes multiple pmlogger processes, and each is configured to retrieve performance metrics from a different remote pmcd host. The centralized logger host is also configured to execute the pmproxy service, which discovers the resulting PCP archives logs and loads the metric data into a Valkey instance.

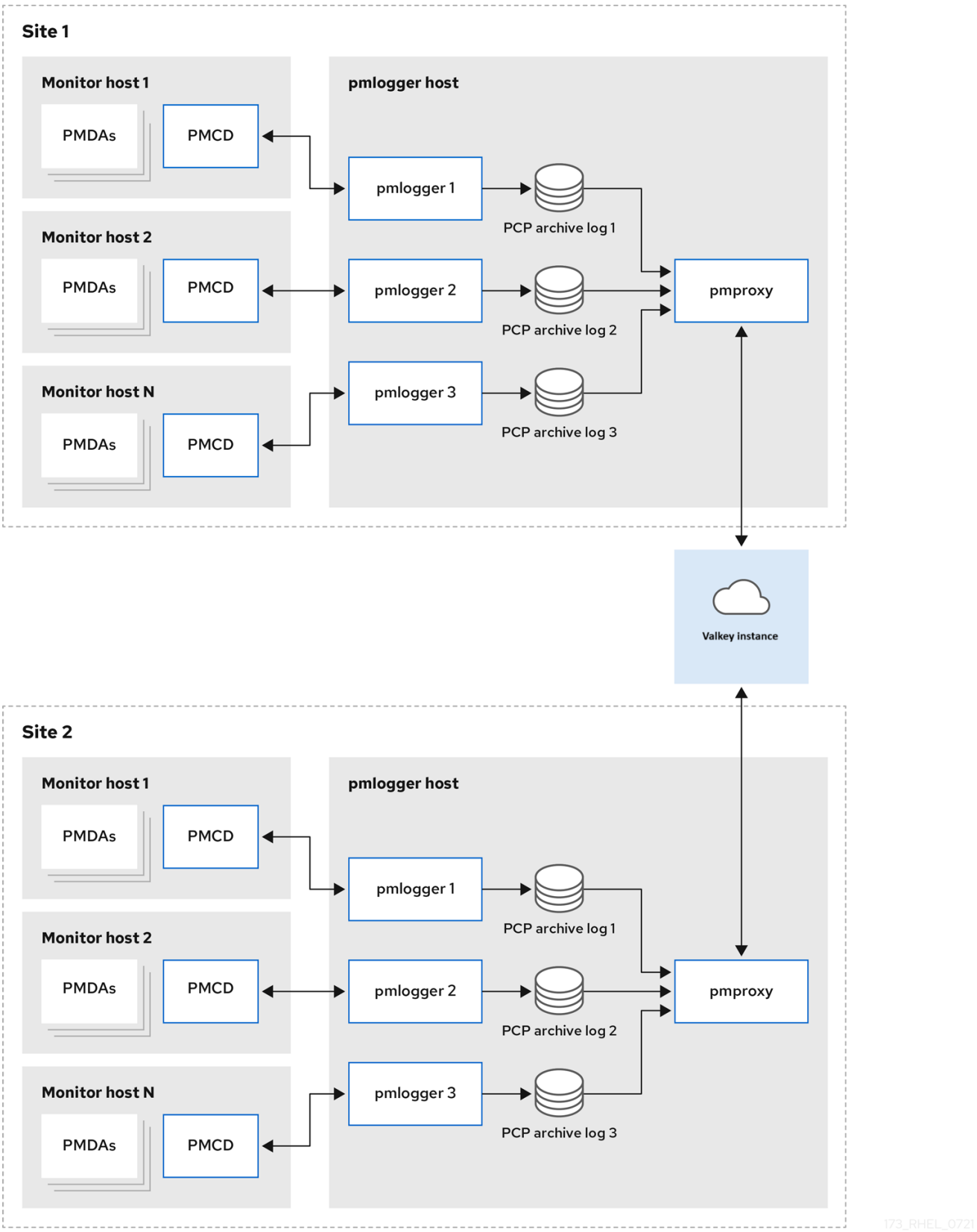
Figure 2.2. Centralized logging - pmlogger farm



Federated - multiple pmlogger farms

For large scale deployments, deploy multiple pmlogger farms in a federated fashion. For example, one pmlogger farm per rack or data center. Each pmlogger farm loads the metrics into a central Valkey instance.

Figure 2.3. Federated - multiple pmlogger farms



NOTE

By default, the deployment setup for Valkey is standalone, localhost. However, Valkey can optionally perform in a highly-available and highly scalable clustered fashion, where data is shared across multiple hosts. Another viable option is to deploy a Valkey cluster in the cloud, or to utilize a managed Valkey cluster from a cloud vendor.

Additional resources

- **pcp(1)**, **pmlogger(1)**, **pmproxy(1)**, and **pmcd(1)** man pages on your system
- Recommended deployment architecture
- Analyzing the centralized logging deployment
- Analyzing the federated setup deployment

2.5. FACTORS AFFECTING SCALING IN PCP LOGGING

The key factors influencing Performance Co-Pilot (PCP) logging are hardware resources, logged metrics, logging intervals, and post-upgrade archive management.

Remote system size

The hardware configuration of the remote system-such as the number of CPUs, disks, and network interfaces-directly impacts the volume of data collected by each **pmlogger** instance on the centralized logging host.

Logged metrics

The number and types of logged metrics significantly affect storage requirements. In particular, the **per-process proc.*** metrics require a large amount of disk space, for example, with the standard pcp-zeroconf setup, 10s logging interval, 11 MB without proc metrics but increases to 155 MB with proc metrics enabled - a ten fold difference. Additionally, the number of instances for each metric, for example the number of CPUs, block devices, and network interfaces also impacts storage capacity needs.

Logging interval

The frequency of metric logging determines storage usage. The expected daily PCP archive file sizes for each **pmlogger** instance are recorded in the **pmlogger.log** file. These estimates represent uncompressed data, but since PCP archives typically achieve a compression ratio of 10:1, long-term disk space requirements can be calculated accordingly.

Managing archive updates with pmlogrewrite

After every PCP upgrade, the pmlogrewrite tool updates existing archives if changes are detected in the metric metadata between versions. The time required for this process scales linearly with the number of stored archives.

Additional resources

- **pmlogrewrite(1)** and **pmlogger(1)** man pages on your system

2.6. ADDITIONAL RESOURCES

- PCPIntro(1) man page on your system
- **/usr/share/doc/pcp-doc/** directory (from the **pcp-doc** RPM package)

- System services and tools distributed with PCP
- Index of Performance Co-Pilot (PCP) articles, solutions, tutorials, and white papers from Red Hat Customer Portal
- Configuring scaling options for performance monitoring
- Troubleshooting high memory usage

CHAPTER 3. CONFIGURING PMDA-OPENMETRICS

Performance Co-Pilot (PCP) is a flexible and extensible system for monitoring and managing system performance. It includes several built-in agents called Performance Metric Domain Agents (PMDAs) that collect metrics from commonly used applications and services, such as PostgreSQL, Apache HTTPD, and KVM virtual machines.

3.1. OVERVIEW OF PMDA-OPENMETRICS

You can run custom or less common applications for which no out-of-the-box PMDA exists in the Red Hat repositories. In such scenarios, the **pmda-openmetrics** agent helps to bridge the gap. The **pmda-openmetrics** PMDA exposes performance metrics from arbitrary applications by converting OpenMetrics-style formatted text files into PCP-compatible metrics. OpenMetrics is a widely adopted format used by Prometheus and other monitoring tools, which makes integration easier.

You can run **pmda-openmetrics** to do the following tasks:

- Monitor custom applications that are not covered by existing PMDAs.
- Integrate existing OpenMetrics or **Prometheus-exported** metrics into the PCP framework.
- Create quick models or test metrics for diagnostic or demonstration purposes.

3.2. INSTALLING AND CONFIGURING PMDA-OPENMETRICS

You must install and configure **pmda-openmetrics** before you start using it. The following example demonstrates how to expose a single numeric value from a text file as a PCP metric by using **pmda-openmetrics**.

Prerequisites

- PCP is installed and **pmcd** is running. For more information, see *installing pcp-zeroconf*.

Procedure

1. Install the **pmda-openmetrics** PMDA.

```
# dnf -y install pcp-pmda-openmetrics
# cd /var/lib/pcp/pmdas/openmetrics/
# ./Install
```

2. Create a sample OpenMetrics file.

```
# echo 'var1 {var2="var3"} 42' > /tmp/example.txt
```

   Replace *example*.txt with the desired file name.

3. Verify that the file is created correctly.

```
# cat /tmp/example.txt
```

4. Register the metric file path with the OpenMetrics agent.

```
# echo "file:///tmp/example.txt" > /etc/pcp/openmetrics/example.url
```

5. Verify that the configuration is created correctly.

```
# cat /etc/pcp/openmetrics/example.url
```

6. Configure **systemd-tmpfiles** to create the necessary symlinks.

```
# echo 'L+ /var/lib/pcp/pmdas/openmetrics/config.d/example.url - - - -
../../../../../../etc/pcp/openmetrics/example.url' \ > /usr/lib/tmpfiles.d/pcp-pmda-openmetrics-
cust.conf
```

7. Verify that the symlinks are configured correctly.

```
# cat /usr/lib/tmpfiles.d/pcp-pmda-openmetrics-cust.conf
```

8. Create the symlinks by applying the tmpfiles configuration.

```
# systemd-tmpfiles --create --remove /usr/lib/tmpfiles.d/pcp-pmda-openmetrics-cust.conf
```

9. Verify that the symlinks are created correctly.

```
# ls -al /var/lib/pcp/pmdas/openmetrics/config.d/
```

10. Verify that the metric is correctly reported.

```
# pminfo -f openmetrics.example.var1
    inst [0 or "0 var2:var3"] value 42
```

Verification

- Run **pcp** and confirm that **openmetrics** is listed.
- Run **systemd-analyze cat-config tmpfiles.d** and confirm that ***example*.url** appears in the output.
- Use **pminfo** to confirm the presence and value of the metric.

CHAPTER 4. LOGGING PERFORMANCE DATA WITH PMLOGGER

With the PCP tool you can log the performance metric values and replay them later. This allows you to perform a retrospective performance analysis. Using the pmlogger tool, you can:

- Create the archived logs of selected metrics on the system
- Specify which metrics are recorded on the system and how often

4.1. MODIFYING THE PMLOGGER CONFIGURATION FILE WITH PMLOGCONF

When the **pmlogger** service is running, PCP logs a default set of metrics on the host. Use the **pmlogconf** utility to check the default configuration. If the **pmlogger** configuration file does not exist, **pmlogconf** creates it with a default metric values.

Prerequisites

- PCP is installed. For more information, see Installing and enabling PCP.

Procedure

1. Create or modify the **pmlogger** configuration file:

```
# pmlogconf -r /var/lib/pcp/config/pmlogger/config.default
```

2. Follow pmlogconf prompts to enable or disable groups of related performance metrics and to control the logging interval for each enabled group. For example,

```
# pmlogconf -r /var/lib/pcp/config/pmlogger/config.default
Group: per logical block device activity
Log this group? [y]
Logging interval? [default]
```

Additional resources

- **pmlogconf(1)** and **pmlogger(1)** man pages on your system
- System services and tools distributed with PCP

4.2. CONFIGURING PMLOGGER MANUALLY

To create a tailored logging configuration with specific metrics and given intervals, edit the **pmlogger** configuration file manually. The default **pmlogger** configuration file is **/var/lib/pcp/config/pmlogger/config.default**. The configuration file specifies which metrics are logged by the primary logging instance.

In manual configuration, you can:

- Record metrics which are not listed in the automatic configuration.
- Choose custom logging frequencies.

- Add **PMDA** with the application metrics.

Prerequisites

- PCP is installed. For more information, see Installing and enabling PCP.

Procedure

- Open and edit the **/var/lib/pcp/config/pmlogger/config.default** file to add specific metrics:

```
# It is safe to make additions from here on ...
#
log mandatory on every 5 seconds {
    xfs.write
    xfs.write_bytes
    xfs.read
    xfs.read_bytes
}
log mandatory on every 10 seconds {
    xfs.allocs
    xfs.block_map
    xfs.transactions
    xfs.log
}
[access]
disallow * : all;
allow localhost : enquire;
```

Additional resources

- **pmlogger(1)** man pages on your system
- System services and tools distributed with PCP

4.3. ENABLING THE PMLOGGER SERVICE

The **pmlogger** service must be started and enabled to log the metric values on the local machine.

Prerequisites

- PCP is installed. For more information, see Installing and enabling PCP.

Procedure

- Start and enable the **pmlogger** service:

```
# systemctl start pmlogger
# systemctl enable pmlogger
```

Verification

- Verify that the **pmlogger** service is enabled:

```
# pcp
 platform: Linux arm10.local 6.12.0-55.13.1.el10_0.aarch64 #1 SMP PREEMPT_DYNAMIC
Mon May 19 07:29:57 UTC 2025 aarch64
 hardware: 4 cpus, 1 disk, 1 node, 3579MB RAM
 timezone: JST-9
 services: pmcd
    pmcd: Version 6.3.7-1, 12 agents, 6 clients
    pmda: root pmcd proc pmproxy xfs linux nfsclient mmv kvm jbd2
          dm openmetrics
 pmlogger: primary logger: /var/log/pcp/pmlogger/arm10.local/20250529.15.49
    pmie: primary engine: /var/log/pcp/pmie/arm10.local/pmie.log
```

Additional resources

- **pmlogger(1)** man pages on your system
- System services and tools distributed with PCP
- **/var/lib/pcp/config/pmlogger/config.default** file

4.4. SETTING UP A CLIENT SYSTEM FOR METRICS COLLECTION

You can configure a client system to enable a central server to collect performance metrics from clients running PCP.

Prerequisites

- PCP is installed. For more information, see Installing and enabling PCP.

Procedure

1. Install the **pcp-system-tools** package:

```
# yum install pcp-system-tools
```

2. Configure an IP address for pmcd:

```
# echo "-i 192.168.4.62" >>/etc/pcp/pmcd/pmcd.options
```

   Replace *192.168.4.62* with the IP address, the client should listen on. By default, **pmcd** is listening on the localhost.

3. Configure the firewall to add the public zone permanently:

```
# firewall-cmd --permanent --zone=public --add-port=44321/tcp
success
# firewall-cmd --reload
success
```

4. Set an SELinux boolean:

```
# setsebool -P pcp_bind_all_unreserved_ports on
```

5. Enable the **pmcd** and **pmlogger** services:

```
# systemctl enable pmcd pmlogger
# systemctl restart pmcd pmlogger
```

Verification

- Verify that the **pmcd** is correctly listening on the configured IP address:

```
# ss -tlp | grep 44321
LISTEN   0   5     127.0.0.1:44321    0.0.0.0:*    users:(("pmcd",pid=151595,fd=6))
LISTEN   0   5  192.168.4.62:44321    0.0.0.0:*    users:(("pmcd",pid=151595,fd=0))
LISTEN   0   5        [::1]:44321       [::]:*    users:(("pmcd",pid=151595,fd=7))
```

Additional resources

- **pmlogger(1)**, **firewall-cmd(1)**, **ss(8)**, and **setsebool(8)** man pages on your system
- System services and tools distributed with PCP
- **/var/lib/pcp/config/pmlogger/config.default** file

4.5. SETTING UP A CENTRAL SERVER TO COLLECT DATA

You can create a central server to collect metrics from clients running PCP.

Prerequisites

- PCP is installed. For more information, see Installing and enabling PCP .
- Client is configured for metrics collection. For more information, see Setting up a client system for metrics collection.

Procedure

1. Install the **pcp-system-tools** package:

```
# yum install pcp-system-tools
```

2. Create the **/etc/pcp/pmlogger/control.d/remote** file with the following content:

```
# DO NOT REMOVE OR EDIT THE FOLLOWING LINE
$version=1.1
192.168.4.13 n n PCP_ARCHIVE_DIR/rhel7u4a -r -T24h10m -c config.rhel7u4a
192.168.4.14 n n PCP_ARCHIVE_DIR/rhel6u10a -r -T24h10m -c config.rhel6u10a
192.168.4.62 n n PCP_ARCHIVE_DIR/rhel8u1a -r -T24h10m -c config.rhel8u1a
```

   Replace *192.168.4.13*, *192.168.4.14* and *192.168.4.62* with the client IP addresses.

3. Enable the pmcd and pmlogger services:

```
# systemctl enable pmcd pmlogger
# systemctl restart pmcd pmlogger
```

Verification

- Verify that you can access the latest archive file from each directory:

```
# for i in /var/log/pcp/pmlogger/rhel*/*.0; do pmdumplog -L $i; done
Log Label (Log Format Version 2)
Performance metrics from host rhel6u10a.local
  commencing Mon Nov 25 21:55:04.851 2019
  ending     Mon Nov 25 22:06:04.874 2019
Archive timezone: JST-9
PID for pmlogger: 24002
Log Label (Log Format Version 2)
Performance metrics from host rhel7u4a
  commencing Tue Nov 26 06:49:24.954 2019
  ending     Tue Nov 26 07:06:24.979 2019
Archive timezone: CET-1
PID for pmlogger: 10941
[..]
```

The archive files from the **/var/log/pcp/pmlogger/** directory can be used for further analysis and graphing.

Additional resources

- **pmlogger(1)** man pages on your system
- System services and tools distributed with PCP
- **/var/lib/pcp/config/pmlogger/config.default** file

4.6. SYSTEMD UNITS AND PMLOGGER

When you deploy the **pmlogger** service, either as a single host monitoring itself or a **pmlogger** farm with a single host collecting metrics from several remote hosts, there are several associated **systemd** service and timer units that are automatically deployed. These services and timers provide routine checks to ensure that your **pmlogger** instances are running, restart any missing instances, and perform archive management such as file compression. The checking and housekeeping services typically deployed by pmlogger are:

pmlogger_daily.service

Runs daily, soon after midnight by default, to aggregate, compress, and rotate one or more sets of PCP archives. Also culls archives older than the limit, 2 weeks by default. Triggered by the **pmlogger_daily.timer** unit, which is required by the **pmlogger.service** unit.

pmlogger_check

Performs half-hourly checks that **pmlogger** instances are running. Restarts any missing instances and performs any required compression tasks. Triggered by the **pmlogger_check.timer** unit, which is required by the **pmlogger.service** unit.

pmlogger_farm_check

Checks the status of all configured **pmlogger** instances. Restarts any missing instances. Migrates all non-primary instances to the **pmlogger_farm** service. Triggered by the **pmlogger_farm_check.timer**, which is required by the **pmlogger_farm.service** unit that is itself required by the **pmlogger**.service unit.

These services are managed through a series of positive dependencies, meaning that they are all

enabled upon activating the primary pmlogger instance. Note that while **pmlogger_daily.service** is disabled by default, **pmlogger_daily.timer** being active via the dependency with **pmlogger.service** will trigger **pmlogger_daily.service** to run.

pmlogger_daily is also integrated with **pmlogrewrite** for automatically rewriting archives before merging. This helps to ensure metadata consistency amid changing production environments and PMDAs. For example, if **pmcd** on one monitored host is updated during the logging interval, the semantics for some metrics on the host might be updated, thus making the new archives incompatible with the previously recorded archives from that host.

Additional resources

- **pmlogrewrite(1)** man page on your system

4.7. MANAGING SYSTEMD SERVICES TRIGGERED BY PMLOGGER

You can create an automated custom archive management system for data collected by your **pmlogger** instances by using the control files.

NOTE

The primary **pmlogger** instance must be running on the same host as the **pmcd** it connects to. You do not need to have a primary instance and you might not need it in your configuration if one central host is collecting data on several **pmlogger** instances connected to **pmcd** instances running on a remote host.

Procedure

- Do one of the following:
  - For the primary pmlogger instance, use **/etc/pcp/pmlogger/control.d/local**
  - For the remote hosts, use **/etc/pcp/pmlogger/control.d/*remote***
    Replace *remote* with your desired file name.

The file should contain one line for each host to be logged. The default format of the primary logger instance that is automatically created looks similar to:

```
# === LOGGER CONTROL SPECIFICATIONS ===
#
#Host     P?  S?   directory      args
# local primary logger
LOCALHOSTNAME   y   n   PCP_ARCHIVE_DIR/LOCALHOSTNAME    -r -T24h10m -c
config.default -v 100Mb
```

Where the fields are: Host:: The name of the host to be logged.

P?

Stands for “Primary?” This field indicates if the host is the primary logger instance, **y**, or not, **n**. There can only be one primary logger across all the files in your configuration and it must be running on the same host as the **pmcd** it connects to.

S?

Stands for “Socks?” This field indicates if this logger instance needs to use the **SOCKS** protocol to connect to **pmcd** through a firewall, **y**, or not, **n**.

directory

All archives associated with this line are created in this directory.

args

Arguments passed to **pmlogger**. The default values for the **args** field are:

-r

Report the archive sizes and growth rate.

T24h10m

Specifies when to end logging for each day. This is typically the time when **pmlogger_daily.service** runs. The default value of **24h10m** indicates that logging should end 24 hours and 10 minutes after it begins, at the latest.

-c config.default

Specifies which configuration file to use. This essentially defines what metrics to record.

-v 100Mb

Specifies the size at which point one data volume is filled and another is created. After it switches to the new archive, the previously recorded one will be compressed by either **pmlogger_daily** or **pmlogger_check**.

Additional resources

- **pmlogger(1)**, **pmlogrewrite(1)**, **pmlogger_daily(1)**, **pmlogger_check(1)**, and **pmlogger.control(5)** man pages on your system

4.8. REPLAYING THE PCP LOG ARCHIVES WITH PMREP

After recording the metric data, you can replay the PCP log archives. To export the logs to text files and import them into spreadsheets, use PCP utilities such as **pcp2csv**, **pcp2xml**, **pmrep** or **pmlogsummary**. By using the **pmrep** tool, you can:

- View the log files.
- Parse the selected PCP log archive and export the values into an ASCII table.
- Extract the entire archive log or only select metric values from the log by specifying individual metrics on the command line.

Prerequisites

- PCP is installed. For more information, see Installing and enabling PCP.
- The **pmlogger** service is enabled. For more information, see Enabling the pmlogger service.
- Installed the **pcp-gui** package.

```
# yum install pcp-gui
```

Procedure

- Display the data on the metric:

```
$ pmrep --start @3:00am --archive 20211128 --interval 5seconds --samples 10 --output csv
disk.dev.write
```

```
Time,"disk.dev.write-sda","disk.dev.write-sdb"
2021-11-28 03:00:00,,
2021-11-28 03:00:05,4.000,5.200
2021-11-28 03:00:10,1.600,7.600
2021-11-28 03:00:15,0.800,7.100
2021-11-28 03:00:20,16.600,8.400
2021-11-28 03:00:25,21.400,7.200
2021-11-28 03:00:30,21.200,6.800
2021-11-28 03:00:35,21.000,27.600
2021-11-28 03:00:40,12.400,33.800
2021-11-28 03:00:45,9.800,20.600
```

The example displays the data on the **disk.dev.write** metric collected in an archive at a 5 second interval in **comma-separated-value** format. Replace *20211128* in this example with a filename containing the **pmlogger** archive you want to display data for.

Additional resources

- **pmlogger(1)**, **pmrep(1)**, and **pmlogsummary(1)** man pages on your system
- System services and tools distributed with PCP

CHAPTER 5. MONITORING PERFORMANCE WITH PERFORMANCE CO-PILOT

Performance Co-Pilot (PCP) is a suite of tools, services, and libraries for monitoring, visualizing, storing, and analyzing system-level performance measurements. As a system administrator, you can monitor the system's performance by using PCP in Red Hat Enterprise Linux.

5.1. MONITORING POSTFIX WITH PMDA-POSTFIX

You can monitor performance metrics of the postfix mail server with pmda-postfix. It helps to check how many emails are received per second.

Prerequisites

- PCP is installed. For more information, see Installing and enabling PCP .
- The **pmlogger** service is enabled. For more information, see Enabling the pmlogger service.

Procedure

1. Install the following packages:
   a. Install the **pcp-system-tools**:

```
# yum install pcp-system-tools
```

   b. Install the **pmda-postfix** package to monitor postfix:

```
# yum install pcp-pmda-postfix postfix
```

   c. Install the **logging** daemon:

```
# yum install rsyslog
```

   d. Install the mail client for testing:

```
# yum install mutt
```

2. Enable the **postfix** and **rsyslog** services:

```
# systemctl enable postfix rsyslog
# systemctl restart postfix rsyslog
```

3. Enable the SELinux boolean, so that **pmda-postfix** can access the required log files:

```
# setsebool -P pcp_read_generic_logs=on
```

4. Install the PMDA:

```
# cd /var/lib/pcp/pmdas/postfix/
# ./Install
Updating the Performance Metrics Name Space (PMNS) ...
```

```
Terminate PMDA if already installed ...
Updating the PMCD control file, and notifying PMCD ...
Waiting for pmcd to terminate ...
Starting pmcd ...
Check postfix metrics have appeared ... 7 metrics and 58 values
```

Verification

- Verify the **pmda-postfix** operation:

```
echo testmail | mutt root
Verify the available metrics:
# pminfo postfix
postfix.received
postfix.sent
postfix.queues.incoming
postfix.queues.maildrop
postfix.queues.hold
postfix.queues.deferred
postfix.queues.active
```

Additional resources

- **rsyslogd(8)**, **postfix(1)**, and **setsebool(8)** man pages on your system
- System services and tools distributed with PCP

5.2. VISUALLY TRACING PCP LOG ARCHIVES WITH THE PCP CHARTS APPLICATION

After recording metric data, you can replay the PCP log archives as graphs. The metrics are sourced from one or more live hosts with alternative options to use metric data from PCP log archives as a source of historical data. To customize the PCP Charts application interface to display the data from the performance metrics, you can use line plot, bar graphs, or utilization graphs.

By using the PCP Charts application, you can:

- Replay the data in the PCP Charts application application and use graphs to visualize the retrospective data alongside live data of the system.
- Plot performance metric values into graphs.
- Display multiple charts simultaneously.

Prerequisites

- PCP is installed. For more information, see Installing and enabling PCP.
- Logged performance data with the **pmlogger**. For more information, see Logging performance data with pmlogger.
- Installed the **pcp-gui** package.

```
# yum install pcp-gui
```

Procedure

1. Launch the PCP Charts application from the command line:

```
# pmchart
```

   The **pmtime** server settings are located at the bottom. With **start** and **pause** buttons you can control:

   - The interval in which PCP polls the metric data
   - The date and time for the metrics of historical data

2. Click **File** and then **New Chart** to select metric from both the local machine and remote machines by specifying their host name or address. Advanced configuration options include the ability to manually set the axis values for the chart, and to manually choose the color of the plots.

3. Record the views created in the PCP Charts application:
   Following are the options to take images or record the views created in the PCP Charts application:

   - Click **File** and then **Export** to save an image of the current view.
   - Click **Record** and then **Start** to start a recording.
   - Click **Record** and then **Stop** to stop the recording. After stopping the recording, the recorded metrics are archived to be viewed later.

4. Optional: In the **PCP Charts** application, the main configuration file, known as the view, allows the metadata associated with one or more charts to be saved. This metadata describes all chart aspects, including the metrics used and the chart columns.

5. Optional: Save the custom view configuration by clicking **File** and then **Save View**, and load the view configuration later.
   The following example of the PCP Charts application view configuration file describes a stacking chart graph showing the total number of bytes read and written to the given XFS file system **loop1**:

```
# pmchart
version 1
chart title "Filesystem Throughput /loop1" style stacking antialiasing off
   plot legend Read rate   metric xfs.read_bytes   instance  loop1
   plot legend Write rate  metric xfs.write_bytes  instance  loop1
```

Additional resources

- **pmchart(1)** and **pmtime(1)** man pages on your system
- System services and tools distributed with PCP

5.3. COLLECTING DATA FROM SQL SERVER BY USING PCP

The SQL Server agent in PCP helps you monitor and analyze database performance issues. You can collect data for Microsoft SQL Server via PCP on your system.

Prerequisites

- You have installed Microsoft SQL Server for Red Hat Enterprise Linux and established a **trusted** connection to an SQL server.
- You have installed the Microsoft ODBC driver for SQL Server for Red Hat Enterprise Linux.

Procedure

1. Install PCP:

```
# yum install pcp-zeroconf
```

2. Install packages required for the **pyodbc** driver:

```
# yum install gcc-c++ python3-devel unixODBC-devel
# yum install python3-pyodbc
```

3. Install the **mssql** agent:

   a. Install the Microsoft SQL Server domain agent for PCP:

```
# yum install pcp-pmda-mssql
```

   b. Edit the **/etc/pcp/mssql/mssql.conf** file to configure the SQL server account's username and password for the mssql agent. Ensure that the account you configure has access rights to performance data.

```
username: user_name
password: user_password
```

   Replace *user_name* with the SQL Server account username and *user_password* with the SQL Server user password for this account.

4. Install the agent:

```
# cd /var/lib/pcp/pmdas/mssql
# ./Install
Updating the Performance Metrics Name Space (PMNS) ...
Terminate PMDA if already installed ...
Updating the PMCD control file, and notifying PMCD ...
Check mssql metrics have appeared ... 168 metrics and 598 values
[...]
```

Verification

- Using the pcp command, verify if the SQL Server PMDA (**mssql**) is loaded and running:

```
$ pcp
Performance Co-Pilot configuration on rhel.local:
platform: Linux rhel.local 4.18.0-167.el8.x86_64 #1 SMP Sun Dec 15 01:24:23 UTC 2019
x86_64
 hardware: 2 cpus, 1 disk, 1 node, 2770MB RAM
 timezone: PDT+7
```

```
services: pmcd pmproxy
    pmcd: Version 5.0.2-1, 12 agents, 4 clients
    pmda: root pmcd proc pmproxy xfs linux nfsclient mmv kvm mssql
          jbd2 dm
pmlogger: primary logger: /var/log/pcp/pmlogger/rhel.local/20200326.16.31
   pmie: primary engine: /var/log/pcp/pmie/rhel.local/pmie.log
```

- View the complete list of metrics that PCP can collect from the SQL Server:

```
# pminfo mssql
```

- After viewing the list of metrics, you can report the rate of transactions. For example, to report on the overall transaction count per second, over a five second time window:

```
# pmval -t 1 -T 5 mssql.databases.transactions
```

- View the graphical chart of these metrics on your system by using the pmchart command. For more information, see Visually tracing PCP log archives with the PCP Charts application.

Additional resources

- **pcp(1)**, **pminfo(1)**, **pmval(1)**, **pmchart(1)**, and **pmdamssql(1)** man pages on your system
- Performance Co-Pilot for Microsoft SQL Server with RHEL 8.2 Red Hat Developers Blog post

CHAPTER 6. SETTING UP GRAPHICAL REPRESENTATION OF PCP METRICS

Using a combination of **pcp**, **grafana**, **valkey**, **pcp bpftrace**, and **pcp vector** provides graphical representation of the live data or data collected by PCP.

6.1. SETTING UP PCP WITH PCP-ZEROCONF

You can set up PCP on a system with the **pcp-zeroconf** package. Once the **pcp-zeroconf** package is installed, the system records the default set of metrics into archived files.

Procedure

- Install the pcp-zeroconf package:

```
# yum install pcp-zeroconf
```

Verification

- Ensure that the pmlogger service is active, and starts archiving the metrics:

```
# pcp | grep pmlogger
pmlogger: primary logger: /var/log/pcp/pmlogger/localhost.localdomain/20200401.00.12
```

Additional resources

- **pmlogger** man page on your system
- Monitoring performance with Performance Co-Pilot

6.2. SETTING UP A GRAFANA-SERVER

Grafana generates graphs that are accessible from a browser. The **grafana-server** is a back-end server for the Grafana dashboard. It listens, by default, on all interfaces, and provides web services accessed through the web browser. The **grafana-pcp** plugin interacts with the **pmproxy** daemon in the backend.

Prerequisites

- PCP is configured. For more information, see Setting up PCP with pcp-zeroconf.

Procedure

1. Install the following packages:

```
# yum install grafana grafana-pcp
```

2. Restart and enable **grafana-server**:

```
# systemctl restart grafana-server
# systemctl enable grafana-server
```

3. Open the server's firewall for network traffic to the Grafana service.

```
# firewall-cmd --permanent --add-service=grafana
success
# firewall-cmd --reload
success
```

Verification

- Ensure that the **grafana-server** is listening and responding to requests:

```
# ss -ntlp | grep 3000
LISTEN  0  128  :3000 *:  users:(("grafana-server",pid=19522,fd=7))
```

- Ensure that the grafana-pcp plugin is installed:

```
# grafana-cli plugins ls | grep performancecopilot-pcp-app
performancecopilot-pcp-app @ 5.2.2
```

Additional resources

- **pmproxy(1)** and **grafana-server** man pages on your system

6.3. CONFIGURING VALKEY

You can use the valkey data source to:

- View data archives
- Query time series using pmseries language
- Analyze data across multiple hosts

Prerequisites

- PCP is configured. For more information, see Setting up PCP with pcp-zeroconf.
- The **grafana-server** is configured. For more information, see Setting up a grafana-server.
- Mail transfer agent, for example, **postfix** is installed and configured.

Procedure

1. Install the **valkey** package:

```
# yum install valkey
```

2. Start and enable the **pmproxy** and **valkey** services:

```
# systemctl start pmproxy valkey
# systemctl enable pmproxy valkey
```

3. Restart **grafana-server**:

```
# systemctl restart grafana-server
```

Verification

- Ensure that the **pmproxy** and **valkey** are working:

```
# pmseries disk.dev.read
2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
```

This command does not return any data if the valkey package is not installed.

Additional resources

- **pmseries(1)** man page on your system

6.4. ACCESSING THE GRAFANA WEB UI

You can access the Grafana web interface. Using the Grafana web interface, you can:

- add Valkey, PCP bpftrace, and PCP Vector data sources
- create a dashboard
- view an overview of any useful metrics
- create alerts in Valkey

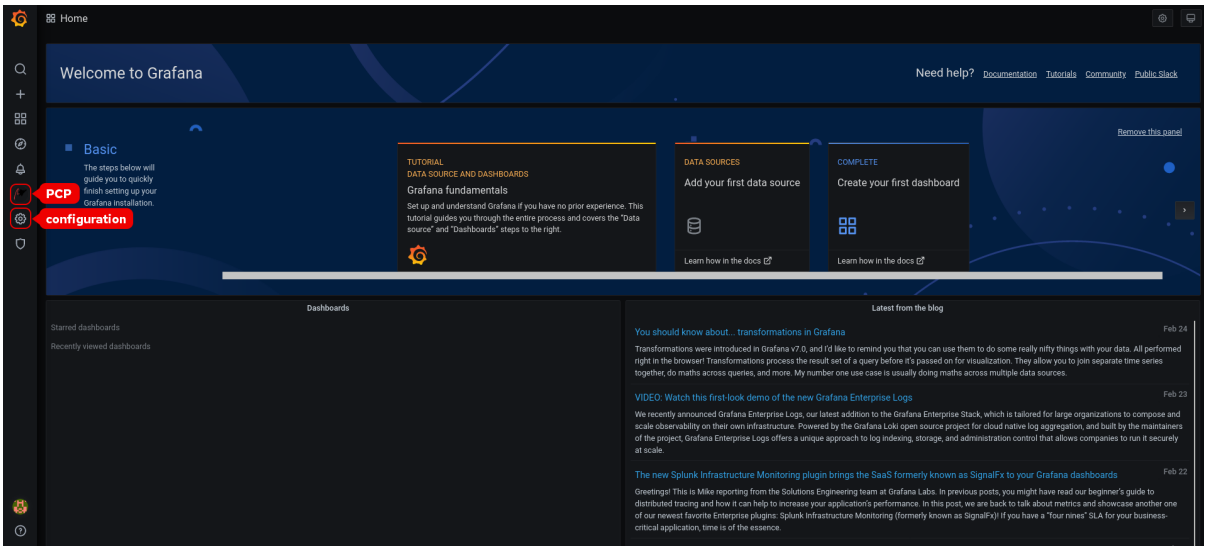
Prerequisites

- PCP is configured. For more information, see Setting up PCP with pcp-zeroconf.
- The **grafana-server** is configured. For more information, see Setting up a grafana-server.

Procedure

1. On the client system, open a browser and access the **grafana-server** on port 3000 by using the *http://192.0.2.0:3000* link.
   Replace *192.0.2.0* with your machine IP when accessing Grafana web UI from a remote machine, or with *localhost* when accessing the web UI locally.
2. For the first login, enter admin in both the **Email** or **username** and **Password** fields.
3. Grafana prompts to set a **New password** to create a secured account. If you want to set it later, click **Skip**.
4. From the hamburger icon (☰) on the top left, click **Administration** > **Plugins**.
5. In the **Plugins** tab, type performance co-pilot in the **Search by name or type** text box and then click the **Performance Co-Pilot (PCP)** plugin.
6. In the **Plugins / Performance Co-Pilot** pane, click **Enable**.
7. Click the **Grafana** icon. The **Grafana Home page** is displayed.

Figure 6.1. Home Dashboard



NOTE

The top corner of the screen has a **Settings** icon, but it controls the general **Dashboard settings**.

8. In the **Grafana Home** page, click Add your first data source to add Valkey, PCP bpftrace, and PCP Vector data sources.
   - To add PCP valkey data source, view default dashboard, create a panel, and an alert rule, see Creating panels and alert in PCP valkey data source .
   - To add pcp bpftrace data source and view the default dashboard, see Viewing the PCP bpftrace System Analysis dashboard.
   - To add pcp vector data source, view the default dashboard, and to view the vector checklist, see Viewing the PCP Vector Checklist.
9. Optional: From the menu, hover over the **admin** profile icon to change the **Preferences** including **Edit Profile**, **Change Password**, or to **Sign out**.

Additional resources

- **grafana-cli** and **grafana-server** man pages on your system

6.5. CONFIGURING SECURE CONNECTIONS FOR VALKEY AND PCP

You can establish secure connections between Performance Co-Pilot (PCP), Grafana, and Valkey. Establishing secure connections between these components helps prevent unauthorized parties from accessing or modifying the data being collected and monitored.

Prerequisites

- PCP is installed. For more information, see Installing and enabling PCP .
- The Grafana server is configured. For more information, see Setting up a Grafana server.
- Valkey is installed. For more information, see Configuring Valkey.

- The private client key is stored in the **/etc/valkey/client.key** file. If you use a different path, modify the path in the corresponding steps of the procedure. For details about creating a private key and certificate signing request (CSR), as well as how to request a certificate from a certificate authority (CA), see your CA's documentation.
- The TLS client certificate is stored in the **/etc/valkey/client.crt** file. If you use a different path, modify the path in the corresponding steps of the procedure.
- The TLS server key is stored in the **/etc/valkey/valkey.key** file. If you use a different path, modify the path in the corresponding steps of the procedure.
- The TLS server certificate is stored in the **/etc/valkey/valkey.crt** file. If you use a different path, modify the path in the corresponding steps of the procedure.
- The CA certificate is stored in the **/etc/valkey/ca.crt** file. If you use a different path, modify the path in the corresponding steps of the procedure.
- For the **pmproxy** daemon, the private server key is stored in the **/etc/pcp/tls/server.key** file. If you use a different path, modify the path in the corresponding steps of the procedure.

Procedure

1. As a root user, open the **/etc/valkey/valkey.conf** file and adjust the TLS/SSL options to reflect the following properties:

```
port 0
tls-port 6379
tls-cert-file /etc/valkey/valkey.crt
tls-key-file /etc/valkey/valkey.key
tls-client-key-file /etc/valkey/client.key
tls-client-cert-file /etc/valkey/client.crt
tls-ca-cert-file /etc/valkey/ca.crt
```

2. Ensure valkey can access the TLS certificates:

```
# su valkey -s /bin/bash -c \
  'ls -1 /etc/valkey/ca.crt /etc/valkey/valkey.key /etc/valkey/valkey.crt'
/etc/valkey/ca.crt
/etc/valkey/valkey.crt
/etc/valkey/valkey.key
```

3. Restart the valkey server to apply the configuration changes:

```
# systemctl restart valkey
```

Verification

- Confirm the TLS configuration works:

```
# valkey-cli --tls --cert /etc/valkey/client.crt \
    --key /etc/valkey/client.key \
    --cacert /etc/valkey/ca.crt <<< "PING"
PONG
Unsuccessful TLS configuration might result in the following error message:
Could not negotiate a TLS connection: Invalid CA Certificate File/Directory
```

6.6. CREATING PANELS AND ALERTS IN PCP VALKEY DATA SOURCE

After adding the PCP Valkey data source, you can view the dashboard with an overview of useful metrics, add a query to visualize the load graph, and create alerts that help you to view the system issues after they occur.

Prerequisites

- The Valkey is configured. For more information, see Configuring Valkey.
- The **grafana-server** is accessible. For more information, see Accessing the Grafana web UI.

Procedure

1. Log into the Grafana web UI.
2. In the **Grafana Home** page, click **Add your first data source**
3. In the **Add data source** pane, type valkey in the **Filter by name or type** text box and then **click Valkey**.
4. In the **Data Sources / Valkey** pane, perform the following:
   a. Add *http://localhost:44322* in the URL field and then click **Save & Test**
   b. Click **Dashboards** tab > **Import** > **Valkey: Host Overview** to see a dashboard with an overview of any useful metrics.

   Figure 6.2. Valkey: Host Overview

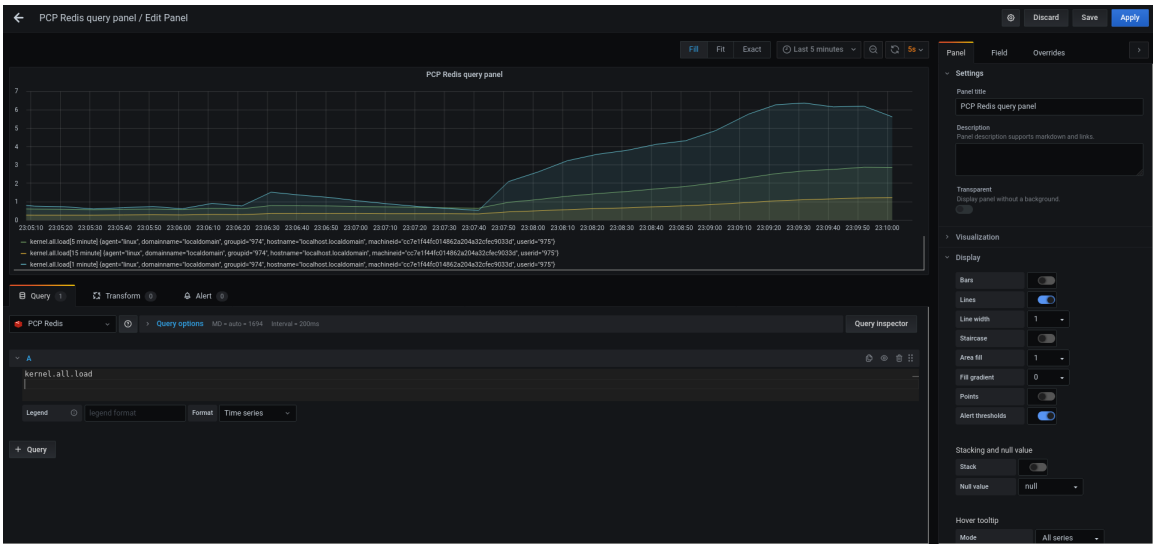
5. Add a new panel:
   a. From the menu, hover over the **Create** icon > **Dashboard** > **Add new panel** icon to add a panel.
   b. In the **Query** tab, select the **Valkey** from the query list instead of the selected **default** option and in the text field of **A**, enter metric, for example, **kernel.all.load** to visualize the kernel load graph.

c. Optional: Add **Panel title** and **Description**, and update other options from the **Settings**.

d. Click **Save** to apply changes and save the dashboard. Add **Dashboard name**.

e. Click **Apply** to apply changes and go back to the dashboard.

Figure 6.3. Valkey query panel



6. Create an alert rule:

a. In the Valkey query panel, click **Alert** and then click **Create Alert**.

b. Edit the **Name**, **Evaluate query**, and **For** fields from the **Rule**, and specify the **Conditions** for your alert.

c. Click **Save** to apply changes and save the dashboard. Click Apply to apply changes and go back to the dashboard.

Figure 6.4. Creating alerts in the Valkey panel

d. Optional: In the same panel, scroll down and click **Delete** icon to delete the created rule.

e. Optional: From the menu, click **Alerting** icon to view the created alert rules with different alert statuses, to edit the alert rule, or to pause the existing rule from the **Alert Rules** tab. To add a notification channel for the created alert rule to receive an alert notification from Grafana, see Adding notification channels for alerts.

6.7. ADDING NOTIFICATION CHANNELS FOR ALERTS

By adding notification channels, you can receive an alert notification from Grafana whenever the alert rule conditions are met and the system needs further monitoring. You can receive these alerts after selecting any one type from the supported list of notifiers, which includes **DingDing, Discord, Email, Google Hangouts Chat, HipChat, Kafka REST Proxy, LINE, Microsoft Teams, OpsGenie, PagerDuty, Prometheus Alertmanager, Pushover, Sensu, Slack, Telegram, Threema Gateway, VictorOps,** and **webhook**.

Prerequisites

- The **grafana-server** is accessible. For more information, see Accessing the Grafana web UI.
- An alert rule is created. For more information, see Creating panels and alert in PCP valkey data source.
- Configure SMTP and add a valid sender's email address in the **grafana/grafana.ini** file:

```
# vi /etc/grafana/grafana.ini
[smtp]
enabled = true
from_address = abc@gmail.com
```

Replace *abc@gmail.com* by a valid email address.

Restart grafana-server

```
# systemctl restart grafana-server.service
```

Procedure

1. From the menu, hover over the **Alerting** icon > click **Notification channels** > **Add channel**.
2. In the Add notification channel details pane, perform the following:
   a. Enter your name in the **Name** text box.
   b. Select the communication **Type**, for example, **Email** and enter the email address. You can add multiple email addresses using the **;** separator.
   c. Optional: **Configure Optional Email** settings and **Notification** settings.
3. Click **Save**.
4. Select a notification channel in the alert rule:
   a. From the menu, hover over the **Alerting** icon and then click **Alert rules**.
   b. From the **Alert Rules** tab, click the created alert rule.
   c. On the **Notifications** tab, select your notification channel name from the **Send to** option, and then add an alert message.
5. Click **Apply**.

Additional resources

- Upstream Grafana documentation for alert notifications

6.8. SETTING UP AUTHENTICATION BETWEEN PCP COMPONENTS

You can set up authentication using the **scram-sha-256** authentication mechanism, which is supported by PCP through the Simple Authentication Security Layer (SASL) framework.

Procedure

1. Install the SASL framework for the **scram-sha-256** authentication mechanism:

```
# yum install cyrus-sasl-scram cyrus-sasl-lib
```

2. Specify the supported authentication mechanism and the user database path in the **pmcd.conf** file:

```
# vi /etc/sasl2/pmcd.conf
mech_list: scram-sha-256
sasldb_path: /etc/pcp/passwd.db
```

3. Create a new user:

```
# useradd -r metrics
```

Replace *metrics* by your user name.

4. Add the created user in the user database:

```
# saslpasswd2 -a pmcd metrics
Password:
Again (for verification):
```

To add the created user, it is required to enter the *metrics* account password.

5. Set the permissions of the user database:

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

6. Restart the *pmcd* service:

```
# systemctl restart pmcd
```

Verification

- Verify the SASL configuration:

```
# pminfo -f -h "pcp://127.0.0.1?username=metrics" disk.dev.read
Password:
disk.dev.read
inst [0 or "sda"] value 19540
```

Additional resources

- **saslauthd(8)**, **pminfo(1)**, and **sha256** man pages on your system
- How can I setup authentication between PCP components, like PMDAs and pmcd in RHEL 8.2? (Red Hat Knowledgebase)

6.9. INSTALLING PCP BPFTRACE

You can install the **PCP bpftrace** agent to introspect a system and to gather metrics from the kernel and user-space tracepoints. The **bpftrace** agent uses **bpftrace** scripts to gather the metrics. The **bpftrace** scripts use the enhanced Berkeley Packet Filter (eBPF).

Prerequisites

- PCP is configured. For more information, see Setting up PCP with pcp-zeroconf.
- The **grafana-server** is configured. For more information, see Setting up a grafana-server.
- The **scram-sha-256** authentication mechanism is configured. For more information, see Setting up authentication between PCP components.

Procedure

1. Install the **pcp-pmda-bpftrace** package:

```
# yum install pcp-pmda-bpftrace
```

2. Edit the bpftrace.conf file and add the user that you have created in Setting up authentication between PCP components:

```
# vi /var/lib/pcp/pmdas/bpftrace/bpftrace.conf
[dynamic_scripts]
enabled = true
auth_enabled = true
allowed_users = root,metrics
```

   Replace *metrics* by your user name.

3. Install **bpftrace** PMDA:

```
# cd /var/lib/pcp/pmdas/bpftrace/
# ./Install
Updating the Performance Metrics Name Space (PMNS) ...
Terminate PMDA if already installed ...
Updating the PMCD control file, and notifying PMCD …
Check bpftrace metrics have appeared ... 7 metrics and 6 values
```

   The **pmda-bpftrace** is now installed, and can only be used after authenticating your user. For more information, see Viewing the PCP bpftrace System Analysis dashboard.

Additional resources

- **pmdabpftrace(1)** and **bpftrace** man pages on your system

6.10. VIEWING THE PCP BPFTRACE SYSTEM ANALYSIS DASHBOARD

Using the PCP bpftrace data source, you can access the live data from sources which are not available as normal data from the pmlogger or archives In the PCP bpftrace data source, you can view the dashboard with an overview of useful metrics.

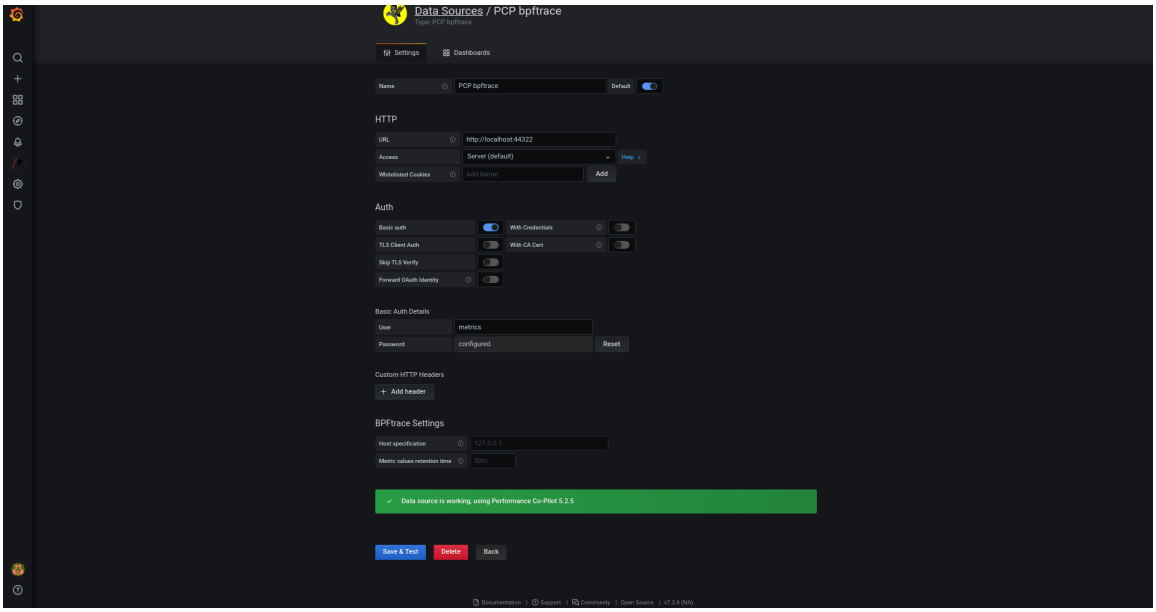
Prerequisites

- The PCP **bpftrace** is installed. For more information, see Installing PCP bpftrace.
- The **grafana-server** is configured. For more information, see Setting up a grafana-server.

Procedure

1. Log into the **Grafana web UI**.
2. In the **Grafana Home** page, click **Add your first data source**
3. In the **Add data source** pane, type **bpftrace** in the **Filter by name or type** text box and then click **PCP bpftrace**.
4. In the **Data Sources / PCP bpftrace** pane, perform the following:
   a. Add *http://localhost:44322* in the URL field.
   b. Toggle the **Basic Auth** option and add the created user credentials in the **User** and **Password** field.
   c. Click **Save & Test**

   Figure 6.5. Adding PCP bpftrace in the data source



   d. Click **Dashboards** tab > **Import** > **PCP bpftrace: System Analysis** to see a dashboard with an overview of any useful metrics.

Figure 6.6. PCP bpftrace: System Analysis

6.11. INSTALLING PCP VECTOR

This procedure describes how to install a pcp vector.

Prerequisites

- PCP is configured. For more information, see Setting up PCP with pcp-zeroconf.
- The **grafana-server** is configured. For more information, see Setting up a grafana-server.

Procedure

- Install the bcc PMDA:

```
# cd /var/lib/pcp/pmdas/bcc
# ./Install
[Wed Apr  1 00:27:48] pmdabcc(22341) Info: Initializing, currently in 'notready' state.
[Wed Apr  1 00:27:48] pmdabcc(22341) Info: Enabled modules:
[Wed Apr  1 00:27:48] pmdabcc(22341) Info: ['biolatency', 'sysfork',
[...]
Updating the Performance Metrics Name Space (PMNS) ...
Terminate PMDA if already installed ...
Updating the PMCD control file, and notifying PMCD …
Check bcc metrics have appeared ... 1 warnings, 1 metrics and 0 values
```

Additional resources

- **pmdabcc(1)** man page on your system

6.12. VIEWING THE PCP VECTOR CHECKLIST

The PCP Vector data source displays live metrics and uses the pcp metrics. It analyzes data for individual hosts. After adding the PCP Vector data source, you can view the dashboard with an overview of useful metrics and view the related troubleshooting or reference links in the checklist.

Prerequisites

- The PCP Vector is installed. For more information, see Installing PCP Vector.
- The **grafana-server** is accessible. For more information, see Accessing the Grafana web UI.

Procedure

1. Log into the **Grafana web UI**.
2. In the **Grafana Home** page, click **Add your first data source**
3. In the **Add data source** pane, type vector in the **Filter by name or type** text box and then click **PCP Vector**.
4. In the **Data Sources / PCP Vector** pane, perform the following:
   a. Add *http://localhost:44322* in the **URL** field and then click **Save & Test**
   b. Click **Dashboards** tab > **Import** * **PCP Vector: Host Overview** to see a dashboard with an overview of any useful metrics.

   Figure 6.7. PCP Vector: Host Overview

5. From the menu, hover over the **Performance Co-Pilot** plugin and then click **PCP Vector Checklist**.
   In the PCP checklist, click help or warning icon to view the related troubleshooting or reference links.

Figure 6.8. Performance Co-Pilot / PCP Vector Checklist

6.13. USING HEATMAPS IN GRAFANA

You can use heatmaps in Grafana to view histograms of your data over time, identify trends and patterns in your data, and see how they change over time. Each column within a heatmap represents a single histogram with different colored cells representing the different densities of observation of a given value within that histogram.

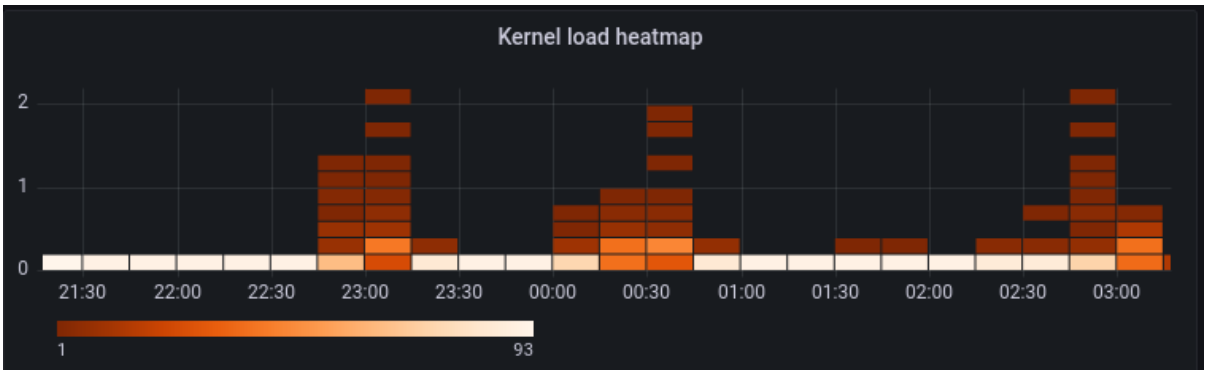
Prerequisites

- Valkey is configured. For more information see Configuring Valkey.
- The **grafana-server** is accessible. For more information, see Accessing the Grafana web UI.
- The PCP valkey data source is configured. For more information see Creating panels and alerts in PCP Valkey data source.

Procedure

1. Hover the cursor over the **Dashboards** tab and click **+ New dashboard**.
2. In the **Add panel** menu, click **Add a new panel**
3. In the **Query** tab:
   a. Select **Valkey** from the query list instead of the selected default option.
   b. In the text field of **A**, enter a metric, for example, **kernel.all.load** to visualize the kernel load graph.
4. Click the visualization dropdown menu, which is set to **Time series** by default, and then click **Heatmap**.
5. Optional: In the **Panel Options** dropdown menu, add a **Panel Title** and **Description**.
6. In the **Heatmap** dropdown menu, under the **Calculate from data** setting, click **Yes**.

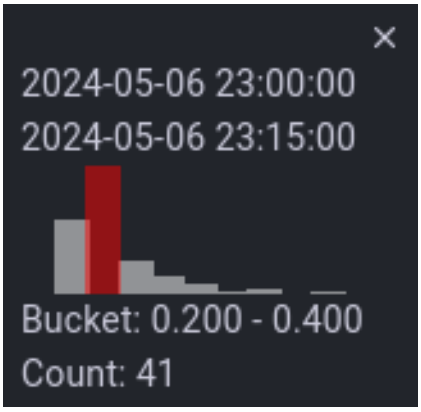
Figure 6.9. Heatmap



7. Optional: In the **Colors** dropdown menu, change the **Scheme** from the default **Orange** and select the number of steps (color shades).

8. Optional: In the **Tooltip** dropdown menu, under the **Show histogram (Y Axis)** setting, click the toggle to display a cell's position within its specific histogram when hovering your cursor over a cell in the heatmap. For example:

Show histogram (Y Axis) cell display



6.14. TROUBLESHOOTING GRAFANA ISSUES

At times, it is necessary to troubleshoot Grafana issues, such as, Grafana does not display any data, the dashboard is black, or similar issues.

Procedure

- Verify that the **pmlogger** service is up and running by executing the following command:

```
$ systemctl status pmlogger
```

- Verify if files were created or modified to the disk by executing the following command:

```
$ ls /var/log/pcp/pmlogger/$(hostname)/ -rlt
total 4024
-rw-r--r--. 1 pcp pcp   45996 Oct 13  2019 20191013.20.07.meta.xz
-rw-r--r--. 1 pcp pcp     412 Oct 13  2019 20191013.20.07.index
-rw-r--r--. 1 pcp pcp   32188 Oct 13  2019 20191013.20.07.0.xz
-rw-r--r--. 1 pcp pcp   44756 Oct 13  2019 20191013.20.30-00.meta.xz
[..]
```

- Verify that the **pmproxy** service is running by executing the following command:

```
$ systemctl status pmproxy
```

- Verify that **pmproxy** is running, time series support is enabled, and a connection to valkey is established by viewing the **/var/log/pcp/pmproxy/pmproxy.log** file and ensure that it contains the following text:

```
pmproxy(1716) Info: valkey slots, command keys, schema version setup
Here, 1716 is the PID of pmproxy, which will be different for every invocation of pmproxy.
Verify if the valkey database contains any keys by executing the following command:
$ valkey-cli dbsize
(integer) 34837
```

- Verify if any PCP metrics are in the valkey database and pmproxy is able to access them by executing the following commands:

```
$ pmseries disk.dev.read
2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
$ pmseries "disk.dev.read[count:10]"
2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
   [Mon Jul 26 12:21:10.085468000 2021] 117971
70e83e88d4e1857a3a31605c6d1333755f2dd17c
   [Mon Jul 26 12:21:00.087401000 2021] 117758
70e83e88d4e1857a3a31605c6d1333755f2dd17c
   [Mon Jul 26 12:20:50.085738000 2021] 116688
70e83e88d4e1857a3a31605c6d1333755f2dd17c
[...]
$ valkey-cli --scan --pattern "*$(pmseries 'disk.dev.read')"
pcp:metric.name:series:2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
pcp:values:series:2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
pcp:desc:series:2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
pcp:labelvalue:series:2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
pcp:instances:series:2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
pcp:labelflags:series:2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
```

- Verify if there are any errors in the Grafana logs by executing the following command:

```
$ journalctl -e -u grafana-server
-- Logs begin at Mon 2021-07-26 11:55:10 IST, end at Mon 2021-07-26 12:30:15 IST. --
Jul 26 11:55:17 localhost.localdomain systemd[1]: Starting Grafana instance...
Jul 26 11:55:17 localhost.localdomain grafana-server[1171]: t=2021-07-26T11:55:17+0530
lvl=info msg="Starting Grafana" logger=server version=7.3.6 c>
Jul 26 11:55:17 localhost.localdomain grafana-server[1171]: t=2021-07-26T11:55:17+0530
lvl=info msg="Config loaded from" logger=settings file=/usr/s>
Jul 26 11:55:17 localhost.localdomain grafana-server[1171]: t=2021-07-26T11:55:17+0530
lvl=info msg="Config loaded from" logger=settings file=/etc/g>
[...]
```

CHAPTER 7. OPTIMIZING THE SYSTEM PERFORMANCE IN THE WEB CONSOLE

You can set a performance profile in the RHEL web console to optimize the performance of the system for a selected task.

7.1. PERFORMANCE TUNING OPTIONS IN THE WEB CONSOLE

Red Hat Enterprise Linux provides several performance profiles that optimize the system for:

- Systems using the desktop
- Throughput performance
- Latency performance
- Network performance
- Low-power consumption
- Virtual machines

The TuneD service optimizes system options to match the selected profile. In the web console, you can set your system's performance profile.

Additional resources

- Optimizing system performance with TuneD

7.2. SETTING A PERFORMANCE PROFILE IN THE WEB CONSOLE

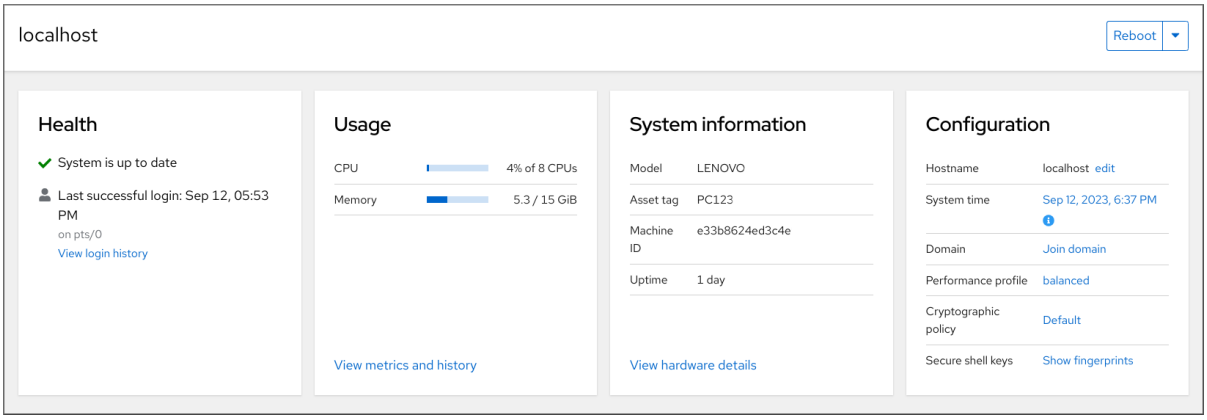
Depending on the task you want to perform, you can use the web console to optimize system performance by setting a suitable performance profile.

Prerequisites

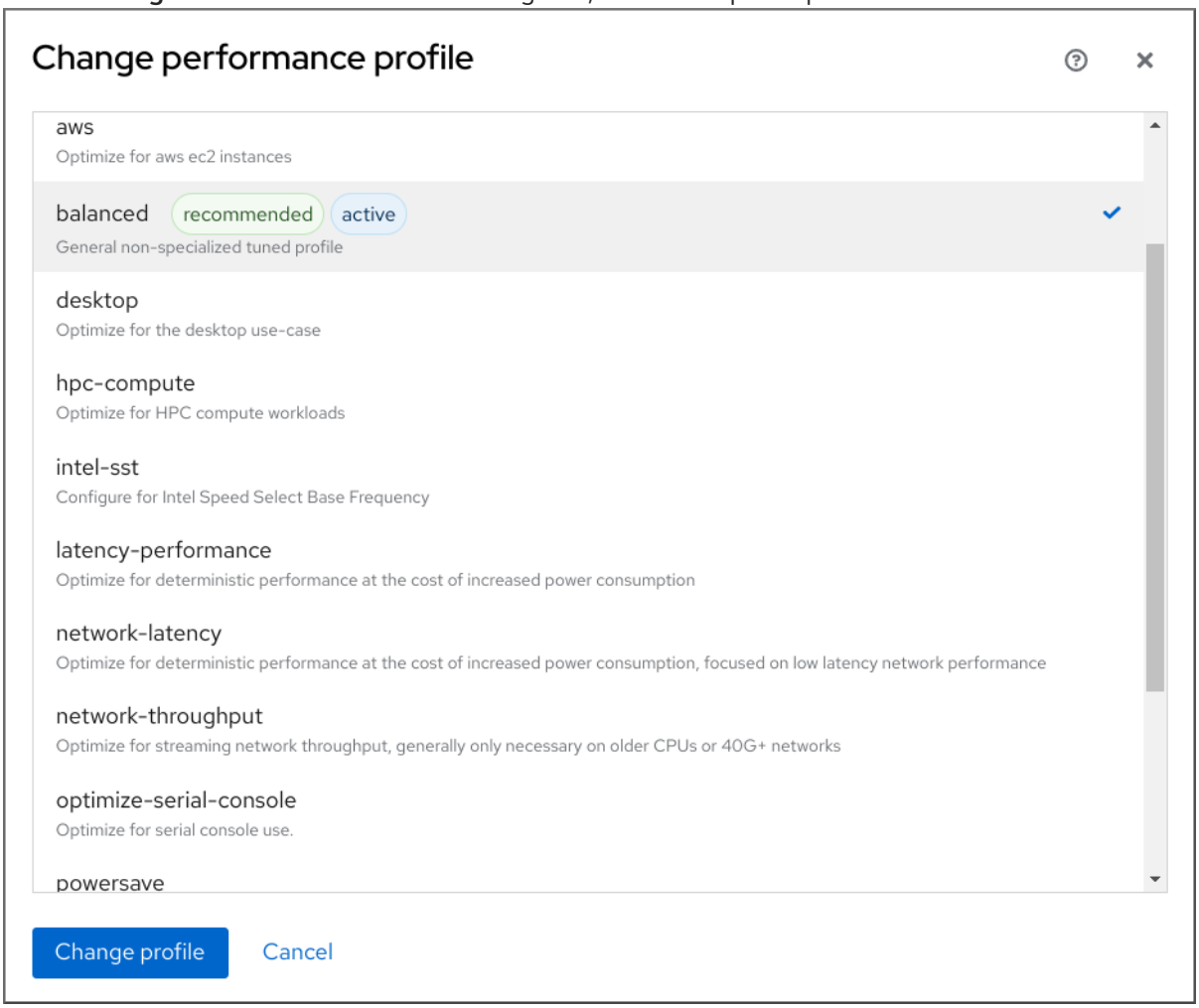
- You have installed the RHEL 10 web console.
  For instructions, see Installing and enabling the web console .

Procedure

1. Log in to the RHEL 10 web console.
   For details, see Logging in to the web console .
2. Click **Overview**.
3. In the **Configuration** section, click the current performance profile.



4. In the **Change Performance Profile** dialog box, set the required profile.



5. Click **Change Profile**.

Verification

- The **Overview** tab now shows the selected performance profile in the **Configuration** section.

7.3. MONITORING PERFORMANCE ON THE LOCAL SYSTEM BY USING THE WEB CONSOLE

The RHEL web console uses the Utilization Saturation and Errors (USE) Method for troubleshooting. The new performance metrics page has a historical view of your data organized chronologically with the newest data at the top. In the **Metrics and history** page, you can view events, errors, and graphical

representation for resource utilization and saturation.

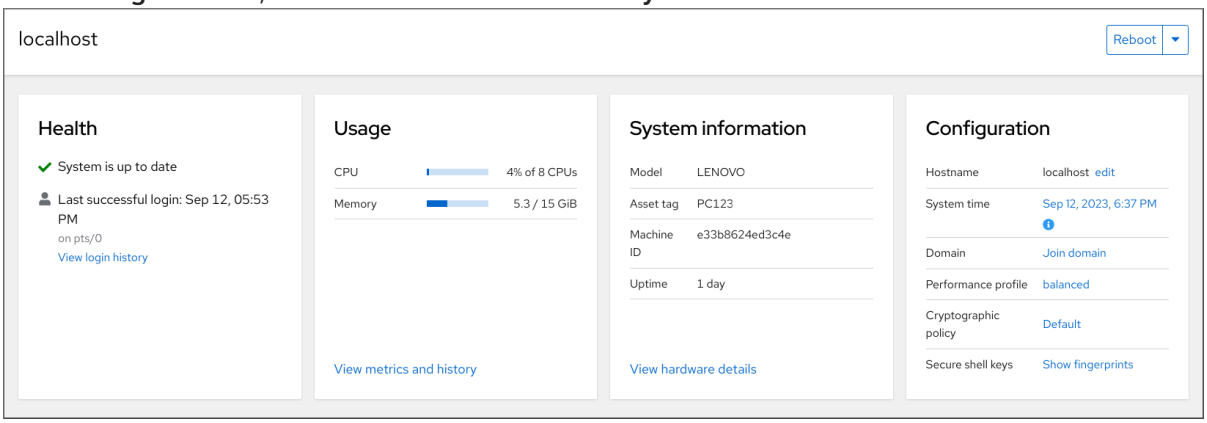
Prerequisites

- You have installed the RHEL 10 web console.
  For instructions, see Installing and enabling the web console .
- The **cockpit-pcp** package is installed.
- The Performance Co-Pilot (PCP) service is enabled:

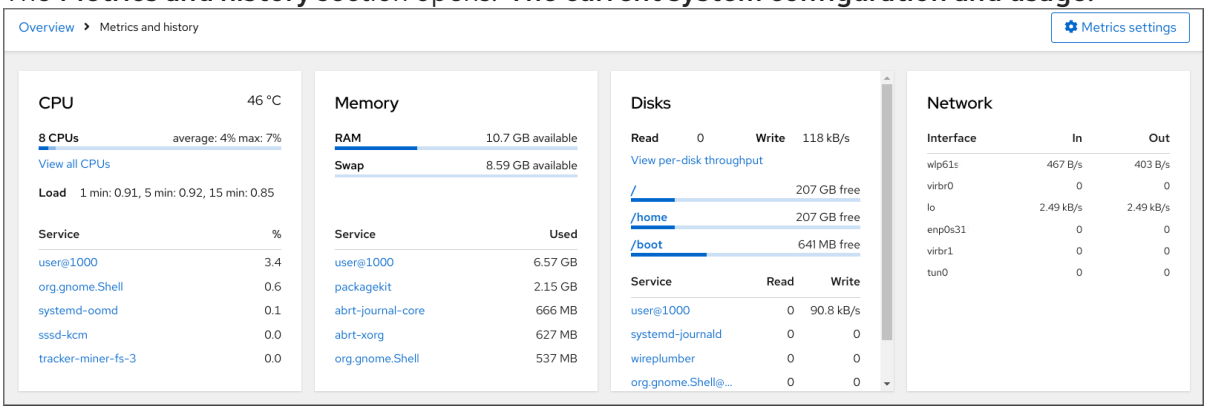
```
# systemctl enable --now pmlogger.service pmproxy.service
```

Procedure

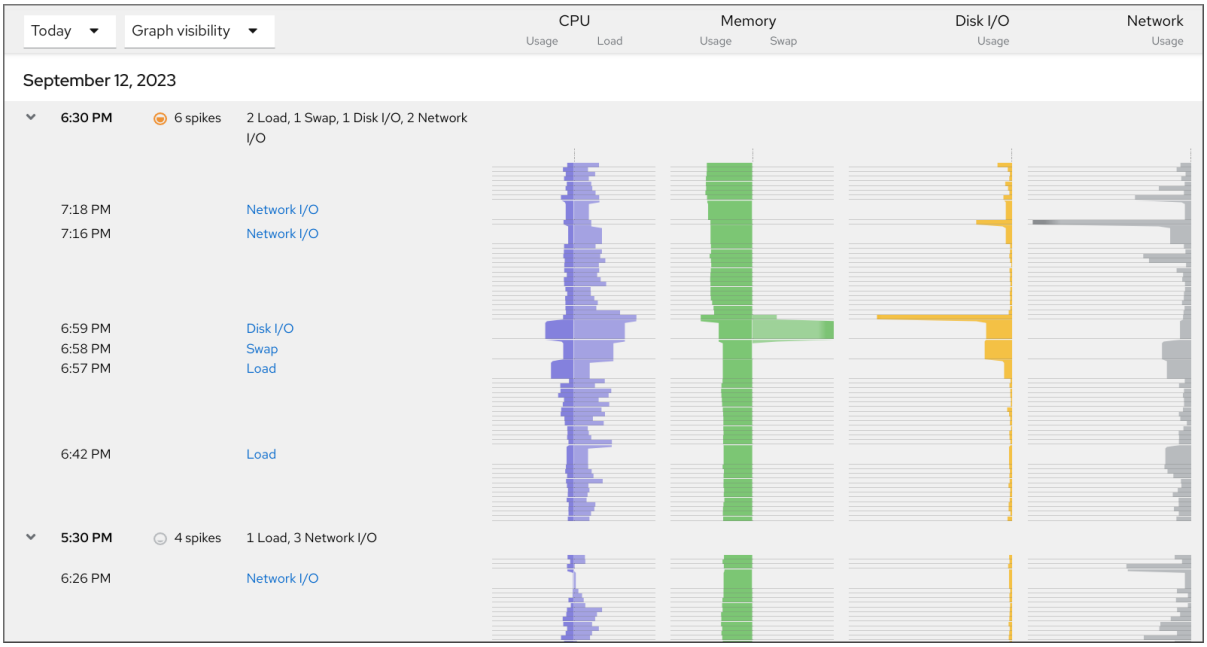
1. Log in to the RHEL 10 web console.
   For details, see Logging in to the web console .
2. Click **Overview**.
3. In the **Usage** section, click **View metrics and history**.



The **Metrics and history** section opens: **The current system configuration and usage:**



The performance metrics in a graphical form over a user-specified time interval:



7.4. MONITORING PERFORMANCE ON SEVERAL SYSTEMS BY USING THE WEB CONSOLE AND GRAFANA

Grafana enables you to collect data from several systems at once and review a graphical representation of their collected Performance Co-Pilot (PCP) metrics. You can set up performance metrics monitoring and export for several systems in the web console interface.

Prerequisites

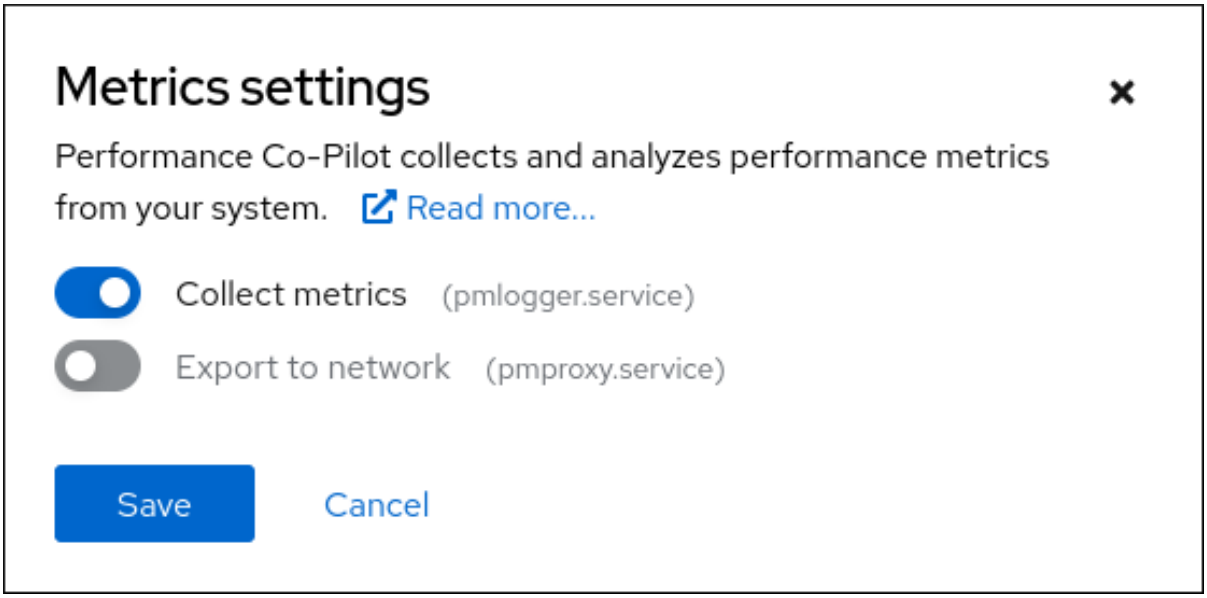
- You have installed the RHEL 10 web console.
  For instructions, see Installing and enabling the web console .
- You have installed the **cockpit-pcp** package.
- You have enabled the **PCP** service:

```
# systemctl enable --now pmlogger.service pmproxy.service
```

- You have set up the **Grafana** dashboard. For more information, see Setting up a grafana-server.
- You have installed the **valkey** package.
  Alternatively, you can install the package from the web console interface later in the procedure.

Procedure

1. Log in to the RHEL 10 web console.
   For details, see Logging in to the web console .
2. In the **Overview** page, click **View metrics and history** in the **Usage** table.
3. Click the **Metrics settings** button.
4. Move the **Export to network** slider to the active position.

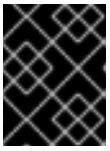


If you do not have the **valkey** package installed, the web console prompts you to install it.

5. To open the **pmproxy** service, select a zone from a drop-down list and click the **Add pmproxy** button.
6. Click **Save**.

Verification

1. Click **Networking**.
2. In the **Firewall** table, click the **Edit rules and zones** button.
3. Search for **pmproxy** in your selected zone.



IMPORTANT

Repeat this procedure on all the systems you want to moponitor the performance.

Additional resources

- Setting up graphical representation of PCP metrics