Red Hat Enterprise Linux 9.0 Beta

Monitoring and managing system status and performance

Optimizing system throughput, latency, and power consumption
Red Hat Enterprise Linux 9.0 Beta Monitoring and managing system status and performance

Optimizing system throughput, latency, and power consumption
Abstract

This documentation collection provides instructions on how to monitor and optimize the throughput, latency, and power consumption of Red Hat Enterprise Linux 9 in different scenarios.
# Table of Contents

**RHEL BETA RELEASE** .................................................. 4

**MAKING OPEN SOURCE MORE INCLUSIVE** ...................................... 5

**PROVIDING FEEDBACK ON RED HAT DOCUMENTATION** ........................................ 6

**CHAPTER 1. MONITORING PERFORMANCE USING RHEL SYSTEM ROLES** ........................................ 7
  1.1. INTRODUCTION TO RHEL SYSTEM ROLES ........................................ 7
  1.2. RHEL SYSTEM ROLES TERMINOLOGY ........................................ 7
  1.3. INSTALLING RHEL SYSTEM ROLES IN YOUR SYSTEM ........................................ 8
  1.4. APPLYING A ROLE ........................................ 8
  1.5. INTRODUCTION TO THE METRICS SYSTEM ROLE ........................................ 11
  1.6. USING THE METRICS SYSTEM ROLE TO MONITOR YOUR LOCAL SYSTEM WITH VISUALIZATION ........................................ 12
  1.7. USING THE METRICS SYSTEM ROLE TO SETUP A FLEET OF INDIVIDUAL SYSTEMS TO MONITOR THEMSELVES ........................................ 12
  1.8. USING THE METRICS SYSTEM ROLE TO MONITOR A FLEET OF MACHINES CENTRALLY VIA YOUR LOCAL MACHINE ........................................ 13
  1.9. SETTING UP AUTHENTICATION WHILE MONITORING A SYSTEM USING THE METRICS SYSTEM ROLE ........................................ 14
  1.10. USING THE METRICS SYSTEM ROLE TO CONFIGURE AND ENABLE METRICS COLLECTION FOR SQL SERVER ........................................ 14

**CHAPTER 2. SETTING UP PCP** ........................................ 17
  2.1. OVERVIEW OF PCP ........................................ 17
  2.2. INSTALLING AND ENABLING PCP ........................................ 17
  2.3. DEPLOYING A MINIMAL PCP SETUP ........................................ 18
  2.4. SYSTEM SERVICES DISTRIBUTED WITH PCP ........................................ 19
  2.5. TOOLS DISTRIBUTED WITH PCP ........................................ 20
  2.6. PCP DEPLOYMENT ARCHITECTURES ........................................ 23
  2.7. RECOMMENDED DEPLOYMENT ARCHITECTURE ........................................ 26
  2.8. SIZING FACTORS ........................................ 26
  2.9. CONFIGURATION OPTIONS FOR PCP SCALING ........................................ 27
  2.10. EXAMPLE: ANALYZING THE CENTRALIZED LOGGING DEPLOYMENT ........................................ 27
  2.11. EXAMPLE: ANALYZING THE FEDERATED SETUP DEPLOYMENT ........................................ 28
  2.12. TROUBLESHOOTING HIGH MEMORY USAGE ........................................ 29

**CHAPTER 3. LOGGING PERFORMANCE DATA WITH PMLOGGER** ........................................ 32
  3.1. MODIFYING THE PMLOGGER CONFIGURATION FILE WITH PMLOGCONF ........................................ 32
  3.2. EDITING THE PMLOGGER CONFIGURATION FILE MANUALLY ........................................ 32
  3.3. ENABLING THE PMLOGGER SERVICE ........................................ 33
  3.4. SETTING UP A CLIENT SYSTEM FOR METRICS COLLECTION ........................................ 34
  3.5. SETTING UP A CENTRAL SERVER TO COLLECT DATA ........................................ 35
  3.6. REPLAYING THE PCP LOG ARCHIVES WITH PMDUMPTEXT ........................................ 36

**CHAPTER 4. MONITORING PERFORMANCE WITH PERFORMANCE CO-PILOT** ........................................ 38
  4.1. MONITORING POSTFIX WITH PMDA-POSTFIX ........................................ 38
  4.2. VISUALLY TRACING PCP LOG ARCHIVES WITH THE PCP CHARTS APPLICATION ........................................ 39
  4.3. COLLECTING DATA FROM SQL SERVER USING PCP ........................................ 41

**CHAPTER 5. PERFORMANCE ANALYSIS OF XFS WITH PCP** ........................................ 44
  5.1. INSTALLING XFS PMDA MANUALLY ........................................ 44
  5.2. EXAMINING XFS PERFORMANCE METRICS WITH PMINFO ........................................ 44
  5.3. RESETTING XFS PERFORMANCE METRICS WITH PMSTORE ........................................ 46
  5.4. PCP METRIC GROUPS FOR XFS ........................................ 46
CHAPTER 6. SETTING UP GRAPHICAL REPRESENTATION OF PCP METRICS

6.1. SETTING UP PCP WITH PCP-ZEROCONF
6.2. SETTING UP A GRAFANA-SERVER
6.3. ACCESSING THE GRAFANA WEB UI
6.4. CONFIGURING PCP REDIS
6.5. CREATING PANELS AND ALERT IN PCP REDIS DATA SOURCE
6.6. ADDING NOTIFICATION CHANNELS FOR ALERTS
6.7. SETTING UP AUTHENTICATION BETWEEN PCP COMPONENTS
6.8. INSTALLING PCP BPFTRACE
6.9. VIEWING THE PCP BPFTRACE SYSTEM ANALYSIS DASHBOARD
6.10. INSTALLING PCP VECTOR
6.11. VIEWING THE PCP VECTOR CHECKLIST
6.12. TROUBLESHOOTING GRAFANA ISSUES

CHAPTER 7. CONFIGURING AN OPERATING SYSTEM TO OPTIMIZE CPU UTILIZATION

7.1. TOOLS FOR MONITORING AND DIAGNOSING PROCESSOR ISSUES
7.2. TYPES OF SYSTEM TOPOLOGY
7.2.1. Displaying system topologies
7.3. CONFIGURING KERNEL TICK TIME
7.4. OVERVIEW OF AN INTERRUPT REQUEST
7.4.1. Balancing interrupts manually
7.4.2. Setting the smp_affinity mask

CHAPTER 8. TUNING SCHEDULING POLICY

8.1. CATEGORIES OF SCHEDULING POLICIES
8.2. STATIC PRIORITY SCHEDULING WITH SCHED_FIFO
8.3. ROUND ROBIN PRIORITY SCHEDULING WITH SCHED_RR
8.4. NORMAL SCHEDULING WITH SCHED_OTHER
8.5. SETTING SCHEDULER POLICIES
8.6. POLICY OPTIONS FOR THE CHRT COMMAND
8.7. CHANGING THE PRIORITY OF SERVICES DURING THE BOOT PROCESS
8.8. PRIORITY MAP
8.9. TUNED CPU-PARTITIONING PROFILE
8.10. USING THE TUNED CPU-PARTITIONING PROFILE FOR LOW-LATENCY TUNING
8.11. CUSTOMIZING THE CPU-PARTITIONING TUNED PROFILE

CHAPTER 9. CONFIGURING RESOURCE MANAGEMENT USING CGROUPS VERSION 2 WITH SYSTEMD

9.1. PREREQUISITES
9.2. INTRODUCTION TO RESOURCE DISTRIBUTION MODELS
9.3. ALLOCATING CPU RESOURCES USING SYSTEMD
9.4. CPU TIME ALLOCATION POLICY OPTIONS FOR SYSTEMD
9.5. ALLOCATING MEMORY RESOURCES USING SYSTEMD
9.6. MEMORY ALLOCATION CONFIGURATION OPTIONS FOR SYSTEMD
9.7. CONFIGURING I/O BANDWIDTH USING SYSTEMD
9.8. I/O BANDWIDTH CONFIGURATION OPTIONS FOR SYSTEMD
RHEL BETA RELEASE

Red Hat provides Red Hat Enterprise Linux Beta access to all subscribed Red Hat accounts. The purpose of Beta access is to:

- Provide an opportunity to customers to test major features and capabilities prior to the general availability release and provide feedback or report issues.
- Provide Beta product documentation as a preview. Beta product documentation is under development and is subject to substantial change.

Note that Red Hat does not support the usage of RHEL Beta releases in production use cases. For more information, see What does Beta mean in Red Hat Enterprise Linux and can I upgrade a RHEL Beta installation to a General Availability (GA) release?.
MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright’s message.
PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Please let us know how we could make it better. To do so:

- For simple comments on specific passages:
  1. Make sure you are viewing the documentation in the Multi-page HTML format. In addition, ensure you see the Feedback button in the upper right corner of the document.
  2. Use your mouse cursor to highlight the part of text that you want to comment on.
  3. Click the Add Feedback pop-up that appears below the highlighted text.
  4. Follow the displayed instructions.

- For submitting more complex feedback, create a Bugzilla ticket:
  1. Go to the Bugzilla website.
  2. As the Component, use Documentation.
  3. Fill in the Description field with your suggestion for improvement. Include a link to the relevant part(s) of documentation.
  4. Click Submit Bug.
CHAPTER 1. MONITORING PERFORMANCE USING RHEL SYSTEM ROLES

As a system administrator, you can use the metrics RHEL System Role to monitor the performance of a system.

1.1. INTRODUCTION TO RHEL SYSTEM ROLES

RHEL System Roles is a collection of Ansible roles and modules. RHEL System Roles provide a configuration interface to remotely manage multiple RHEL systems. The interface enables managing system configurations across multiple versions of RHEL, as well as adopting new major releases.

On Red Hat Enterprise Linux 9, the interface currently consists of the following roles:

- network
- certificate
- postfix
- kernel_settings
- metrics
- nbde_client and nbde_server
- tlog
- ssh
- ssd
- crypto_policies

All these roles are provided by the `rhel-system-roles` package available in the AppStream repository.

Additional resources

- Red Hat Enterprise Linux (RHEL) System Roles
- `/usr/share/doc/rhel-system-roles` documentation [1]

1.2. RHEL SYSTEM ROLES TERMINOLOGY

You can find the following terms across this documentation:

System Roles terminology

Ansible playbook

Playbooks are Ansible’s configuration, deployment, and orchestration language. They can describe a policy you want your remote systems to enforce, or a set of steps in a general IT process.

Control node

Any machine with Ansible installed. You can run commands and playbooks, invoking `/usr/bin/ansible`
or /usr/bin/ansible-playbook, from any control node. You can use any computer that has Python installed on it as a control node - laptops, shared desktops, and servers can all run Ansible. However, you cannot use a Windows machine as a control node. You can have multiple control nodes.

**Inventory**

A list of managed nodes. An inventory file is also sometimes called a “hostfile”. Your inventory can specify information like IP address for each managed node. An inventory can also organize managed nodes, creating and nesting groups for easier scaling. To learn more about inventory, see the Working with Inventory section.

**Managed nodes**

The network devices, servers, or both that you manage with Ansible. Managed nodes are also sometimes called “hosts”. Ansible is not installed on managed nodes.

### 1.3. INSTALLING RHEL SYSTEM ROLES IN YOUR SYSTEM

To use the RHEL System Roles, install the required packages in your system.

**Prerequisites**

- You have Ansible packages installed in the system you want to use as a control node:

**Procedure**

1. Install the `rhel-system-roles` package on the system that you want to use as a control node:
   
   ```bash
   # dnf install rhel-system-roles
   ```

2. Install the Ansible Core package:

   ```bash
   # dnf install ansible-core
   ```

The Ansible Core package provides the `ansible-playbook` CLI, the Ansible Vault functionality, and the basic modules and filters required by RHEL Ansible content.

As a result, you are able to create an Ansible playbook.

**Additional resources**

- The [Red Hat Enterprise Linux (RHEL) System Roles](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/system_roles)

- The `ansible-playbook` man page.

### 1.4. APPLYING A ROLE

The following procedure describes how to apply a particular role.

**Prerequisites**

- Ensure that the `rhel-system-roles` package is installed on the system that you want to use as a control node:

  ```bash
  # dnf install rhel-system-roles
  ```
1. Install the Ansible Core package:

```
# dnf install ansible-core
```

The Ansible Core package provides the `ansible-playbook` CLI, the Ansible Vault functionality, and the basic modules and filters required by RHEL Ansible content.

- Ensure that you are able to create an Ansible inventory.
  Inventories represent the hosts, host groups, and some of the configuration parameters used by the Ansible playbooks.

Playbooks are typically human-readable, and are defined in `ini`, `yaml`, `json`, and other file formats.

- Ensure that you are able to create an Ansible playbook.
  Playbooks represent Ansible’s configuration, deployment, and orchestration language. By using playbooks, you can declare and manage configurations of remote machines, deploy multiple remote machines or orchestrate steps of any manual ordered process.

A playbook is a list of one or more plays. Every play can include Ansible variables, tasks, or roles.

Playbooks are human-readable, and are defined in the `yaml` format.

**Procedure**

1. Create the required Ansible inventory containing the hosts and groups that you want to manage. Here is an example using a file called `inventory.ini` of a group of hosts called `webservers`:

```
[webservers]
host1
host2
host3
```

2. Create an Ansible playbook including the required role. The following example shows how to use roles through the `roles:` option for a playbook:

The following example shows how to use roles through the `roles:` option for a given play:

```
---
- hosts: webservers
  roles:
    - rhel-system-roles.network
    - rhel-system-roles.postfix
```
NOTE

Every role includes a README file, which documents how to use the role and supported parameter values. You can also find an example playbook for a particular role under the documentation directory of the role. Such documentation directory is provided by default with the `rhel-system-roles` package, and can be found in the following location:

```
/usr/share/doc/rhel-system-roles/SUBSYSTEM/
```

Replace `SUBSYSTEM` with the name of the required role, such as `postfix`, `metrics`, `network`, `tlog`, or `ssh`.

3. To execute the playbook on specific hosts, you must perform one of the following:

   - Edit the playbook to use `hosts: host1[,host2,...]`, or `hosts: all`, and execute the command:
     ```
     # ansible-playbook name.of.the.playbook
     ```
   - Edit the inventory to ensure that the hosts you want to use are defined in a group, and execute the command:
     ```
     # ansible-playbook -i name.of.the.inventory name.of.the.playbook
     ```
   - Specify all hosts when executing the `ansible-playbook` command:
     ```
     # ansible-playbook -i host1,host2,... name.of.the.playbook
     ```

IMPORTANT

Be aware that the `-i` flag specifies the inventory of all hosts that are available. If you have multiple targeted hosts, but want to select a host against which you want to run the playbook, you can add a variable in the playbook to be able to select a host. For example:

```
Ansible Playbook | example-playbook.yml:

- hosts: "{{ target_host }}"
  roles:
    - rhel-system-roles.network
    - rhel-system-roles.postfix
```

Playbook execution command:

```
# ansible-playbook -i host1,..hostn -e target_host=host5 example-playbook.yml
```

Additional resources

- Ansible playbooks
- Using roles in Ansible playbook
1.5. INTRODUCTION TO THE METRICS SYSTEM ROLE

RHEL System Roles is a collection of Ansible roles and modules that provide a consistent configuration interface to remotely manage multiple RHEL systems. The metrics System Role configures performance analysis services for the local system and, optionally, includes a list of remote systems to be monitored by the local system. The metrics System Role enables you to use `pcp` to monitor your systems performance without having to configure `pcp` separately, as the set-up and deployment of `pcp` is handled by the playbook.

Table 1.1. Metrics system role variables

<table>
<thead>
<tr>
<th>Role variable</th>
<th>Description</th>
<th>Example usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>metrics_monitored_hosts</code></td>
<td>List of remote hosts to be analyzed by the target host. These hosts will have metrics recorded on the target host, so ensure enough disk space exists below <code>/var/log</code> for each host.</td>
<td><code>metrics_monitored_hosts: [&quot;webserver.example.com&quot;, &quot;database.example.com&quot;]</code></td>
</tr>
<tr>
<td><code>metrics_retention_days</code></td>
<td>Configures the number of days for performance data retention before deletion.</td>
<td><code>metrics_retention_days: 14</code></td>
</tr>
<tr>
<td><code>metrics_graph_service</code></td>
<td>A boolean flag that enables the host to be set up with services for performance data visualization via <code>pcp</code> and <code>grafana</code>. Set to false by default.</td>
<td><code>metrics_graph_service: false</code></td>
</tr>
<tr>
<td><code>metrics_query_service</code></td>
<td>A boolean flag that enables the host to be set up with time series query services for querying recorded <code>pcp</code> metrics via <code>redis</code>. Set to false by default.</td>
<td><code>metrics_query_service: false</code></td>
</tr>
<tr>
<td><code>metrics_provider</code></td>
<td>Specifies which metrics collector to use to provide metrics. Currently, <code>pcp</code> is the only supported metrics provider.</td>
<td><code>metrics_provider: &quot;pcp&quot;</code></td>
</tr>
</tbody>
</table>

NOTE

For details about the parameters used in `metrics_connections` and additional information about the metrics System Role, see the `/usr/share/ansible/roles/rhel-system-roles.metrics/README.md` file.
1.6. USING THE METRICS SYSTEM ROLE TO MONITOR YOUR LOCAL SYSTEM WITH VISUALIZATION

This procedure describes how to use the metrics RHEL System Role to monitor your local system while simultaneously provisioning data visualization via grafana.

Prerequisites

- The Ansible Core package is installed on the control machine.
- You have the rhel-system-roles package installed on the machine you want to monitor.

Procedure

1. Configure localhost in the the /etc/ansible/hosts Ansible inventory by adding the following content to the inventory:

   localhost ansible_connection=local

2. Create an Ansible playbook with the following content:

   ```yaml
   ---
   - hosts: localhost
     vars:
       metrics_graph_service: yes
     roles:
       - rhel-system-roles.metrics
   ```

3. Run the Ansible playbook:

   ```bash
   # ansible-playbook name_of_your_playbook.yml
   ```

   **NOTE**

   Since the metrics_graph_service boolean is set to value="yes", grafana is automatically installed and provisioned with pcp added as a data source.

4. To view visualization of the metrics being collected on your machine, access the grafana web interface as described in Accessing the Grafana web UI.

1.7. USING THE METRICS SYSTEM ROLE TO SETUP A FLEET OF INDIVIDUAL SYSTEMS TO MONITOR THEMSELVES

This procedure describes how to use the metrics System Role to set up a fleet of machines to monitor themselves.

Prerequisites

- The Ansible Core package is installed on the control machine.
- You have the rhel-system-roles package installed on the machine you want to use to run the playbook.
Procedure

1. Add the name or IP of the machines you wish to monitor via the playbook to the `/etc/ansible/hosts` Ansible inventory file under an identifying group name enclosed in brackets:

   ```
   [remotes]
   webserver.example.com
   database.example.com
   ```

2. Create an Ansible playbook with the following content:

   ```
   ---
   - hosts: remotes
     vars:
       metrics_retention_days: 0
     roles:
       - rhel-system-roles.metrics
   ```

3. Run the Ansible playbook:

   ```
   # ansible-playbook name_of_your_playbook.yml
   ```

1.8. USING THE METRICS SYSTEM ROLE TO MONITOR A FLEET OF MACHINES CENTRALLY VIA YOUR LOCAL MACHINE

This procedure describes how to use the metrics System Role to set up your local machine to centrally monitor a fleet of machines while also provisioning visualization of the data via `grafana` and querying of the data via `redis`.

Prerequisites

- The Ansible Core package is installed on the control machine.
- You have the `rhel-system-roles` package installed on the machine you want to use to run the playbook.

Procedure

1. Create an Ansible playbook with the following content:

   ```
   ---
   - hosts: localhost
     vars:
       metrics_graph_service: yes
       metrics_query_service: yes
       metrics_retention_days: 10
       metrics_monitored_hosts: ["database.example.com", "webserver.example.com"]
     roles:
       - rhel-system-roles.metrics
   ```

2. Run the Ansible playbook:

   ```
   # ansible-playbook name_of_your_playbook.yml
   ```
Since the `metrics_graph_service` and `metrics_query_service` booleans are set to value="yes", Grafana is automatically installed and provisioned with `pcp` added as a data source with the `pcp` data recording indexed into `redis`, allowing the `pcp` querying language to be used for complex querying of the data.

3. To view graphical representation of the metrics being collected centrally by your machine and to query the data, access the Grafana web interface as described in Accessing the Grafana web UI.

### 1.9. Setting Up Authentication While Monitoring a System Using the Metrics System Role

PCP supports the `scram-sha-256` authentication mechanism through the Simple Authentication Security Layer (SASL) framework. The metrics RHEL System Role automates the steps to setup authentication using the `scram-sha-256` authentication mechanism. This procedure describes how to setup authentication using the metrics RHEL System Role.

**Prerequisites**

- The Ansible Core package is installed on the control machine.
- You have the `rhel-system-roles` package installed on the machine you want to use to run the playbook.

**Procedure**

1. Include the following variables in the Ansible playbook you want to setup authentication for:

```yaml
---
vars:
  metrics_username: your_username
  metrics_password: your_password
```

2. Run the Ansible playbook:

   ```
   # ansible-playbook name_of_your_playbook.yml
   ```

**Verification steps**

- Verify the `sasl` configuration:

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

### 1.10. Using the Metrics System Role to Configure and Enable Metrics Collection for SQL Server

This procedure describes how to use the metrics RHEL System Role to automate the configuration and enabling of metrics collection for Microsoft SQL Server via `pcp` on your local system.
Prerequisites

- The Ansible Core package is installed on the control machine.
- You have the `rhel-system-roles` package installed on the machine you want to monitor.
- 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. Configure `localhost` in the the `/etc/ansible/hosts` Ansible inventory by adding the following content to the inventory:

   ```
   localhost ansible_connection=local
   ```

2. Create an Ansible playbook that contains the following content:

   ```yaml
   ---
   - hosts: localhost
     roles:
       - role: rhel-system-roles.metrics
     vars:
       metrics_from_mssql: yes
   ```

3. Run the Ansible playbook:

   ```
   # ansible-playbook name_of_your_playbook.yml
   ```

Verification steps

- Use the `pcp` command to verify that SQL Server PMDA agent (mssql) is loaded and running:

  ```
  # pcp
  platform: Linux rhel82-2.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/rhel82-2.local/20200326.16.31
  pmie: primary engine: /var/log/pcp/pmie/rhel82-2.local/pmie.log
  ```

Additional resources

- For more information about using Performance Co-Pilot for Microsoft SQL Server, see this Red Hat Developers Blog post.
This documentation is installed automatically with the `rhel-system-roles` package.
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.

This section describes how to install and enable PCP on your system.

2.1. OVERVIEW OF PCP

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, which is useful during the centralized analysis of complex systems.
- It allows the monitoring and management of real-time data.
- It allows logging and retrieval of 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 package provides the command-line tools and underlying functionality.

- The pcp-gui package provides the graphical application. Install the pcp-gui package by executing the dnf install pcp-gui command. For more information, see Visually tracing PCP log archives with the PCP Charts application.

Additional resources

- pcp(1) man page
- /usr/share/doc/pcp-doc/ directory
- Tools distributed with PCP
- Index of Performance Co-Pilot (PCP) articles, solutions, tutorials, and white papers fromon Red Hat Customer Portal
- Side-by-side comparison of PCP tools with legacy tools Red Hat Knowledgebase article
- PCP upstream documentation

2.2. INSTALLING AND ENABLING PCP
To begin using PCP, install all the required packages and enable the PCP monitoring services.

This procedure describes how to install PCP using the `pcp` package. If you want to automate the PCP installation, install it using the `pcp-zeroconf` package. For more information on installing PCP by using `pcp-zeroconf`, see Setting up PCP with pcp-zeroconf.

Procedure

1. Install the `pcp` package:

   ```
   # {PackageManagerCommand} install pcp
   ```

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

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

Verification steps

- Verify if the `pmcd` process is running on the host:

  ```
  # pcp
  ```

Performance Co-Pilot configuration on workstation:

- platform: Linux workstation 4.18.0-80.el8.x86_64 #1 SMP Wed Mar 13 12:02:46 UTC 2019 x86_64
- hardware: 12 cpus, 2 disks, 1 node, 36023MB RAM
- timezone: CEST-2
- services: pmcd
- pmcd: Version 4.3.0-1, 8 agents
- pmda: root pmcd proc xfs linux mmv kvm jbd2

Additional resources

- `pmcd(1)` man page
- Tools distributed with PCP

2.3. 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 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. Stop the pmcd and pmlogger services:
   
   ```
   # systemctl stop pmcd.service
   # systemctl stop pmlogger.service
   ```

5. 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)
   ```

   Extract this file and analyze the data using PCP tools.

Additional resources

- `pmlogconf(1)`, `pmlogger(1)`, and `pmcd(1)` man pages
- Tools distributed with PCP
- System services distributed with PCP

## 2.4. SYSTEM SERVICES DISTRIBUTED WITH PCP

The following table describes roles of various system services, which are distributed with PCP.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pmcd</td>
<td>The Performance Metric Collector Daemon (PMCD).</td>
</tr>
<tr>
<td>pmie</td>
<td>The Performance Metrics Inference Engine.</td>
</tr>
<tr>
<td>pmlogger</td>
<td>The performance metrics logger.</td>
</tr>
</tbody>
</table>
pmproxy | The realtime and historical performance metrics proxy, time series query and REST API service.

2.5. TOOLS DISTRIBUTED WITH PCP

The following table describes usage of various tools, which are distributed with PCP.

Table 2.2. Usage of tools distributed with PCP

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pcp</td>
<td>Displays the current status of a Performance Co-Pilot installation.</td>
</tr>
<tr>
<td>pcp-atop</td>
<td>Shows the system-level occupation of the most critical hardware resources from the performance point of view: CPU, memory, disk, and network.</td>
</tr>
<tr>
<td>pcp-atopsar</td>
<td>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.</td>
</tr>
<tr>
<td>pcp-dmcache</td>
<td>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.</td>
</tr>
<tr>
<td>pcp-dstat</td>
<td>Displays metrics of one system at a time. To display metrics of multiple systems, use the --host option.</td>
</tr>
<tr>
<td>pcp-free</td>
<td>Reports on free and used memory in a system.</td>
</tr>
<tr>
<td>pcp-htop</td>
<td>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.</td>
</tr>
<tr>
<td>pcp-ipcs</td>
<td>Displays information on the inter-process communication (IPC) facilities that the calling process has read access for.</td>
</tr>
<tr>
<td>pcp-numastat</td>
<td>Displays NUMA allocation statistics from the kernel memory allocator.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>pcp-pidstat</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>pcp-ss</strong></td>
<td>Displays socket statistics collected by the pmdasockets Performance Metrics Domain Agent (PMDA).</td>
</tr>
<tr>
<td><strong>pcp-uptime</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>pcp-vmstat</strong></td>
<td>Provides a high-level system performance overview every 5 seconds. Displays information about processes, memory, paging, block I/O, traps, and CPU activity.</td>
</tr>
<tr>
<td><strong>pmchart</strong></td>
<td>Plots performance metrics values available through the facilities of the Performance Co-Pilot.</td>
</tr>
<tr>
<td><strong>pmclient</strong></td>
<td>Displays high-level system performance metrics by using the Performance Metrics Application Programming Interface (PMAPI).</td>
</tr>
<tr>
<td><strong>pmconfig</strong></td>
<td>Displays the values of configuration parameters.</td>
</tr>
<tr>
<td><strong>pmdbg</strong></td>
<td>Displays available Performance Co-Pilot debug control flags and their values.</td>
</tr>
<tr>
<td><strong>pmdiff</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>pmdumplog</strong></td>
<td>Displays control, metadata, index, and state information from a Performance Co-Pilot archive file.</td>
</tr>
<tr>
<td><strong>pmdumptext</strong></td>
<td>Outputs the values of performance metrics collected live or from a Performance Co-Pilot archive.</td>
</tr>
<tr>
<td><strong>pmerr</strong></td>
<td>Displays available Performance Co-Pilot error codes and their corresponding error messages.</td>
</tr>
<tr>
<td><strong>pmfind</strong></td>
<td>Finds PCP services on the network.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>pmie</td>
<td>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.</td>
</tr>
<tr>
<td>pmieconf</td>
<td>Displays or sets configurable pmie variables.</td>
</tr>
<tr>
<td>pmiectl</td>
<td>Manages non-primary instances of pmie.</td>
</tr>
<tr>
<td>pminfo</td>
<td>Displays information about performance metrics. The metrics are collected either from a live system, or from a Performance Co-Pilot archive file.</td>
</tr>
<tr>
<td>pmiostat</td>
<td>Reports I/O statistics for SCSI devices (by default) or device-mapper devices (with the -x dm option).</td>
</tr>
<tr>
<td>pmlc</td>
<td>Interactively configures active pmlogger instances.</td>
</tr>
<tr>
<td>pmlogcheck</td>
<td>Identifies invalid data in a Performance Co-Pilot archive file.</td>
</tr>
<tr>
<td>pmlogconf</td>
<td>Creates and modifies a pmlogger configuration file.</td>
</tr>
<tr>
<td>pmlogctl</td>
<td>Manages non-primary instances of pmlogger.</td>
</tr>
<tr>
<td>pmloglabel</td>
<td>Verifies, modifies, or repairs the label of a Performance Co-Pilot archive file.</td>
</tr>
<tr>
<td>pmlogsummary</td>
<td>Calculates statistical information about performance metrics stored in a Performance Co-Pilot archive file.</td>
</tr>
<tr>
<td>pmprobe</td>
<td>Determines the availability of performance metrics.</td>
</tr>
<tr>
<td>pmrep</td>
<td>Reports on selected, easily customizable, performance metrics values.</td>
</tr>
<tr>
<td>pmsocks</td>
<td>Allows access to a Performance Co-Pilot hosts through a firewall.</td>
</tr>
<tr>
<td>pmstat</td>
<td>Periodically displays a brief summary of system performance.</td>
</tr>
<tr>
<td>pmstore</td>
<td>Modifies the values of performance metrics.</td>
</tr>
<tr>
<td>pmtrace</td>
<td>Provides a command line interface to the trace PMDA.</td>
</tr>
<tr>
<td>pmval</td>
<td>Displays the current value of a performance metric.</td>
</tr>
</tbody>
</table>
2.6. PCP DEPLOYMENT ARCHITECTURES

Performance Co-Pilot (PCP) offers many options to accomplish advanced setups. From the huge variety of possible architectures, this section describes how to scale your PCP deployment based on the recommended deployment set up by Red Hat, sizing factors, and configuration options.

PCP supports multiple deployment architectures, based on the scale of the PCP deployment.

Available scaling deployment setup variants:

**Localhost**

Each service runs locally on the monitored machine. When you start a service without any configuration changes, this is the default deployment. Scaling beyond the individual node is not possible in this case.

By default, the deployment setup for Redis is standalone, localhost. However, Redis 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 Redis cluster in the cloud, or to utilize a managed Redis cluster from a cloud vendor.

**Decentralized**

The only difference between localhost and decentralized setup is the centralized Redis 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 Redis instance.

![Figure 2.1. Decentralized logging](image)

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 Redis instance.

![Centralized logging - pmlogger farm](image)
Federated - multiple pmlogger farms

For large scale deployments, Red Hat recommends to 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 Redis instance.
NOTE

By default, the deployment setup for Redis is standalone, localhost. However, Redis 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 Redis cluster in the cloud, or to utilize a managed Redis cluster from a cloud vendor.
Additional resources

- `pcp(1)`, `pmlogger(1)`, `pmproxy(1)`, and `pmcd(1)` man pages
- Recommended deployment architecture

2.7. RECOMMENDED DEPLOYMENT ARCHITECTURE

The following table describes the recommended deployment architectures based on the number of monitored hosts.

<table>
<thead>
<tr>
<th>Number of hosts (N)</th>
<th>1-10</th>
<th>10-100</th>
<th>100-1000</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pmcd servers</strong></td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td><strong>pmlogger servers</strong></td>
<td>1 to N</td>
<td>N/10 to N</td>
<td>N/100 to N</td>
</tr>
<tr>
<td><strong>pmproxy servers</strong></td>
<td>1 to N</td>
<td>1 to N</td>
<td>N/100 to N</td>
</tr>
<tr>
<td>Redis servers</td>
<td>1 to N</td>
<td>1 to N/10</td>
<td>N/100 to N/10</td>
</tr>
<tr>
<td>Redis cluster</td>
<td>No</td>
<td>Maybe</td>
<td>Yes</td>
</tr>
<tr>
<td>Recommended</td>
<td>Localhost, Decentralized, or Centralized logging</td>
<td>Decentralized, Centralized logging, or Federated</td>
<td>Decentralized or Federated</td>
</tr>
<tr>
<td>deployment setup</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2.8. SIZING FACTORS

The following are the sizing factors required for scaling:

**Remote system size**

The number of CPUs, disks, network interfaces, and other hardware resources affects the amount of data collected by each `pmlogger` on the centralized logging host.

**Logged Metrics**

The number and types of logged metrics play an important role. 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 versus 155 MB with proc metrics - a factor of 10 times more. Additionally, the number of instances for each metric, for example the number of CPUs, block devices, and network interfaces also impacts the required storage capacity.

**Logging Interval**

The interval how often metrics are logged, affects the storage requirements. The expected daily PCP archive file sizes are written to the `pmlogger.log` file for each `pmlogger` instance. These values are uncompressed estimates. Since PCP archives compress very well, approximately 10:1, the actual long term disk space requirements can be determined for a particular site.

`pmlogrewrite`
After every PCP upgrade, the `pmlogrewrite` tool is executed and rewrites old archives if there were changes in the metric metadata from the previous version and the new version of PCP. This process duration scales linear with the number of archives stored.

Additional resources

- `pmlogrewrite(1)` and `pmlogger(1)` man pages

### 2.9. CONFIGURATION OPTIONS FOR PCP SCALING

The following are the configuration options, which are required for scaling:

#### sysctl and rlimit settings

When archive discovery is enabled, `pmproxy` requires four descriptors for every `pmlogger` that it is monitoring or log-tailing, along with the additional file descriptors for the service logs and `pmproxy` client sockets, if any. Each `pmlogger` process uses about 20 file descriptors for the remote `pmcd` socket, archive files, service logs, and others. In total, this can exceed the default 1024 soft limit on a system running around 200 `pmlogger` processes. The `pmproxy` service in `pcp-5.3.0` and later automatically increases the soft limit to the hard limit. On earlier versions of PCP, tuning is required if a high number of `pmlogger` processes are to be deployed, and this can be accomplished by increasing the soft or hard limits for `pmlogger`. For more information, see How to set limits (ulimit) for services run by systemd.

#### Local Archives

The `pmlogger` service stores metrics of local and remote `pmcds` in the `/var/log/pcp/pmlogger/` directory. To control the logging interval of the local system, update the `/etc/pcp/pmlogger/control.d/configfile` file and add `-t X` in the arguments, where `X` is the logging interval in seconds. To configure which metrics should be logged, execute `pmlogconf /var/lib/pcp/config/pmlogger/config.clienthostname`. This command deploys a configuration file with a default set of metrics, which can optionally be further customized. To specify retention settings, that is when to purge old PCP archives, update the `/etc/sysconfig/pmlogger_timers` file and specify `PMLOGGER_DAILY_PARAMS="-E -k X"`, where `X` is the amount of days to keep PCP archives.

#### Redis

The `pmproxy` service sends logged metrics from `pmlogger` to a Redis instance. The following are the available two options to specify the retention settings in the `/etc/pcp/pmproxy/pmproxy.conf` configuration file:

- `stream.expire` specifies the duration when stale metrics should be removed, that is metrics which were not updated in a specified amount of time in seconds.

- `stream.maxlen` specifies the maximum number of metric values for one metric per host. This setting should be the retention time divided by the logging interval, for example 20160 for 14 days of retention and 60s logging interval (60*60*24*14/60)

Additional resources

- `pmproxy(1)`, `pmlogger(1)`, and `sysctl(8)` man pages

### 2.10. EXAMPLE: ANALYZING THE CENTRALIZED LOGGING DEPLOYMENT

The following results were gathered on a centralized logging setup, also known as pmlogger farm.
The following results were gathered on a centralized logging setup, also known as pmlogger farm deployment, with a default `pcp-zeroconf 5.3.0` installation, where each remote host is an identical container instance running `pmcd` on a server with 64 CPU cores, 376 GB RAM, and one disk attached.

The logging interval is 10s, proc metrics of remote nodes are not included, and the memory values refer to the Resident Set Size (RSS) value.

**Table 2.4. Detailed utilization statistics for 10s logging interval**

<table>
<thead>
<tr>
<th>Number of Hosts</th>
<th>10</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCP Archives Storage per Day</td>
<td>91 MB</td>
<td>522 MB</td>
</tr>
<tr>
<td>pmlogger Memory</td>
<td>160 MB</td>
<td>580 MB</td>
</tr>
<tr>
<td>pmlogger Network per Day (In)</td>
<td>2 MB</td>
<td>9 MB</td>
</tr>
<tr>
<td>pmproxy Memory</td>
<td>1.4 GB</td>
<td>6.3 GB</td>
</tr>
<tr>
<td>Redis Memory per Day</td>
<td>2.6 GB</td>
<td>12 GB</td>
</tr>
</tbody>
</table>

**Table 2.5. Used resources depending on monitored hosts for 60s logging interval**

<table>
<thead>
<tr>
<th>Number of Hosts</th>
<th>10</th>
<th>50</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCP Archives Storage per Day</td>
<td>20 MB</td>
<td>120 MB</td>
<td>271 MB</td>
</tr>
<tr>
<td>pmlogger Memory</td>
<td>104 MB</td>
<td>524 MB</td>
<td>1049 MB</td>
</tr>
<tr>
<td>pmlogger Network per Day (In)</td>
<td>0.38 MB</td>
<td>1.75 MB</td>
<td>3.48 MB</td>
</tr>
<tr>
<td>pmproxy Memory</td>
<td>2.67 GB</td>
<td>5.5GB</td>
<td>9 GB</td>
</tr>
<tr>
<td>Redis Memory per Day</td>
<td>0.54 GB</td>
<td>2.65 GB</td>
<td>5.3 GB</td>
</tr>
</tbody>
</table>

**NOTE**

The `pmproxy` queues Redis requests and employs Redis pipelining to speed up Redis queries. This can result in high memory usage. For troubleshooting this issue, see Troubleshooting high memory usage.

**2.11. EXAMPLE: ANALYZING THE FEDERATED SETUP DEPLOYMENT**

The following results were observed on a federated setup, also known as multiple `pmlogger` farms, consisting of three centralized logging (`pmlogger` farm) setups, where each `pmlogger` farm was monitoring 100 remote hosts, that is 300 hosts in total.
This setup of the pmlogger farms is identical to the configuration mentioned in the Example: Analyzing the centralized logging deployment for 60s logging interval, except that the Redis servers were operating in cluster mode.

Table 2.6. Used resources depending on federated hosts for 60s logging interval

<table>
<thead>
<tr>
<th>PCP Archives Storage per Day</th>
<th>pmlogger Memory</th>
<th>Network per Day (In/Out)</th>
<th>pmproxy Memory</th>
<th>Redis Memory per Day</th>
</tr>
</thead>
<tbody>
<tr>
<td>277 MB</td>
<td>1058 MB</td>
<td>15.6 MB / 12.3 MB</td>
<td>6-8 GB</td>
<td>5.5 GB</td>
</tr>
</tbody>
</table>

Here, all values are per host. The network bandwidth is higher due to the inter-node communication of the Redis cluster.

2.12. TROUBLESHOOTING HIGH MEMORY USAGE

The following scenarios can result in high memory usage:

- The pmproxy process is busy processing new PCP archives and does not have spare CPU cycles to process Redis requests and responses.
- The Redis node or cluster is overloaded and cannot process incoming requests on time.

The pmproxy service daemon uses Redis streams and supports the configuration parameters, which are PCP tuning parameters and affects Redis memory usage and key retention. The /etc/pcp/pmproxy/pmproxy.conf file lists the available configuration options for pmproxy and the associated APIs.

This section describes how to troubleshoot high memory usage issue.

Prerequisites

1. Install the pcp-pmda-redis package:

   # {PackageManagerCommand} install pcp-pmda-redis

2. Install the redis PMDA:

   # cd /var/lib/pcp/pmdas/redis && ./Install

Procedure

- To troubleshoot high memory usage, execute the following command and observe the inflight column:

   $ pmrep :pmproxy

   backlog inflight reqs/s resp/s wait req err resp err changed throttled
   byte count count/s count/s s/s count/s count/s count/s count/s count/s
   14:59:08 0   0    N/A   N/A   N/A   N/A   N/A   N/A   N/A   N/A
   14:59:09 0   0    2268.9 2268.9 28   0    0    2.0   4.0
   14:59:10 0   0    0.0   0.0   0.0   0.0   0.0   0.0   0.0
   14:59:11 0   0    0.0   0.0   0.0   0.0   0.0   0.0   0.0

   This column shows how many Redis requests are in-flight, which means they are queued or sent.
This column shows how many Redis requests are in-flight, which means they are queued or sent, and no reply was received so far.

A high number indicates one of the following conditions:

- The pmproxy process is busy processing new PCP archives and does not have spare CPU cycles to process Redis requests and responses.
- The Redis node or cluster is overloaded and cannot process incoming requests on time.
- To troubleshoot the high memory usage issue, reduce the number of pmlogger processes for this farm, and add another pmlogger farm. Use the federated - multiple pmlogger farms setup. If the Redis node is using 100% CPU for an extended amount of time, move it to a host with better performance or use a clustered Redis setup instead.

- To view the pmproxy.redis.* metrics, use the following command:

```
$ pminfo -ftd pmproxy.redis
pmproxy.redis.responses.wait [wait time for responses]
  Data Type: 64-bit unsigned int  InDom: PM_INDOM_NULL 0xffffffff
  Semantics: counter  Units: microsec
  value 546028367374
pmproxy.redis.responses.error [number of error responses]
  Data Type: 64-bit unsigned int  InDom: PM_INDOM_NULL 0xffffffff
  Semantics: counter  Units: count
  value 1164
[...]
pmproxy.redis.requests.inflight.bytes [bytes allocated for inflight requests]
  Data Type: 64-bit int  InDom: PM_INDOM_NULL 0xffffffff
  Semantics: discrete  Units: byte
  value 0
pmproxy.redis.requests.inflight.total [inflight requests]
  Data Type: 64-bit unsigned int  InDom: PM_INDOM_NULL 0xffffffff
  Semantics: discrete  Units: count
  value 0
[...]
```

To view how many Redis requests are inflight, see the pmproxy.redis.requests.inflight.total metric and pmproxy.redis.requests.inflight.bytes metric to view how many bytes are occupied by all current inflight Redis requests.

In general, the redis request queue would be zero but can build up based on the usage of large pmlogger farms, which limits scalability and can cause high latency for pmproxy clients.

- Use the pminfo command to view information about performance metrics. For example, to view the redis.* metrics, use the following command:

```
$ pminfo -ftd redis
redis.redis_build_id [Build ID]
  Data Type: string  InDom: 24.0 0x6000000
  Semantics: discrete  Units: count
  inst [0 or "localhost:6379"] value "87e335e57cfa755"
redis.total_commands_processed [Total number of commands processed by the server]
  Data Type: 64-bit unsigned int  InDom: 24.0 0x6000000
  Semantics: counter  Units: count
```
To view the peak memory usage, see the `redis.used_memory_peak` metric.

### Additional resources

- `pmdaredis(1)`, `pmproxy(1)`, and `pminfo(1)` man pages
- `PCP deployment architectures`
CHAPTER 3. 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

3.1. 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:

   ```bash
   # 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.

Additional resources

- pmlogconf(1) and pmlogger(1) man pages
- Tools distributed with PCP
- System services distributed with PCP

3.2. EDITING THE PMLOGGER CONFIGURATION FILE 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 page

• Tools distributed with PCP

• System services distributed with PCP

3.3. ENABLING THE PMLOGGER SERVICE

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

This procedure describes how to enable the pmlogger service.

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 steps**

- Verify if the `pmlogger` service is enabled:

```bash
# pcp
```

Performance Co-Pilot configuration on workstation:

```
platform: Linux workstation 4.18.0-80.el8.x86_64 #1 SMP Wed Mar 13 12:02:46 UTC 2019
x86_64
hardware: 12 cpus, 2 disks, 1 node, 36023MB RAM
timezone: CEST-2
services: pmcd
pmcd: Version 4.3.0-1, 8 agents, 1 client
pmda: root pmcd proc xfs linux mmv kvm jbd2
pmlogger: primary logger: /var/log/pcp/pmlogger/workstation/20190827.15.54
```

**Additional resources**

- `pmlogger(1)` man page
- Tools distributed with PCP
- System services distributed with PCP
- `/var/lib/pcp/config/pmlogger/config.default` file

### 3.4. SETTING UP A CLIENT SYSTEM FOR METRICS COLLECTION

This procedure describes how to set up a client system so that a central server can collect 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:
   ```bash
   # {PackageManagerCommand} install pcp-system-tools
   ```

2. Configure an IP address for `pmcd`:
   ```bash
   # 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:

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

# firewall-cmd --reload
success
```

4. Set an SELinux boolean:

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

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

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

**Verification steps**

- Verify if the `pmcd` is correctly listening on the configured IP address:

```bash
# ss -t -lp | 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
- Tools distributed with PCP
- System services distributed with PCP
- `/var/lib/pcp/config/pmlogger/config.default` file

**3.5. SETTING UP A CENTRAL SERVER TO COLLECT DATA**

This procedure describes how to 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:

```bash
# {PackageManagerCommand} install pcp-system-tools
```
2. Create the `/etc/pcp/pmlogger/control.d/remote` file with the following content:

```bash
# DO NOT REMOVE OR EDIT THE FOLLOWING LINE
$version=1.1

192.168.4.13 n n PCP_LOG_DIR/pmlogger/rhel7u4a -r -T24h10m -c config.rhel7u4a
192.168.4.14 n n PCP_LOG_DIR/pmlogger/rhel6u10a -r -T24h10m -c config.rhel6u10a
192.168.4.62 n n PCP_LOG_DIR/pmlogger/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:

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

Verification steps

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

```bash
# 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
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 page
- Tools distributed with PCP
- System services distributed with PCP
- `/var/lib/pcp/config/pmlogger/config.default` file

3.6. REPLAYING THE PCP LOG ARCHIVES WITH PMDUMPTEXT

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 `pmdumptext`, `pmrep`, or `pmlogsummary`.

Using the `pmdumptext` 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.
• Install the pcp-gui package:
  # {PackageManagerCommand} install pcp-gui

Procedure
• Display the data on the metric:
  $ pmdumptext -t 5seconds -H -a 20170605 xfs.perdev.log.writes

  Time local::xfs.perdev.log.writes["/dev/mapper/fedora-home"]
  local::xfs.perdev.log.writes["/dev/mapper/fedora-root"]
  ? 0.000 0.000
  none count / second count / second
  Mon Jun 5 12:28:45 ? ?
  Mon Jun 5 12:28:50 0.000 0.000
  Mon Jun 5 12:28:55 0.200 0.200
  Mon Jun 5 12:29:00 6.800 1.000

  The mentioned example displays the data on the xfs.perdev.log metric collected in an archive at a 5 second interval and display all the headers.

  NOTE
  Replace 20170605 in this example with a filename containing the pmlogger archive you want to display data for.

Additional resources
• pmlogger(1), pmdumptext(1), pmrep(1), and pmlogsummary(1) man pages
• Tools distributed with PCP
• System services distributed with PCP
CHAPTER 4. 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 using the the PCP application in Red Hat Enterprise Linux 9.

4.1. MONITORING POSTFIX WITH PMDA-POSTFIX

This procedure describes how to 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:
      ```
      # (PackageManagerCommand) install pcp-system-tools
      ```
   b. Install the pmda-postfix package to monitor postfix:
      ```
      # (PackageManagerCommand) install pcp-pmda-postfix postfix
      ```
   c. Install the logging daemon:
      ```
      # (PackageManagerCommand) install rsyslog
      ```
   d. Install the mail client for testing:
      ```
      # (PackageManagerCommand) 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/
   ```
Verification steps

- Verify the `pmda-postfix` operation:
  
  ```bash
echo testmail | mutt root
  ```

- Verify the available metrics:

  ```bash
  # 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
- Tools distributed with PCP
- System services distributed with PCP
- `/var/lib/pcp/config/pmlogger/config.default` file

4.2. 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.

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.
- Install the pcp-gui package:

  ```
  # {PackageManagerCommand} install pcp-gui
  ```

Procedure

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

  ```
  # pmchart
  ```

Figure 4.1. PCP Charts application

The pmtime server settings are located at the bottom. The start and pause button allows you to 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. 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**:

```plaintext
#kmchart
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
- **Tools distributed with PCP**

### 4.3. COLLECTING DATA FROM SQL SERVER USING PCP

The SQL Server agent is available in Performance Co-Pilot (PCP), which helps you to monitor and analyze database performance issues.

This procedure describes how to 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:

   ```bash
   # {PackageManagerCommand} install pcp-zeroconf
   ```

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

   ```bash
   # {PackageManagerCommand} install python3-pyodbc
   ```

3. Install the **mssql** agent:

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

      ```bash
      # {PackageManagerCommand} 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 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 steps**

- 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 Visualy tracing PCP log archives with the PCP Charts application.

**Additional resources**

- `pcp(1)`, `pminfo(1)`, `pmval(1)`, `pmchart(1)`, and `pmdamssql(1)` man pages
Performance Co-Pilot for Microsoft SQL Server with RHEL 8.2 Red Hat Developers Blog post
CHAPTER 5. PERFORMANCE ANALYSIS OF XFS WITH PCP

The XFS PMDA ships as part of the pcp package and is enabled by default during the installation. It is used to gather performance metric data of XFS file systems in Performance Co-Pilot (PCP).

This section describes how to analyze XFS file system’s performance using PCP.

5.1. INSTALLING XFS PMDA MANUALLY

If the XFS PMDA is not listed in the pcp configuration output, install the PMDA agent manually.

This procedure describes how to manually install the PMDA agent.

Prerequisites

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

Procedure

1. Navigate to the xfs directory:

```
# cd /var/lib/pcp/pmdas/xfs/
```

Verification steps

- Verify that the pmcd process is running on the host and the XFS PMDA is listed as enabled in the configuration:

```
# pcp
Performance Co-Pilot configuration on workstation:
platform: Linux workstation 4.18.0-80.el8.x86_64 #1 SMP Wed Mar 13 12:02:46 UTC 2019
x86_64
hardware: 12 cpus, 2 disks, 1 node, 36023MB RAM
timezone: CEST-2
services: pmcd
pmcd: Version 4.3.0-1, 8 agents
pmda: root pmcd proc xfs linux mmv kvm jbd2
```

Additional resources

- pmcd(1) man page
- Tools distributed with PCP

5.2. EXAMINING XFS PERFORMANCE METRICS WITH PMINFO

PCP enables XFS PMDA to allow the reporting of certain XFS metrics per each of the mounted XFS file systems. This makes it easier to pinpoint specific mounted file system issues and evaluate performance.

The pminfo command provides per-device XFS metrics for each mounted XFS file system.
This procedure displays a list of all available metrics provided by the XFS PMDA.

Prerequisites

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

Procedure

- Display the list of all available metrics provided by the XFS PMDA:

  ```
  # pminfo xfs
  ```

- Display information for the individual metrics. The following examples examine specific XFS read and write metrics using the `pminfo` tool:

  - Display a short description of the `xfs.write_bytes` metric:

    ```
    # pminfo --oneline xfs.write_bytes
    xfs.write_bytes [number of bytes written in XFS file system write operations]
    ```

  - Display a long description of the `xfs.read_bytes` metric:

    ```
    # pminfo --helptext xfs.read_bytes
    xfs.read_bytes
    Help:
    This is the number of bytes read via read(2) system calls to files in XFS file systems. It can be used in conjunction with the read_calls count to calculate the average size of the read operations to file in XFS file systems.
    ```

  - Obtain the current performance value of the `xfs.read_bytes` metric:

    ```
    # pminfo --fetch xfs.read_bytes
    xfs.read_bytes
    value 4891346238
    ```

  - Obtain per-device XFS metrics with `pminfo`:

    ```
    # pminfo --fetch --oneline xfs.perdev.read xfs.perdev.write
    xfs.perdev.read [number of XFS file system read operations]
    inst [0 or "loop1"] value 0
    inst [0 or "loop2"] value 0
    
    xfs.perdev.write [number of XFS file system write operations]
    inst [0 or "loop1"] value 86
    inst [0 or "loop2"] value 0
    ```

Additional resources
5.3. RESETTING XFS PERFORMANCE METRICS WITH PMSTORE

With PCP, you can modify the values of certain metrics, especially if the metric acts as a control variable, such as the `xfs.control.reset` metric. To modify a metric value, use the `pmstore` tool.

This procedure describes how to reset XFS metrics using the `pmstore` tool.

**Prerequisites**

- PCP is installed. For more information, see [Installing and enabling PCP](#).

**Procedure**

1. Display the value of a metric:
   
   ```
   $ pminfo -f xfs.write
   xfs.write
   value 325262
   ```

2. Reset all the XFS metrics:
   
   ```
   # pmstore xfs.control.reset 1
   xfs.control.reset old value=0 new value=1
   ```

**Verification steps**

- View the information after resetting the metric:
  
  ```
  $ pminfo --fetch xfs.write
  xfs.write
  value 0
  ```

**Additional resources**

- `pmstore(1)` and `pminfo(1)` man pages

**5.4. PCP METRIC GROUPS FOR XFS**

The following table describes the available PCP metric groups for XFS.
Table 5.1. Metric groups for XFS

<table>
<thead>
<tr>
<th>Metric Group</th>
<th>Metrics provided</th>
</tr>
</thead>
<tbody>
<tr>
<td>xfs.*</td>
<td>General XFS metrics including the read and write operation counts, read and write byte counts. Along with counters for the number of times inodes are flushed, clustered and number of failure to cluster.</td>
</tr>
<tr>
<td>xfs.allocs.*</td>
<td>Range of metrics regarding the allocation of objects in the file system, these include number of extent and block creations/frees. Allocation tree lookup and compares along with extend record creation and deletion from the btree.</td>
</tr>
<tr>
<td>xfs.alloc_btree.*</td>
<td></td>
</tr>
<tr>
<td>xfs.block_map.*</td>
<td>Metrics include the number of block map read/write and block deletions, extent list operations for insertion, deletions and lookups. Also operations counters for compares, lookups, insertions and deletion operations from the blockmap.</td>
</tr>
<tr>
<td>xfs.bmap_btree.*</td>
<td></td>
</tr>
<tr>
<td>xfs.dir_ops.*</td>
<td>Counters for directory operations on XFS file systems for creation, entry deletions, count of “getdent” operations.</td>
</tr>
<tr>
<td>xfs.transactions.*</td>
<td>Counters for the number of meta-data transactions, these include the count for the number of synchronous and asynchronous transactions along with the number of empty transactions.</td>
</tr>
<tr>
<td>xfs.inode_ops.*</td>
<td>Counters for the number of times that the operating system looked for an XFS inode in the inode cache with different outcomes. These count cache hits, cache misses, and so on.</td>
</tr>
<tr>
<td>xfs.log.*</td>
<td>Counters for the number of log buffer writes over XFS file systems includes the number of blocks written to disk. Metrics also for the number of log flushes and pinning.</td>
</tr>
<tr>
<td>xfs.log_tail.*</td>
<td></td>
</tr>
<tr>
<td>xfs.xstrat.*</td>
<td>Counts for the number of bytes of file data flushed out by the XFS flush deamon along with counters for number of buffers flushed to contiguous and non-contiguous space on disk.</td>
</tr>
<tr>
<td>xfs.attr.*</td>
<td>Counts for the number of attribute get, set, remove and list operations over all XFS file systems.</td>
</tr>
<tr>
<td>xfs.quota.*</td>
<td>Metrics for quota operation over XFS file systems, these include counters for number of quota reclams, quota cache misses, cache hits and quota data reclams.</td>
</tr>
<tr>
<td>Metric Group</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>xfs.buffer.*</td>
<td>Range of metrics regarding XFS buffer objects. Counters include the number of requested buffer calls, successful buffer locks, waited buffer locks, miss_locks, miss_retries and buffer hits when looking up pages.</td>
</tr>
<tr>
<td>xfs.btree.*</td>
<td>Metrics regarding the operations of the XFS btree.</td>
</tr>
<tr>
<td>xfs.control.reset</td>
<td>Configuration metrics which are used to reset the metric counters for the XFS stats. Control metrics are toggled by means of the pmstore tool.</td>
</tr>
</tbody>
</table>

**5.5. PER-DEVICE PCP METRIC GROUPS FOR XFS**

The following table describes the available per-device PCP metric group for XFS.

<table>
<thead>
<tr>
<th>Metric Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xfs.perdev.*</td>
<td>General XFS metrics including the read and write operation counts, read and write byte counts. Along with counters for the number of times inodes are flushed, clustered and number of failure to cluster.</td>
</tr>
<tr>
<td>xfs.perdev.allocs.*</td>
<td>Range of metrics regarding the allocation of objects in the file system, these include number of extent and block creations/frees. Allocation tree lookup and compares along with extend record creation and deletion from the btree.</td>
</tr>
<tr>
<td>xfs.perdev.alloc_btree.*</td>
<td>Range of metrics regarding the allocation of objects in the file system, these include number of extent and block creations/frees. Allocation tree lookup and compares along with extend record creation and deletion from the btree.</td>
</tr>
<tr>
<td>xfs.perdev.block_map.*</td>
<td>Metrics include the number of block map read/write and block deletions, extent list operations for insertion, deletions and lookups. Also operations counters for compares, lookups, insertions and deletion operations from the blockmap.</td>
</tr>
<tr>
<td>xfs.perdev.bmap_btree.*</td>
<td>Metrics include the number of block map read/write and block deletions, extent list operations for insertion, deletions and lookups. Also operations counters for compares, lookups, insertions and deletion operations from the blockmap.</td>
</tr>
<tr>
<td>xfs.perdev.dir_ops.*</td>
<td>Counters for directory operations of XFS file systems for creation, entry deletions, count of “getdent” operations.</td>
</tr>
<tr>
<td>xfs.perdev.transactions.*</td>
<td>Counters for the number of meta-data transactions, these include the count for the number of synchronous and asynchronous transactions along with the number of empty transactions.</td>
</tr>
<tr>
<td>xfs.perdev.inode_ops.*</td>
<td>Counters for the number of times that the operating system looked for an XFS inode in the inode cache with different outcomes. These count cache hits, cache misses, and so on.</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>xfs.perdev.log.*</td>
<td>Counters for the number of log buffer writes over XFS filesystems includes the number of blocks written to disk. Metrics also for the number of log flushes and pinning.</td>
</tr>
<tr>
<td>xfs.perdev.log_tail.*</td>
<td></td>
</tr>
<tr>
<td>xfs.perdev.xstrat.*</td>
<td>Counts for the number of bytes of file data flushed out by the XFS flush daemon along with counters for number of buffers flushed to contiguous and non-contiguous space on disk.</td>
</tr>
<tr>
<td>xfs.perdev.attr.*</td>
<td>Counts for the number of attribute get, set, remove and list operations over all XFS file systems.</td>
</tr>
<tr>
<td>xfs.perdev.quota.*</td>
<td>Metrics for quota operation over XFS file systems, these include counters for number of quota reclaims, quota cache misses, cache hits and quota data reclaims.</td>
</tr>
<tr>
<td>xfs.perdev.buffer.*</td>
<td>Range of metrics regarding XFS buffer objects. Counters include the number of requested buffer calls, successful buffer locks, waited buffer locks, miss_locks, miss_retries and buffer hits when looking up pages.</td>
</tr>
<tr>
<td>xfs.perdev.btree.*</td>
<td>Metrics regarding the operations of the XFS btree.</td>
</tr>
</tbody>
</table>
CHAPTER 6. SETTING UP GRAPHICAL REPRESENTATION OF PCP METRICS

Using a combination of `pcp`, `grafana`, `pcp redis`, `pcp bpftrace`, and `pcp vector` provides graphs, based on the live data or data collected by Performance Co-Pilot (PCP).

This section describes how to set up and access the graphical representation of PCP metrics.

6.1. SETTING UP PCP WITH PCP-ZEROCONF

This procedure describes how to 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:

  ```
  # {PackageManagerCommand} install pcp-zeroconf
  ```

Verification steps

- 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
- 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` protocol in the backend.

This procedure describes how to set up a `grafana-server`.

Prerequisites

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

Procedure

1. Install the following packages:

   ```
   # {PackageManagerCommand} install grafana grafana-pcp
   ```

2. Restart and enable the following service:
# systemctl restart grafana-server
# systemctl enable grafana-server

Verification steps

- 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 @ 3.1.0
  ```

Additional resources

- `pmproxy(1)` and `grafana-server` man pages

### 6.3. ACCESSING THE GRAFANA WEB UI

This procedure describes how to access the Grafana web interface.

Using the Grafana web interface, you can:

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

Prerequisites

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

2. 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, using `http://192.0.2.0:3000` link. Replace `192.0.2.0` with your machine IP.

2. For the first login, enter `admin` in both the Email or username and Password field. Grafana prompts to set a New password to create a secured account. If you want to set it later, click Skip.

3. From the menu, hover over the Configuration icon and then click Plugins.
4. In the Plugins tab, type performance co-pilot in the Search by name or type text box and then click Performance Co-Pilot (PCP) plugin.

5. In the Plugins / Performance Co-Pilot pane, click Enable.

6. Click Grafana icon. The Grafana Home page is displayed.

![Figure 6.1. Home Dashboard](image)

**NOTE**

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

7. In the Grafana Home page, click Add your first data source to add PCP Redis, PCP bpftrace, and PCP Vector data sources. For more information on adding data source, see:

- To add pcp redis data source, view default dashboard, create a panel, and an alert rule, see Creating panels and alert in PCP Redis 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.

8. 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

### 6.4. CONFIGURING PCP REDIS
This section provides information for configuring PCP Redis data source.

Use the PCP Redis data source to:

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

**Prerequisites**

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

**Procedure**

1. Install the **redis** package:
   
   ```
   # {PackageManagerCommand} install redis
   ```

2. Start and enable the following services:
   
   ```
   # systemctl start pmproxy redis
   # systemctl enable pmproxy redis
   ```

3. Mail transfer agent, for example, **sendmail** or **postfix** is installed and configured.

4. Ensure that the **allow_loading_unsigned_plugins** parameter is set to PCP Redis database in the **grafana.ini** file:
   
   ```
   # vi /etc/grafana/grafana.ini
   
   allow_loading_unsigned_plugins = pcp-redis-datasource
   ```

5. Restart the **grafana-server**:
   
   ```
   # systemctl restart grafana-server
   ```

**Verification steps**

- Ensure that the **pmproxy** and **redis** are working:
  
  ```
  # pmseries disk.dev.read
  2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
  ```

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

**Additional resources**

- **pmseries(1)** man page
6.5. CREATING PANELS AND ALERT IN PCP REDIS DATA SOURCE

After adding the PCP Redis 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

1. The PCP Redis is configured. For more information, see Configuring PCP Redis.
2. 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 redis in the Filter by name or type text box and then click PCP Redis.
4. In the Data Sources / PCP Redis pane, perform the following:
   a. Add http://localhost:44322 in the URL field and then click Save & Test.
   b. Click Dashboards tab → Import → PCP Redis: Host Overview to see a dashboard with an overview of any useful metrics.

Figure 6.2. PCP Redis: Host Overview

5. Add a new panel:
   a. From the menu, hover over the icon to add a panel.
   b. In the Query tab, select the PCP Redis 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. PCP Redis query panel**

![PCP Redis query panel](image)

6. Create an alert rule:

   a. In the **PCP Redis 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 PCP Redis panel**

   ![Creating alerts in the PCP Redis panel](image)

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

### 6.6. 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**

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

```bash
# vi /etc/grafana/grafana.ini

[smtp]
enabled = true
from_address = abc@gmail.com

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

**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.

d. Click Apply.

Additional resources

- Upstream Grafana documentation for alert notifications

### 6.7. SETTING UP AUTHENTICATION BETWEEN PCP COMPONENTS

You can setup 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:

   ```
   # {PackageManagerCommand} 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, you are 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 steps

- Verify the `sasl` configuration:

  ```bash
  # 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

- How can I setup authentication between PCP components, like PMDAs and pmcd in RHEL 8.2?

## 6.8. INSTALLING PCP BPFTRACE

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).

This procedure describes how to install a `pcp bpftrace`.

### Prerequisites

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

2. The `grafana-server` is configured. For more information, see Setting up a grafana-server.

3. 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:

   ```bash
   # {PackageManagerCommand} install pcp-pmda-bpftrace
   ```

2. Edit the `bpftrace.conf` file and add the user that you have created in the [setting-up-authentication-between-pcp-components]:

   ```bash
   # 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

### 6.9. 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**

1. The PCP bpftrace is installed. For more information, see Installing PCP bpftrace.

2. 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 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**.
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

6.10. INSTALLING PCP VECTOR

This procedure describes how to install a **pcp vector**.

**Prerequisites**

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

2. The **grafana-server** is configured. For more information, see **Setting up a grafana-server**.

**Procedure**

1. Install the **pcp-pmda-bcc** package:
# dnf install pcp-pmda-bcc

2. 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

### 6.11. 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

1. The PCP Vector is installed. For more information, see Installing PCP Vector.

2. 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.
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.

### 6.12. TROUBLESHOOTING GRAFANA ISSUES

This section describes how 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:

  ```bash
  $ systemctl status pmlogger
  ```
• Verify if files were created or modified to the disk by executing the following command:

```bash
$ 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:

```bash
$ systemctl status pmproxy
```

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

```
pmproxy(1716) Info: Redis 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 Redis database contains any keys by executing the following command:

```bash
$ redis-cli dbsize
(integer) 34837
```

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

```bash
$ pmseries disk.dev.read
2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df

$ pmseries "disk.dev.read[\text{count:10}]"
2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
  70e83e88d4e1857a3a31605c6d1333755f2dd17c
  [Mon Jul 26 12:21:00.087401000 2021] 117758
  70e83e88d4e1857a3a31605c6d1333755f2dd17c
  70e83e88d4e1857a3a31605c6d1333755f2dd17c
  ...

$ redis-cli --scan --pattern "*$(pmseries 'disk.dev.read')"
```

• 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...
  lvl=info msg="Starting Grafana" logger=server version=7.3.6
  lvl=info msg="Config loaded from" logger=settings file=/usr/s
  lvl=info msg="Config loaded from" logger=settings file=/etc/g
[...]
CHAPTER 7. CONFIGURING AN OPERATING SYSTEM TO OPTIMIZE CPU UTILIZATION

This section describes how to configure the operating system to optimize CPU utilization across their workloads.

7.1. TOOLS FOR MONITORING AND DIAGNOSING PROCESSOR ISSUES

The following are the tools available in Red Hat Enterprise Linux 9 to monitor and diagnose processor-related performance issues:

- **turbostat** tool prints counter results at specified intervals to help administrators identify unexpected behavior in servers, such as excessive power usage, failure to enter deep sleep states, or system management interrupts (SMIs) being created unnecessarily.

- **numactl** utility provides a number of options to manage processor and memory affinity. The numactl package includes the **libnuma** library which offers a simple programming interface to the NUMA policy supported by the kernel, and can be used for more fine-grained tuning than the numactl application.

- **numastat** tool displays per-NUMA node memory statistics for the operating system and its processes, and shows administrators whether the process memory is spread throughout a system or is centralized on specific nodes. This tool is provided by the numactl package.

- **numad** is an automatic NUMA affinity management daemon. It monitors NUMA topology and resource usage within a system in order to dynamically improve NUMA resource allocation and management.

- **/proc/interrupts** file displays the interrupt request (IRQ) number, the number of similar interrupt requests handled by each processor in the system, the type of interrupt sent, and a comma-separated list of devices that respond to the listed interrupt request.

- **pqos** utility is available in the intel-cmt-cat package. It monitors CPU cache and memory bandwidth on recent Intel processors. It monitors:
  - The instructions per cycle (IPC).
  - The count of last level cache MISSES.
  - The size in kilobytes that the program executing in a given CPU occupies in the LLC.
  - The bandwidth to local memory (MBL).
  - The bandwidth to remote memory (MBR).

- **x86_energy_perf_policy** tool allows administrators to define the relative importance of performance and energy efficiency. This information can then be used to influence processors that support this feature when they select options that trade off between performance and energy efficiency.

- **taskset** tool is provided by the util-linux package. It allows administrators to retrieve and set the processor affinity of a running process, or launch a process with a specified processor affinity.

**Additional resources**

- turbostat(8), numactl(8), numastat(8), numa(7), numad(8), pqos(8)
7.2. TYPES OF SYSTEM TOPOLOGY

In modern computing, the idea of a CPU is a misleading one, as most modern systems have multiple processors. The topology of the system is the way these processors are connected to each other and to other system resources. This can affect system and application performance, and the tuning considerations for a system.

The following are the two primary types of topology used in modern computing:

Symmetric Multi-Processor (SMP) topology

SMP topology allows all processors to access memory in the same amount of time. However, because shared and equal memory access inherently forces serialized memory accesses from all the CPUs, SMP system scaling constraints are now generally viewed as unacceptable. For this reason, practically all modern server systems are NUMA machines.

Non-Uniform Memory Access (NUMA) topology

NUMA topology was developed more recently than SMP topology. In a NUMA system, multiple processors are physically grouped on a socket. Each socket has a dedicated area of memory and processors that have local access to that memory, these are referred to collectively as a node. Processors on the same node have high speed access to that node’s memory bank, and slower access to memory banks not on their node.

Therefore, there is a performance penalty when accessing non-local memory. Thus, performance sensitive applications on a system with NUMA topology should access memory that is on the same node as the processor executing the application, and should avoid accessing remote memory wherever possible.

Multi-threaded applications that are sensitive to performance may benefit from being configured to execute on a specific NUMA node rather than a specific processor. Whether this is suitable depends on your system and the requirements of your application. If multiple application threads access the same cached data, then configuring those threads to execute on the same processor may be suitable. However, if multiple threads that access and cache different data execute on the same processor, each thread may evict cached data accessed by a previous thread. This means that each thread ‘misses’ the cache and wastes execution time fetching data from memory and replacing it in the cache. Use the perf tool to check for an excessive number of cache misses.

7.2.1. Displaying system topologies

There are a number of commands that help understand the topology of a system. This procedure describes how to determine the system topology.

Procedure

- To display an overview of your system topology:

```
$ numactl --hardware
available: 4 nodes (0-3)
node 0 cpus: 0 4 8 12 16 20 24 28 32 36
node 0 size: 65415 MB
node 0 free: 43971 MB
[...]```
To gather the information about the CPU architecture, such as the number of CPUs, threads, cores, sockets, and NUMA nodes:

```
$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                40
On-line CPU(s) list:   0-39
Thread(s) per core:    1
Core(s) per socket:    10
Socket(s):             4
NUMA node(s):          4
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 47
Model name:            Intel(R) Xeon(R) CPU E7- 4870  @ 2.40GHz
Stepping:              2
CPU MHz:               2394.204
BogoMIPS:              4787.85
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              30720K
NUMA node0 CPU(s):     0,4,8,12,16,20,24,28,32,36
NUMA node1 CPU(s):     2,6,10,14,18,22,26,30,34,38
NUMA node2 CPU(s):     1,5,9,13,17,21,25,29,33,37
NUMA node3 CPU(s):     3,7,11,15,19,23,27,31,35,39
```

To view a graphical representation of your system:

```
# {PackageManagerCommand} install hwloc-gui
# lstopo
```
To view the detailed textual output:

```bash
# {PackageManagerCommand} install hwloc
# lstopo-no-graphics
Machine (15GB)
Package L#0 + L3 L#0 (8192KB)
  L2 L#0 (256KB) + L1d L#0 (32KB) + L1i L#0 (32KB) + Core L#0
    PU L#0 (P#0)
    PU L#1 (P#4)
  HostBridge L#0
PCI 8086:5917
    GPU L#0 "renderD128"
    GPU L#1 "controlD64"
    GPU L#2 "card0"
PCIBridge
  PCI 8086:24fd
    Net L#3 "wlp61s0"
PCIBridge
  PCI 8086:f1a6
PCI 8086:15d7
  Net L#4 "enp0s31f6"
```

Additional resources

- `numactl(8), lscpu(1), and lstopo(1)` man pages

### 7.3. CONFIGURING KERNEL TICK TIME
By default, Red Hat Enterprise Linux 9 uses a tickless kernel, which does not interrupt idle CPUs in order to reduce power usage and allow new processors to take advantage of deep sleep states.

Red Hat Enterprise Linux 9 also offers a dynamic tickless option, which is useful for latency-sensitive workloads, such as high performance computing or realtime computing. By default, the dynamic tickless option is disabled. Red Hat recommends using the `cpu-partitioning TuneD` profile to enable the dynamic tickless option for cores specified as `isolated_cores`.

This procedure describes how to manually persistently enable dynamic tickless behavior.

**Procedure**

1. To enable dynamic tickless behavior in certain cores, specify those cores on the kernel command line with the `nohz_full` parameter. On a 16 core system, append this parameter on the `GRUB_CMDLINE_LINUX` option in the `/etc/default/grub` file:

   ```
   nohz_full=1-15
   ```

   This enables dynamic tickless behavior on cores 1 through 15, moving all timekeeping to the only unspecified core (core 0).

2. To persistently enable the dynamic tickless behavior, regenerate the GRUB2 configuration using the edited default file. On systems with BIOS firmware, execute the following command:

   ```
   # grub2-mkconfig -o /etc/grub2.cfg
   ```

   On systems with UEFI firmware, execute the following command:

   ```
   # grub2-mkconfig -o /etc/grub2-efi.cfg
   ```

3. When the system boots, manually move the `rcu` threads to the non-latency-sensitive core, in this case core 0:

   ```
   # for i in `pgrep rcu[^c]` ; do taskset -pc 0 $i ; done
   ```

4. Optional: Use the `isolcpus` parameter on the kernel command line to isolate certain cores from user-space tasks.

5. Optional: Set the CPU affinity for the kernel’s `write-back bdi-flush` threads to the housekeeping core:

   ```
   echo 1 > /sys/bus/workqueue/devices/writeback/cpumask
   ```

**Verification steps**

- Once the system is rebooted, verify if `dynticks` are enabled:

  ```
  # journalctl -xe | grep dynticks
  ```

- Verify that the dynamic tickless configuration is working correctly:

  ```
  # perf stat -C 1 -e irq_vectors:local_timer_entry taskset -c 1 sleep 3
  ```
This command measures ticks on CPU 1 while telling CPU 1 to sleep for 3 seconds.

- The default kernel timer configuration shows around 3100 ticks on a regular CPU:

  ```bash
  # perf stat -C 0 -e irq_vectors:local_timer_entry taskset -c 0 sleep 3
  Performance counter stats for 'CPU(s) 0':
    3,107   irq_vectors:local_timer_entry
  3.001342790 seconds time elapsed
  ```

- With the dynamic tickless kernel configured, you should see around 4 ticks instead:

  ```bash
  # perf stat -C 1 -e irq_vectors:local_timer_entry taskset -c 1 sleep 3
  Performance counter stats for 'CPU(s) 1':
    4    irq_vectors:local_timer_entry
  3.001544078 seconds time elapsed
  ```

Additional resources

- `perf(1)` and `cpuset(7)` man pages
- All about nohz_full kernel parameter Red Hat Knowledgebase article
- How to verify the list of "isolated" and "nohz_full" CPU information from sysfs? Red Hat Knowledgebase article

7.4. OVERVIEW OF AN INTERRUPT REQUEST

An interrupt request or IRQ is a signal for immediate attention sent from a piece of hardware to a processor. Each device in a system is assigned one or more IRQ numbers which allow it to send unique interrupts. When interrupts are enabled, a processor that receives an interrupt request immediately pauses execution of the current application thread in order to address the interrupt request.

Because interrupt halts normal operation, high interrupt rates can severely degrade system performance. It is possible to reduce the amount of time taken by interrupts by configuring interrupt affinity or by sending a number of lower priority interrupts in a batch (coalescing a number of interrupts).

Interrupt requests have an associated affinity property, `smp_affinity`, which defines the processors that handle the interrupt request. To improve application performance, assign interrupt affinity and process affinity to the same processor, or processors on the same core. This allows the specified interrupt and application threads to share cache lines.

On systems that support interrupt steering, modifying the `smp_affinity` property of an interrupt request sets up the hardware so that the decision to service an interrupt with a particular processor is made at the hardware level with no intervention from the kernel.

7.4.1. Balancing interrupts manually
If your BIOS exports its NUMA topology, the `irqbalance` service can automatically serve interrupt requests on the node that is local to the hardware requesting service.

**Procedure**

1. Check which devices correspond to the interrupt requests that you want to configure.

2. Find the hardware specification for your platform. Check if the chipset on your system supports distributing interrupts.
   
   a. If it does, you can configure interrupt delivery as described in the following steps. Additionally, check which algorithm your chipset uses to balance interrupts. Some BIOSes have options to configure interrupt delivery.
   
   b. If it does not, your chipset always routes all interrupts to a single, static CPU. You cannot configure which CPU is used.

3. Check which Advanced Programmable Interrupt Controller (APIC) mode is in use on your system:
   
   ```
   $ journalctl --dmesg | grep APIC
   ```
   
   Here,
   
   - If your system uses a mode other than `flat`, you can see a line similar to `Setting APIC routing to physical flat`.
   
   - If you can see no such message, your system uses `flat` mode. If your system uses `x2apic` mode, you can disable it by adding the `nox2apic` option to the kernel command line in the bootloader configuration.
     
     Only non-physical flat mode (`flat`) supports distributing interrupts to multiple CPUs. This mode is available only for systems that have up to 8 CPUs.

4. Calculate the `smp_affinity mask`. For more information on how to calculate the `smp_affinity mask`, see Setting the `smp_affinity mask`.

**Additional resources**

- `journalctl(1)` and `taskset(1)` man pages

### 7.4.2. Setting the `smp_affinity mask`

The `smp_affinity` value is stored as a hexadecimal bit mask representing all processors in the system. Each bit configures a different CPU. The least significant bit is CPU 0.

The default value of the mask is $f$, which means that an interrupt request can be handled on any processor in the system. Setting this value to 1 means that only processor 0 can handle the interrupt.

**Procedure**

1. In binary, use the value 1 for CPUs that handle the interrupts. For example, to set CPU 0 and CPU 7 to handle interrupts, use 0000000010000001 as the binary code:

   **Table 7.1. Binary Bits for CPUs**
2. Convert the binary code to hexadecimal:
   For example, to convert the binary code using Python:

   ```python
   >>> hex(int('0000000010000001', 2))
   '0x81'
   ```

   On systems with more than 32 processors, you must delimit the `smp_affinity` values for discrete 32 bit groups. For example, if you want only the first 32 processors of a 64 processor system to service an interrupt request, use `0xffffffff,00000000`.

3. The interrupt affinity value for a particular interrupt request is stored in the associated `/proc/irq/irq_number/smp_affinity` file. Set the `smp_affinity` mask in this file:

   ```bash
   # echo mask > /proc/irq/irq_number/smp_affinity
   ```

Additional resources

- `journalctl(1)`, `irqbalance(1)`, and `taskset(1)` man pages
CHAPTER 8. TUNING SCHEDULING POLICY

In Red Hat Enterprise Linux, the smallest unit of process execution is called a thread. The system scheduler determines which processor runs a thread, and for how long the thread runs. However, because the scheduler’s primary concern is to keep the system busy, it may not schedule threads optimally for application performance.

For example, say an application on a NUMA system is running on Node A when a processor on Node B becomes available. To keep the processor on Node B busy, the scheduler moves one of the application’s threads to Node B. However, the application thread still requires access to memory on Node A. But, this memory will take longer to access because the thread is now running on Node B and Node A memory is no longer local to the thread. Thus, it may take longer for the thread to finish running on Node B than it would have taken to wait for a processor on Node A to become available, and then to execute the thread on the original node with local memory access.

8.1. CATEGORIES OF SCHEDULING POLICIES

Performance sensitive applications often benefit from the designer or administrator determining where threads are run. The Linux scheduler implements a number of scheduling policies which determine where and for how long a thread runs.

The following are the two major categories of scheduling policies:

Normal policies
Normal threads are used for tasks of normal priority.

Realtime policies
Realtime policies are used for time-sensitive tasks that must complete without interruptions. Realtime threads are not subject to time slicing. This means the thread runs until they block, exit, voluntarily yield, or are preempted by a higher priority thread. The lowest priority realtime thread is scheduled before any thread with a normal policy. For more information, see Static priority scheduling with SCHED_FIFO and Round robin priority scheduling with SCHED_RR.

Additional resources

- sched(7), sched_setaffinity(2), sched_getaffinity(2), sched_setscheduler(2), and sched_getscheduler(2) man pages

8.2. STATIC PRIORITY SCHEDULING WITH SCHED_FIFO

The SCHED_FIFO, also called static priority scheduling, is a realtime policy that defines a fixed priority for each thread. This policy allows administrators to improve event response time and reduce latency. It is recommended to not execute this policy for an extended period of time for time sensitive tasks.

When SCHED_FIFO is in use, the scheduler scans the list of all the SCHED_FIFO threads in order of priority and schedules the highest priority thread that is ready to run. The priority level of a SCHED_FIFO thread can be any integer from 1 to 99, where 99 is treated as the highest priority. Red Hat recommends starting with a lower number and increasing priority only when you identify latency issues.
Because realtime threads are not subject to time slicing, Red Hat does not recommend setting a priority as 99. This keeps your process at the same priority level as migration and watchdog threads; if your thread goes into a computational loop and these threads are blocked, they will not be able to run. Systems with a single processor will eventually hang in this situation.

Administrators can limit SCHED_FIFO bandwidth to prevent realtime application programmers from initiating realtime tasks that monopolize the processor.

The following are some of the parameters used in this policy:

```
/proc/sys/kernel/sched_rt_period_us
This parameter defines the time period, in microseconds, that is considered to be one hundred percent of the processor bandwidth. The default value is 1000000 µs, or 1 second.
```

```
/proc/sys/kernel/sched_rt_runtime_us
This parameter defines the time period, in microseconds, that is devoted to running real-time threads. The default value is 950000 µs, or 0.95 seconds.
```

8.3. ROUND ROBIN PRIORITY SCHEDULING WITH SCHED_RR

The SCHED_RR is a round-robin variant of the SCHED_FIFO. This policy is useful when multiple threads need to run at the same priority level.

Like SCHED_FIFO, SCHED_RR is a realtime policy that defines a fixed priority for each thread. The scheduler scans the list of all SCHED_RR threads in order of priority and schedules the highest priority thread that is ready to run. However, unlike SCHED_FIFO, threads that have the same priority are scheduled in a round-robin style within a certain time slice.

You can set the value of this time slice in milliseconds with the sched_rr_timeslice_ms kernel parameter in the /proc/sys/kernel/sched_rr_timeslice_ms file. The lowest value is 1 millisecond.

8.4. NORMAL SCHEDULING WITH SCHED_OTHER

The SCHED_OTHER is the default scheduling policy in Red Hat Enterprise Linux 9. This policy uses the Completely Fair Scheduler (CFS) to allow fair processor access to all threads scheduled with this policy. This policy is most useful when there are a large number of threads or when data throughput is a priority, as it allows more efficient scheduling of threads over time.

When this policy is in use, the scheduler creates a dynamic priority list based partly on the niceness value of each process thread. Administrators can change the niceness value of a process, but cannot change the scheduler’s dynamic priority list directly.

8.5. SETTING SCHEDULER POLICIES

Check and adjust scheduler policies and priorities by using the chrt command line tool. It can start new processes with the desired properties, or change the properties of a running process. It can also be used for setting the policy at runtime.
Procedure

1. View the process ID (PID) of the active processes:

   ```
   # ps
   ```

   Use the `--pid` or `-p` option with the `ps` command to view the details of the particular PID.

2. Check the scheduling policy, PID, and priority of a particular process:

   ```
   # chrt -p 468
   pid 468's current scheduling policy: SCHED_FIFO
   pid 468's current scheduling priority: 85
   
   # chrt -p 476
   pid 476's current scheduling policy: SCHED_OTHER
   pid 476's current scheduling priority: 0
   ```

   Here, 468 and 476 are PID of a process.

3. Set the scheduling policy of a process:

   a. For example, to set the process with PID 1000 to `SCHED_FIFO`, with a priority of 50:
      
      ```
      # chrt -f -p 50 1000
      ```

   b. For example, to set the process with PID 1000 to `SCHED_OTHER`, with a priority of 0:
      
      ```
      # chrt -o -p 0 1000
      ```

   c. For example, to set the process with PID 1000 to `SCHED_RR`, with a priority of 10:
      
      ```
      # chrt -r -p 10 1000
      ```

   d. To start a new application with a particular policy and priority, specify the name of the application:
      
      ```
      # chrt -f 36 /bin/my-app
      ```

Additional resources

- `chrt(1)` man page
- Policy Options for the chrt command
- Changing the priority of services during the boot process

8.6. POLICY OPTIONS FOR THE CHRT COMMAND

Using the `chrt` command, you can view and set the scheduling policy of a process.

The following table describes the appropriate policy options, which can be used to set the scheduling policy of a process.
Table 8.1. Policy Options for the chrt Command

<table>
<thead>
<tr>
<th>Short option</th>
<th>Long option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-f</td>
<td>--fifo</td>
<td>Set schedule to SCHED_FIFO</td>
</tr>
<tr>
<td>-o</td>
<td>--other</td>
<td>Set schedule to SCHED_OTHER</td>
</tr>
<tr>
<td>-r</td>
<td>--rr</td>
<td>Set schedule to SCHED_RR</td>
</tr>
</tbody>
</table>

8.7. CHANGING THE PRIORITY OF SERVICES DURING THE BOOT PROCESS

Using the systemd service, it is possible to set up real-time priorities for services launched during the boot process. The unit configuration directives are used to change the priority of a service during the boot process.

The boot process priority change is done by using the following directives in the service section:

**CPUSchedulingPolicy=**

Sets the CPU scheduling policy for executed processes. It is used to set other, fifo, and rr policies.

**CPUSchedulingPriority=**

Sets the CPU scheduling priority for executed processes. The available priority range depends on the selected CPU scheduling policy. For real-time scheduling policies, an integer between 1 (lowest priority) and 99 (highest priority) can be used.

The following procedure describes how to change the priority of a service, during the boot process, using the mcelog service.

Prerequisites

1. Install the tuned package:

   ```
   # {PackageManagerCommand} install tuned
   ```

2. Enable and start the tuned service:

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

Procedure

1. View the scheduling priorities of running threads:

   ```
   # tuna --show_threads
thread  ctxt_switches pid SCHED_rrtpr affivity voluntary nonvoluntary cmd
   1 OTHER 0 0xff 3181 292 systemd
   2 OTHER 0 0xff 254 0 kthread
   3 OTHER 0 0xff 2 0 rcu_gp
   4 OTHER 0 0xff 2 0 rcu_par_gp
   ```
2. Create a supplementary `mcelog` service configuration directory file and insert the policy name and priority in this file:

```bash
# cat <<-EOF > /etc/systemd/system/mcelog.service.d/priority.conf
>
[SERVICE]
CPUSchedulingPolicy=_fifo_
CPUSchedulingPriority=_20_
EOF
```

3. Reload the `systemd` scripts configuration:

```bash
# systemctl daemon-reload
```

4. Restart the `mcelog` service:

```bash
# systemctl restart mcelog
```

**Verification steps**

- Display the `mcelog` priority set by `systemd` issue:

```bash
# tuna -t mcelog -P
thread   ctxt_switches
  pid SCHED_ rtpri affinity voluntary nonvoluntary    cmd
826     FIFO    20 0,1,2,3        13            0          mcelog
```

**Additional resources**

- `systemd(1)` and `tuna(8)` man pages
- **Description of the priority range**

### 8.8. PRIORITY MAP

Priorities are defined in groups, with some groups dedicated to certain kernel functions. For real-time scheduling policies, an integer between 1 (lowest priority) and 99 (highest priority) can be used.

The following table describes the priority range, which can be used while setting the scheduling policy of a process.

**Table 8.2. Description of the priority range**
### Priority Threads Description

<table>
<thead>
<tr>
<th>Priority</th>
<th>Threads</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Low priority kernel threads</td>
<td>This priority is usually reserved for the tasks that need to be just above <code>SCHED_OTHER</code>.</td>
</tr>
<tr>
<td>2 - 49</td>
<td>Available for use</td>
<td>The range used for typical application priorities.</td>
</tr>
<tr>
<td>50</td>
<td>Default hard-IRQ value</td>
<td></td>
</tr>
<tr>
<td>51 - 98</td>
<td>High priority threads</td>
<td>Use this range for threads that execute periodically and must have quick response times. Do not use this range for CPU-bound threads as you will starve interrupts.</td>
</tr>
<tr>
<td>99</td>
<td>Watchdogs and migration</td>
<td>System threads that must run at the highest priority.</td>
</tr>
</tbody>
</table>

#### 8.9. TUNED CPU-PARTITIONING PROFILE

For tuning Red Hat Enterprise Linux 8 for latency-sensitive workloads, Red Hat recommends to use the `cpu-partitioning` TuneD profile.

Prior to Red Hat Enterprise Linux 8, the low-latency Red Hat documentation described the numerous low-level steps needed to achieve low-latency tuning. In Red Hat Enterprise Linux 8, you can perform low-latency tuning more efficiently by using the `cpu-partitioning` TuneD profile. This profile is easily customizable according to the requirements for individual low-latency applications.

The following figure is an example to demonstrate how to use the `cpu-partitioning` profile. This example uses the cpu and node layout.

**Figure 8.1. Figure cpu-partitioning**

You can configure the `cpu-partitioning` profile in the `/etc/tuned/cpu-partitioning-variables.conf` file using the following configuration options:

- **Isolated cpus with load balancing**
In the cpu-partitioning figure, the blocks numbered from 4 to 23, are the default isolated cpus. The kernel scheduler’s process load balancing is enabled on these cpus. It is designed for low-latency processes with multiple threads that need the kernel scheduler load balancing.

You can configure the cpu-partitioning profile in the `/etc/tuned/cpu-partitioning-variables.conf` file using the `isolated_cores=cpu-list` option, which lists CPUs to isolate that will use the kernel scheduler load balancing.

The list of isolated CPUs is comma-separated or you can specify a range using a dash, such as `3-5`. This option is mandatory. Any CPU missing from this list is automatically considered a housekeeping CPU.

**Isolated cpus without load balancing**

In the cpu-partitioning figure, the blocks numbered 2 and 3, are the isolated cpus that do not provide any additional kernel scheduler process load balancing.

You can configure the cpu-partitioning profile in the `/etc/tuned/cpu-partitioning-variables.conf` file using the `no_balance_cores=cpu-list` option, which lists CPUs to isolate that will not use the kernel scheduler load balancing.

Specifying the `no_balance_cores` option is optional, however any cpus in this list must be a subset of the cpus listed in the `isolated_cores` list.

Application threads using these cpus need to be pinned individually to each cpu.

**Housekeeping cpus**

Any cpu not isolated in the `cpu-partitioning-variables.conf` file is automatically considered a housekeeping cpu. On the housekeeping cpus, all services, daemons, user processes, movable kernel threads, interrupt handlers, and kernel timers are permitted to execute.

**Additional resources**

- `tuned-profiles-cpu-partitioning(7)` man page

**8.10. USING THE TUNED CPU-PARTITIONING PROFILE FOR LOW-LATENCY TUNING**

This procedure describes how to tune a system for low-latency using the TuneD’s `cpu-partitioning` profile. It uses the example of a low-latency application that can use `cpu-partitioning` and the CPU layout as mentioned in the `cpu-partitioning` figure.

The application in this case uses:

- One dedicated reader thread that reads data from the network will be pinned to CPU 2.

- A large number of threads that process this network data will be pinned to CPUs 4-23.

- A dedicated writer thread that writes the processed data to the network will be pinned to CPU 3.

**Prerequisites**

- You have installed the `cpu-partitioning` TuneD profile by using the `dnf install tuned-profiles-cpu-partitioning` command as root.
Procedure

1. Edit `/etc/tuned/cpu-partitioning-variables.conf` file and add the following information:

```bash
# Isolated CPUs with the kernel’s scheduler load balancing:
isolated_cores=2-23
# Isolated CPUs without the kernel’s scheduler load balancing:
no_balance_cores=2,3
```

2. Set the `cpu-partitioning` TuneD profile:

```bash
# tuned-adm profile cpu-partitioning
```

3. Reboot

After rebooting, the system is tuned for low-latency, according to the isolation in the `cpu-partitioning` figure. The application can use `taskset` to pin the reader and writer threads to cpus 2 and 3, and the remaining application threads on cpus 4-23.

Additional resources

- `tuned-profiles-cpu-partitioning(7)` man page

8.11. CUSTOMIZING THE CPU-PARTITIONING TUNED PROFILE

You can extend the TuneD profile to make additional tuning changes.

For example, the `cpu-partitioning` profile sets the cpus to use `cstate=1`. In order to use the `cpu-partitioning` profile but to additionally change the CPU cstate from cstate1 to cstate0, the following procedure describes a new TuneD profile named `my_profile`, which inherits the `cpu-partitioning` profile and then sets C state-0.

Procedure

1. Create the `/etc/tuned/my_profile` directory:

```bash
# mkdir /etc/tuned/my_profile
```

2. Create a `tuned.conf` file in this directory, and add the following content:

```bash
# vi /etc/tuned/my_profile/tuned.conf
[main]
summary=Customized tuning on top of cpu-partitioning
include=cpu-partitioning
[cpu]
force_latency=cstate.id:0|1
```

3. Use the new profile:

```bash
# tuned-adm profile my_profile
```
NOTE

In the shared example, a reboot is not required. However, if the changes in the `my_profile` profile require a reboot to take effect, then reboot your machine.

Additional resources

- `tuned-profiles-cpu-partitioning(7)` man page
CHAPTER 9. CONFIGURING RESOURCE MANAGEMENT USING CGROUPS VERSION 2 WITH SYSTEMD

The core of systemd is service management and supervision. systemd ensures that the right services start at the right time and in the correct order during the boot process. When the services are running, they have to run smoothly to use the underlying hardware platform optimally. Therefore, systemd also provides capabilities to define resource management policies and to tune various options, which can improve the performance of the service.

9.1. PREREQUISITES

- Basic knowledge of the Linux cgroup subsystem.

9.2. INTRODUCTION TO RESOURCE DISTRIBUTION MODELS

For resource management, systemd uses the cgroups v2 interface.

Note that RHEL 9 uses cgroups v1 by default. Therefore, you must enable cgroups v2 so that systemd can use the cgroups v2 interface for resource management. For more information on how to enable cgroups v2, see Setting CPU limits to applications using cgroups-v2.

To modify the distribution of system resources, you can apply one or more of the following resource distribution models:

Weights

The resource is distributed by adding up the weights of all sub-groups and giving each sub-group the fraction matching its ratio against the sum.

For example, if you have 10 cgroups, each with Weight of value 100, the sum is 1000 and each cgroup receives one tenth of the resource.

Weight is usually used to distribute stateless resources. The CPUWeight= option is an implementation of this resource distribution model.

Limits

A cgroup can consume up to the configured amount of the resource, but you can also overcommit resources. Therefore, the sum of sub-group limits can exceed the limit of the parent cgroup. The MemoryMax= option is an implementation of this resource distribution model.

Protections

A protected amount of a resource can be set up for a cgroup. If the resource usage is below the protection boundary, the kernel will try not to penalize this cgroup in favor of other cgroups that compete for the same resource. An overcommit is also allowed.

The MemoryLow= option is an implementation of this resource distribution model.

Allocations

Exclusive allocations of an absolute amount of a finite resource. An overcommit is not allowed. An example of this resource type in Linux is the real-time budget.

Additional resources

- Managing CPU with systemd
9.3. ALLOCATING CPU RESOURCES USING SYSTEMD

On a system managed by systemd, each system service is started in its cgroup. By enabling the support for the CPU cgroup controller, the system uses the service-aware distribution of CPU resources instead of the per-process distribution. In the service-aware distribution, each service receives approximately the same amount of CPU time relative to all other services running on the system, regardless of the number of processes that comprise the service.

If a specific service requires more CPU resources, you can grant them by changing the CPU time allocation policy for the service.

Procedure

To set a CPU time allocation policy option when using systemd:

1. Check the assigned values of the CPU time allocation policy option in the service of your choice:
   
   $ systemctl show --property <CPU time allocation policy option> <service name>

2. Set the required value of the CPU time allocation policy option as a root:
   
   # systemctl set-property <service name> <CPU time allocation policy option>=<value>

   **NOTE**

   The cgroup properties are applied immediately after they are set. Therefore, the service does not need to be restarted.

   The cgroup properties are applied immediately after they are set. Therefore, the service does not need to be restarted.

Verification steps

- To verify whether you successfully changed the required value of the CPU time allocation policy option for your service, run the following command:

   $ systemctl show --property <CPU time allocation policy option> <service name>

Additional resources

- CPU time allocation policy options for systemd
- Introduction to resource distribution models

9.4. CPU TIME ALLOCATION POLICY OPTIONS FOR SYSTEMD

The most frequently used CPU time allocation policy options include:

CPUWeight=
Assigns higher priority to a particular service over all other services. You can select a value from the interval 1 – 10,000. The default value is 100. For example, to give `httpd.service` twice as much CPU as to all other services, set the value to `CPUWeight=200`.

Note that `CPUWeight=` is applied only in cases when the operating system is overloaded.

**CPUQuota=**

Assigns the absolute CPU time quota to a service. The value of this option specifies the maximum percentage of CPU time that a service will receive relative to the total CPU time available, for example `CPUQuota=30%`. Note that `CPUQuota=` represents the limit value for particular resource distribution models described in Introduction to resource distribution models.

For more information on `CPUQuota=`, see the `systemd.resource-control(5)` man page.

Additional resources

- Introduction to resource distribution models
- Allocating CPU resources using systemd

### 9.5. ALLOCATING MEMORY RESOURCES USING SYSTEMD

This section describes how to use any of the memory configuration options (`MemoryMin`, `MemoryLow`, `MemoryHigh`, `MemoryMax`, `MemorySwapMax`) to allocate memory resources using systemd.

**Procedure**

To set a memory allocation configuration option when using systemd:

1. Check the assigned values of the memory allocation configuration option in the service of your choice:

   `$ systemctl show --property <memory allocation configuration option> <service name>`

2. Set the required value of the memory allocation configuration option as a root:

   `# systemctl set-property <service name> <memory allocation configuration option>=<value>`

   **NOTE**

   The cgroup properties are applied immediately after they are set. Therefore, the service does not need to be restarted.

**Verification steps**

- To verify whether you successfully changed the required value of the memory allocation configuration option for your service, run the following command:

  `$ systemctl show --property <memory allocation configuration option> <service name>`
9.6. MEMORY ALLOCATION CONFIGURATION OPTIONS FOR SYSTEMD

You can use the following options when using systemd to configure system memory allocation:

**MemoryMin**
- Hard memory protection. If the memory usage is below the limit, the cgroup memory will not be reclaimed.

**MemoryLow**
- Soft memory protection. If the memory usage is below the limit, the cgroup memory can be reclaimed only if no memory is reclaimed from unprotected cgroups.

**MemoryHigh**
- Memory throttle limit. If the memory usage goes above the limit, the processes in the cgroup are throttled and put under a heavy reclaim pressure.

**MemoryMax**
- Absolute limit for the memory usage. You can use the kilo (K), mega (M), giga (G), tera (T) suffixes, for example `MemoryMax=1G`.

**MemorySwapMax**
- Hard limit on the swap usage.

**NOTE**
When you exhaust your memory limit, the Out-of-memory (OOM) killer will stop the running service. To prevent this, lower the `OOMScoreAdjust` value to increase the memory tolerance.

Additional resources

- Allocating memory resources using systemd
- Introduction to resource distribution models

9.7. CONFIGURING I/O BANDWIDTH USING SYSTEMD

To improve the performance of a specific service in RHEL 9, you can allocate I/O bandwidth resources to that service using systemd.

To do so, you can use the following I/O configuration options:

- IOWeight
- IODeviceWeight
- IORedBandwidthMax
- IOWriteBandwidthMax
- IOReadIOPSMax
- IOWriteIOPSMax

Procedure

To set a I/O bandwidth configuration option using systemd:

1. Check the assigned values of the I/O bandwidth configuration option in the service of your choice:

   ```
   $ systemctl show --property <I/O bandwidth configuration option> <service name>
   ```

2. Set the required value of the I/O bandwidth configuration option as a root:

   ```
   # systemctl set-property <service name> <I/O bandwidth configuration option>=<value>
   ```

The cgroup properties are applied immediately after they are set. Therefore, the service does not need to be restarted.

Verification steps

- To verify whether you successfully changed the required value of the I/O bandwidth configuration option for your service, run the following command:

  ```
  $ systemctl show --property <I/O bandwidth configuration option> <service name>
  ```

Additional resources

- I/O bandwidth configuration options for systemd
- Introduction to resource distribution models

9.8. I/O BANDWIDTH CONFIGURATION OPTIONS FOR SYSTEMD

To manage the block layer I/O policies with systemd, the following configuration options are available:

**IOWeight**

Sets the default I/O weight. The weight value is used as a basis for the calculation of how much of the real I/O bandwidth the service receives in relation to the other services.

**IODeviceWeight**

Sets the I/O weight for a specific block device.

For example, `IODeviceWeight=/dev/disk/by-id/dm-name-rhel-root 200`.

**IOReadBandwidthMax, IOWriteBandwidthMax**

Sets the absolute bandwidth per device or a mount point.

For example, `IOWriteBandwidth=/var/log 5M`.

**NOTE**

Systemd handles the file-system-to-device translation automatically.
**IOReadIOPSMax, IOWriteIOPSMax**

A similar option to the previous one: sets the absolute bandwidth in Input/Output Operations Per Second (IOPS).

**NOTE**

Weight-based options are supported only if the block device is using the CFQ I/O scheduler. No option is supported if the device uses the Multi-Queue Block I/O queuing mechanism.

**Additional resources**

- Configuring I/O bandwidth using systemd
- Introduction to resource distribution models