readme.md for @push.rocks/smartmetrics

Powerful system metrics collection for Node.js applications with Prometheus integration □□

Issue Reporting and Security

For reporting bugs, issues, or security vulnerabilities, please visit community.foss.global/. This is the central community hub for all issue reporting. Developers who sign and comply with our contribution agreement and go through identification can also get a code.foss.global/ account to submit Pull Requests directly.

What is SmartMetrics?

SmartMetrics is a comprehensive metrics collection library that monitors your Node.js application's resource usage in real-time. It tracks CPU usage, memory consumption, and system metrics across your main process and all child processes, providing insights through both JSON and Prometheus formats.

Key Features

- □□ **Real-time Metrics Collection** – Monitor CPU and memory usage across all processes
- □□ **Automatic Child Process Tracking** – Aggregates metrics from main and child processes via `pidtree` + `pidusage`
- □□ **Docker-Aware** – Detects container memory limits from cgroup automatically
- □□ **Prometheus Integration** – Built-in HTTP endpoint for Prometheus scraping with `prom-client`
- □□ **Flexible Output Formats** – Get metrics as JSON objects or Prometheus text exposition format
- □□ **Automatic Heartbeat Logging** – Optional periodic metrics logging via `@push.rocks/smartlog`

- **Zero Configuration** – Works out of the box with sensible defaults

Installation

```
pnpm install @push.rocks/smartmetrics
# or
npm install @push.rocks/smartmetrics
```

Quick Start

```
import { SmartMetrics } from '@push.rocks/smartmetrics';
import { Smartlog } from '@push.rocks/smartlog';

// Create a logger instance
const logger = new Smartlog({
  logContext: null,
  minimumLogLevel: 'info',
});
logger.enableConsole();

// Initialize SmartMetrics
const metrics = new SmartMetrics(logger, 'my-service');

// Get metrics on-demand
const currentMetrics = await metrics.getMetrics();
console.log(`CPU Usage: ${currentMetrics.cpuUsageText}`);
console.log(`Memory: ${currentMetrics.memoryUsageText}`);

// Enable automatic heartbeat logging (every 20 seconds)
metrics.start();

// Enable Prometheus endpoint
metrics.enablePrometheusEndpoint(9090);
// Metrics now available at http://localhost:9090/metrics
```

```
// Clean shutdown
metrics.stop();
```

Core Concepts

Process Aggregation

SmartMetrics doesn't just monitor your main process – it automatically discovers and aggregates metrics from all child processes spawned by your application using `pidtree`. This gives you a complete picture of your application's resource footprint, not just the parent process.

Memory Limit Detection

The library automatically detects available memory whether running on bare metal, in Docker containers, or with Node.js heap restrictions. It picks the most restrictive of:

1. **System total memory** (`os.totalmem()`)
2. **Docker cgroup limit** – supports both cgroup v2 (`/sys/fs/cgroup/memory.max`) and cgroup v1 (`/sys/fs/cgroup/memory/memory.limit_in_bytes`)
3. **V8 heap size limit** (`v8.getHeapStatistics().heap_size_limit`)

This ensures accurate percentage calculations regardless of environment.

Dual Output Formats

- **JSON Format** (`getMetrics()`) – Ideal for application monitoring, custom dashboards, and programmatic access
- **Prometheus Format** (`getPrometheusFormattedMetrics()`) – Perfect for integration with Prometheus/Grafana monitoring stacks

API Reference

`new SmartMetrics(logger, sourceName)`

Creates a new SmartMetrics instance.

Parameter	Type	Description
`logger`	`Smartlog`	A `@push.rocks/smartlog` logger instance
`sourceName`	`string`	Identifier for your service/application

`async getMetrics(): Promise<IMetricsSnapshot>`

Retrieves current system metrics as a structured object.

Returns `IMetricsSnapshot`:

```
{
  process_cpu_seconds_total: number;    // Total CPU time in seconds
  nodejs_active_handles_total: number;   // Always 0 (deprecated Node.js API; real values tracked by Prometheus default collectors)
  nodejs_active_requests_total: number;  // Always 0 (deprecated Node.js API; real values tracked by Prometheus default collectors)
  nodejs_heap_size_total_bytes: number;  // V8 heap size in bytes
  cpuPercentage: number;                 // Aggregated CPU usage across all child processes
  cpuUsageText: string;                  // Human-readable CPU usage (e.g. "12.5 %")
  memoryPercentage: number;              // Memory usage as percentage of detected limit
  memoryUsageBytes: number;              // Total memory in bytes across all child processes
  memoryUsageText: string;               // Human-readable (e.g. "45% | 920 MB / 2 GB")
}
```

Example:

```
const metrics = await smartMetrics.getMetrics();
if (metrics.cpuPercentage > 80) {
  console.warn('High CPU usage detected!');
}
```

`start(): void`

Starts automatic metrics collection and heartbeat logging every 20 seconds via the provided `Smartlog` instance.

```
smartMetrics.start();
// Logs: "sending heartbeat for my-service with metrics" every 20 seconds
```

`stop(): void`

Stops automatic metrics collection, closes heartbeat loop, and shuts down any open Prometheus endpoints.

`async getPrometheusFormattedMetrics(): Promise<string>`

Returns all metrics in Prometheus text exposition format, including default Node.js collectors and custom SmartMetrics gauges.

```
const promMetrics = await smartMetrics.getPrometheusFormattedMetrics();
// Returns:
// # HELP smartmetrics_cpu_percentage Current CPU usage percentage
// # TYPE smartmetrics_cpu_percentage gauge
// smartmetrics_cpu_percentage 15.2
// # HELP smartmetrics_memory_percentage Current memory usage percentage
// # TYPE smartmetrics_memory_percentage gauge
// smartmetrics_memory_percentage 45.3
// # HELP smartmetrics_memory_usage_bytes Current memory usage in bytes
// # TYPE smartmetrics_memory_usage_bytes gauge
// smartmetrics_memory_usage_bytes 965214208
// ... plus all default Node.js metrics from prom-client
```

`enablePrometheusEndpoint(port?: number): void`

Starts an HTTP server that exposes metrics at `/metrics` for Prometheus scraping.

Parameter	Type	Default	Description
`port`	`number`	`9090`	Port to listen on

```
smartMetrics.enablePrometheusEndpoint(3000);
// GET http://localhost:3000/metrics → Prometheus text format
```

```
// GET http://localhost:3000/anything-else → 404
```

`disablePrometheusEndpoint(): void`

Gracefully shuts down the Prometheus HTTP server.

`formatBytes(bytes, decimals?): string`

Utility method to convert byte values to human-readable strings.

```
smartMetrics.formatBytes(1073741824); // "1 GB"
smartMetrics.formatBytes(1536, 1);    // "1.5 KB"
```

Use Cases

□□ Application Performance Monitoring

```
const metricsBefore = await smartMetrics.getMetrics();
await performHeavyOperation();
const metricsAfter = await smartMetrics.getMetrics();

console.log(`Operation consumed ${
  metricsAfter.process_cpu_seconds_total - metricsBefore.process_cpu_seconds_total
} CPU seconds`);
```

□□ Resource Limit Enforcement

```
async function checkResources() {
  const metrics = await smartMetrics.getMetrics();

  if (metrics.memoryPercentage > 90) {
    throw new Error('Memory usage too high, refusing new operations');
  }

  if (metrics.cpuPercentage > 95) {
```

```
    await delay(1000); // Back off when CPU is stressed
  }
}
```

□□ Prometheus + Grafana Stack

```
smartMetrics.enablePrometheusEndpoint(9090);

// prometheus.yml:
// scrape_configs:
//   - job_name: 'my-app'
//     scrape_interval: 15s
//     static_configs:
//       - targets: ['localhost:9090']
```

□□ Container Resource Monitoring

```
// Automatically detects Docker/cgroup memory limits
const metrics = await smartMetrics.getMetrics();
console.log(metrics.memoryUsageText);
// Output: "45% | 920 MB / 2 GB" (container limit detected)
```

□□ Health Check Endpoint

```
import express from 'express';

const app = express();

app.get('/health', async (req, res) => {
  const metrics = await smartMetrics.getMetrics();

  res.json({
    status: metrics.memoryPercentage < 90 ? 'healthy' : 'degraded',
    cpu: metrics.cpuUsageText,
    memory: metrics.memoryUsageText,
  });
```

```
});
```

Graceful Restart on High Memory (PM2)

```
setInterval(async () => {
  const metrics = await smartMetrics.getMetrics();

  if (metrics.memoryPercentage > 95) {
    console.error('Memory limit reached, requesting restart');
    process.exit(0); // PM2 will restart the process
  }
}, 10000);
```

License and Legal Information

This repository contains open-source code licensed under the MIT License. A copy of the license can be found in the LICENSE file.

Please note: The MIT License does not grant permission to use the trade names, trademarks, service marks, or product names of the project, except as required for reasonable and customary use in describing the origin of the work and reproducing the content of the NOTICE file.

Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH or third parties, and are not included within the scope of the MIT license granted herein.

Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines or the guidelines of the respective third-party owners, and any usage must be approved in writing. Third-party trademarks used herein are the property of their respective owners and used only in a descriptive manner, e.g. for an implementation of an API or similar.

Company Information

Task Venture Capital GmbH Registered at District Court Bremen HRB 35230 HB, Germany

For any legal inquiries or further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.

Revision #3
Created 2026-03-28 11:11:31 UTC by foss.global Team
Updated 2026-03-28 12:18:21 UTC by foss.global Team