

**Politechnika Wrocławskas  
Wydział Informatyki i Telekomunikacji  
Technologie Sieciowe Laboratoria**

**Raport z zadania nr.2  
Monitorowanie wydajności  
systemu - Raspberry Pi  
28.10.2025**

Autorzy:  
Daniel Gościński 280878  
Łukasz Duda 280916

Prowadzący:  
mgr. inż. Karol Puchała

# Spis treści

1.	Wstęp .....	3
2.	Cel ćwiczeń .....	3
3.	Przebieg wykonanych zadań .....	3
3.1.	Nawiązanie połączenia przez SSH .....	3
3.2.	Usługa NetData .....	4
3.2.1.	Instalacja usługi .....	4
3.2.2.	Drzewo zależności .....	5
3.2.3.	Konfiguracja .....	6
3.3.	Monitorowanie zasobów .....	7
3.3.1.	Połączenie z komputera stacjonarnego .....	7
3.3.2.	Program stress .....	8
3.3.3.	Stress test pamięci .....	8
3.3.4.	Stress test procesora .....	9
3.3.5.	Stress test dysku .....	10
4.	Wnioski .....	11
5.	Odpowiedzi na dodatkowe pytania .....	12
5.1.	Pytanie 1.....	12
5.2.	Pytanie 2.....	12
5.3.	Pytanie 3.....	12

## **1. Wstęp**

Na zajęciach dany był dostęp do dwóch stacji komputerowych oraz urządzenia Raspberry Pi 4, wszystkie podłączone do tej samej sieci. Raspberry Pi, na którym skupiają się laboratoria wykorzystuje system *Linux* wraz ze środowiskiem graficznym *GNOME* opierającym się o prostotę i łatwość użytkowania. Dodatkowo ta dystrybucja używa systemu inicjalizacji *systemd* zajmującym się zarządzaniem zasobami i usługami systemowymi.

## **2. Cel ćwiczenia**

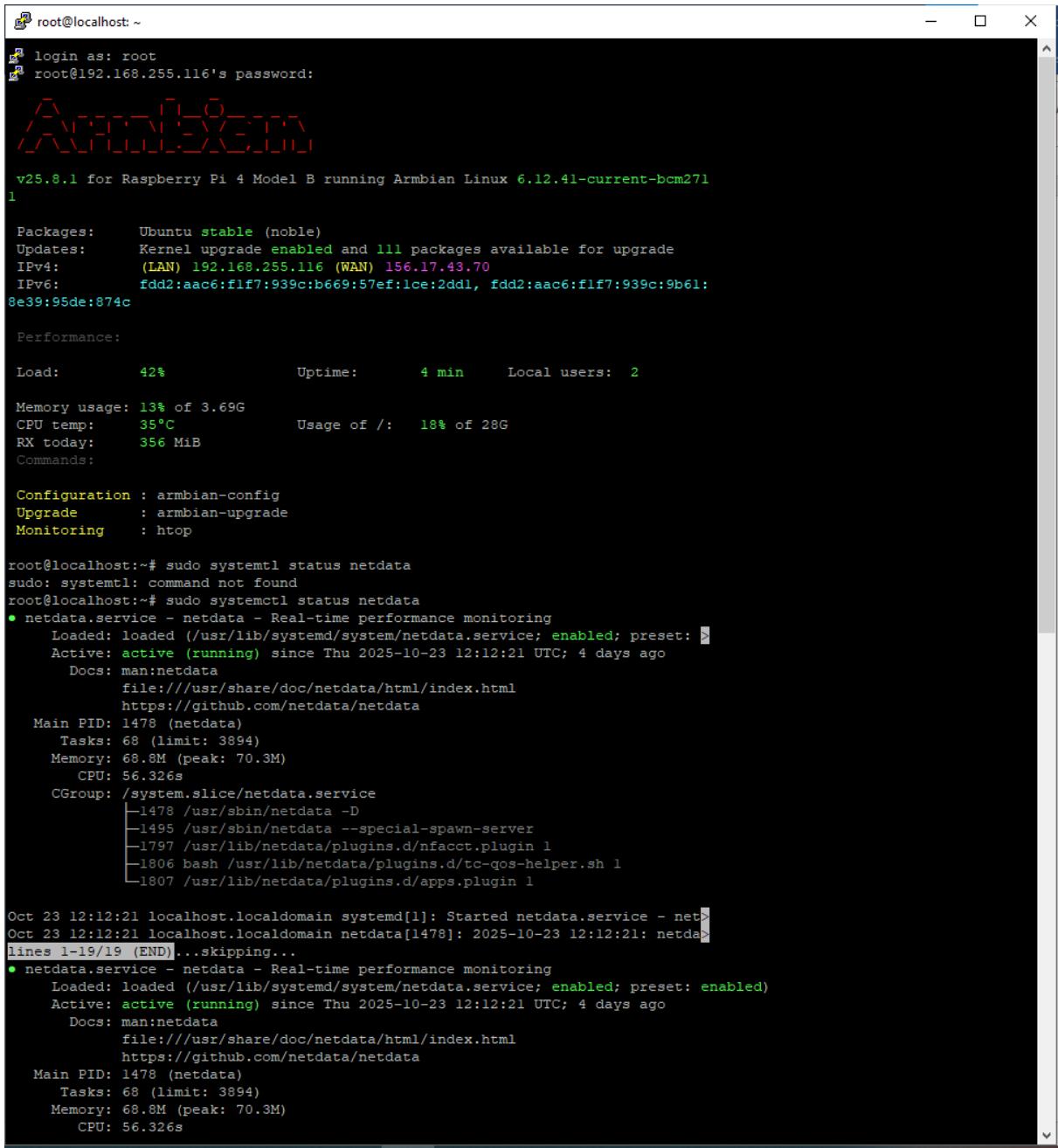
Celem ćwiczenie jest wykorzystanie urządzenia Raspberry Pi w celu monitorowania ruchu internetowego jak i również zużycia zasobów systemu takich jak pamięć lub procesor. Aby tego dokonać wykorzystane zostanie narzędzie NetData, zezwalające na obserwacje zużycia oraz analizę poprzez szczegółowe wykresy

## **3. Przebieg wykonanych zadań**

### **3.1. Nawiązanie połączenia przez SSH**

Na początek podłączono urządzenie Raspberry do zasilania oraz peryferii takich jak monitor, klawiatura oraz mysz. To pozwoliło na dostęp do interfejsu graficznego urządzenia. Stamtąd włączony został terminal i polecenie **ip a** wypisujące wszystkie adresy ip. W tym wypadku po podłączeniu do sieci przypisany został adres 192.168.255.116.

Mając ten adres połączono się zdalnie z komputera stacjonarnego wykorzystując program PUTTY. Po pomyślnym połączeniu zalogowano się na konto root używając hasła podanego w treści zadania.



```
root@localhost:~  
login as: root  
root@192.168.255.116's password:  
  
v25.8.1 for Raspberry Pi 4 Model B running Armbian Linux 6.12.41-current-bcm271  
1  
  
Packages: Ubuntu stable (noble)  
Updates: Kernel upgrade enabled and 111 packages available for upgrade  
IPv4: (LAN) 192.168.255.116 (WAN) 156.17.43.70  
IPv6: fdd2:aac6:f1f7:939c:b669:57ef:1ce:2dd1, fdd2:aac6:f1f7:939c:9b61:  
8e39:95de:874c  
  
Performance:  
  
Load: 42% Uptime: 4 min Local users: 2  
  
Memory usage: 13% of 3.69G CPU temp: 35°C Usage of /: 18% of 28G  
RX today: 356 MiB Commands:  
  
Configuration : armbian-config  
Upgrade : armbian-upgrade  
Monitoring : htop  
  
root@localhost:~# sudo systemctl status netdata  
sudo: systemctl: command not found  
root@localhost:~# sudo systemctl status netdata  
● netdata.service - netdata - Real-time performance monitoring  
  Loaded: loaded (/usr/lib/systemd/system/netdata.service; enabled; preset: enabled)  
  Active: active (running) since Thu 2025-10-23 12:12:21 UTC; 4 days ago  
    Docs: man:netdata  
          file:///usr/share/doc/netdata/html/index.html  
          https://github.com/netdata/netdata  
  Main PID: 1478 (netdata)  
     Tasks: 68 (limit: 3894)  
   Memory: 68.8M (peak: 70.3M)  
     CPU: 56.326s  
  CGroup: /system.slice/netdata.service  
          ├─1478 /usr/sbin/netdata -D  
          ├─1495 /usr/sbin/netdata --special-spawn-server  
          ├─1797 /usr/lib/netdata/plugins.d/nfacct.plugin 1  
          ├─1806 bash /usr/lib/netdata/plugins.d/tc-qos-helper.sh 1  
          └─1807 /usr/lib/netdata/plugins.d/apps.plugin 1  
  
Oct 23 12:12:21 localhost.localdomain systemd[1]: Started netdata.service - netdata.  
Oct 23 12:12:21 localhost.localdomain netdata[1478]: 2025-10-23 12:12:21: netdata  
lines 1-19/19 (END)...skipping...  
● netdata.service - netdata - Real-time performance monitoring  
  Loaded: loaded (/usr/lib/systemd/system/netdata.service; enabled; preset: enabled)  
  Active: active (running) since Thu 2025-10-23 12:12:21 UTC; 4 days ago  
    Docs: man:netdata  
          file:///usr/share/doc/netdata/html/index.html  
          https://github.com/netdata/netdata  
  Main PID: 1478 (netdata)  
     Tasks: 68 (limit: 3894)  
   Memory: 68.8M (peak: 70.3M)  
     CPU: 56.326s
```

Rysunek 1: Pomyślne połączenie przez SSH

## 3.2. Usługa NetData

### 3.2.1. Instalacja usługi

Po zalogowaniu na konto root pierwszym krokiem było zainstalowanie usługi NetData. Wykorzystano do tego polecenie **sudo apt-get update && sudo apt-get install netdata**. Pierwsza część polecenia aktualizuje cały system, druga instaluje usługę.

```
root@localhost:~# sudo apt-get update && sudo apt-get install netdata
Hit:1 http://ports.ubuntu.com noble InRelease
Get:2 http://ports.ubuntu.com noble-security InRelease [126 kB]
Get:3 https://github.armbian.com/configng stable InRelease [5,467 B]
Get:4 https://packages.microsoft.com/repos/code stable InRelease [3,590 B]
Get:5 http://ports.ubuntu.com noble-updates InRelease [126 kB]
Get:7 http://ports.ubuntu.com noble-backports InRelease [126 kB]
Get:8 https://github.armbian.com/configng stable/main arm64 Packages [425 B]
Get:6 http://armbian.lv.auroradev.org/apt noble InRelease [61.9 kB]
Get:9 https://packages.microsoft.com/repos/code stable/main amd64 Packages [20.2 kB]
Get:10 https://packages.microsoft.com/repos/code stable/main armhf Packages [20.3 kB]
Get:11 https://packages.microsoft.com/repos/code stable/main arm64 Packages [20.3 kB]
Get:12 http://ports.ubuntu.com noble-security/main arm64 Packages [1,691 kB]
Get:13 http://ports.ubuntu.com noble-security arm64 Contents (deb) [146 MB]
Get:14 http://armbian.lv.auroradev.org/apt noble/main all Packages [22.7 kB]
Get:15 http://armbian.lv.auroradev.org/apt noble/main arm64 Packages [2,333 kB]
Get:16 http://armbian.lv.auroradev.org/apt noble/main all Contents (deb) [43.8 kB]
Get:17 http://armbian.lv.auroradev.org/apt noble/main arm64 Contents (deb) [38.0 MB]
Get:18 http://ports.ubuntu.com noble-security/restricted arm64 Packages [3,591 kB]
Get:19 http://ports.ubuntu.com noble-security/universe arm64 Packages [1,152 kB]
Get:20 http://ports.ubuntu.com noble-updates/main arm64 Packages [2,034 kB]
Get:21 http://ports.ubuntu.com noble-updates arm64 Contents (deb) [153 MB]
Get:22 http://armbian.lv.auroradev.org/apt noble/noble-utils arm64 Packages [366 kB]
Get:23 http://armbian.lv.auroradev.org/apt noble/noble-desktop arm64 Packages [243 kB]
Get:24 http://armbian.lv.auroradev.org/apt noble/noble-desktop all Packages [54.3 kB]
Get:25 http://ports.ubuntu.com noble-updates/restricted arm64 Packages [3,722 kB]
Get:26 http://ports.ubuntu.com noble-updates/universe arm64 Packages [1,862 kB]
Fetched 354 MB in 2min 9s (2,755 kB/s)
Reading package lists... Done
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
netdata is already the newest version (1.43.2-lbuild2).
0 upgraded, 0 newly installed, 0 to remove and 116 not upgraded.
root@localhost:~#
```

Rysunek 2: Instalacja usługi NetData

### 3.2.2. Drzewo zależności

Po instalacji sprawdzono drzewo zależności usługi używając komendy **systemctl list-dependencies netdata**. Wynika z niego, że usługa zależna jest od czterech elementów:

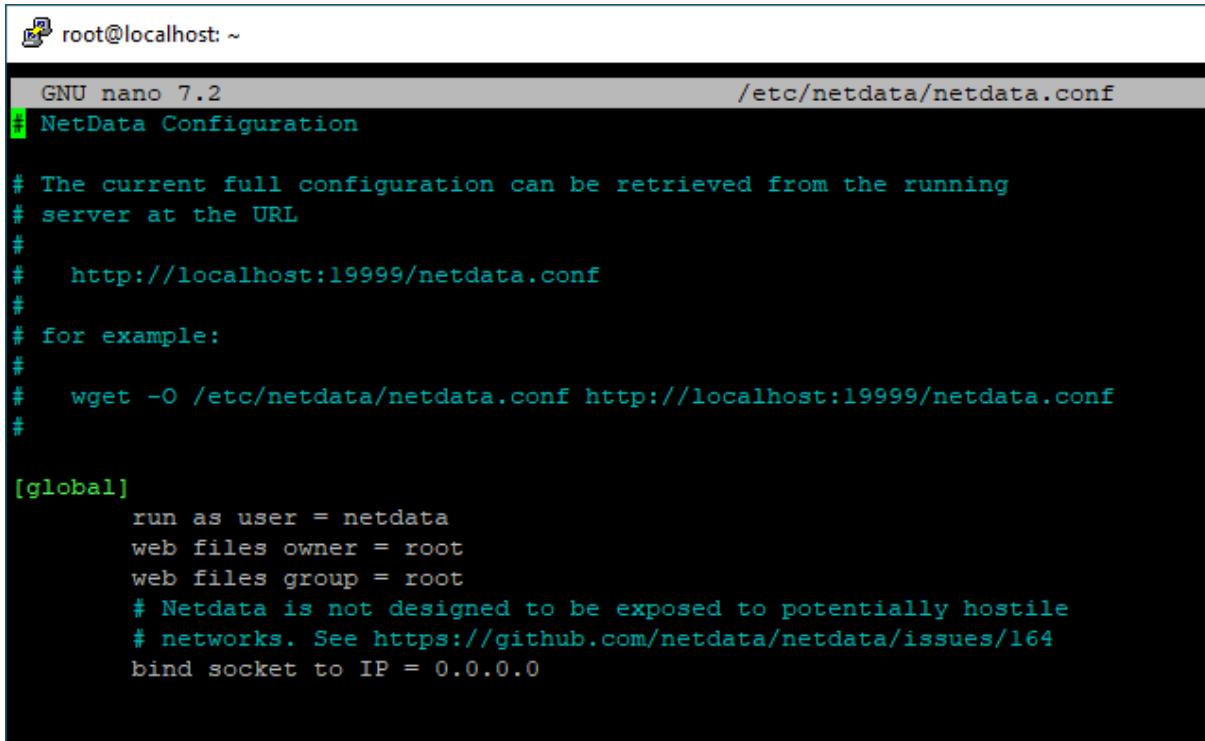
- .mount – punktu montowania plików
- system.slice – grupy jednostek zarządzania zasobami
- tmp.mount – systemu plików tmp
- sysinit.target – punktu synchronizacji

```
root@localhost:~# systemctl list-dependencies netdata
netdata.service
●
└─.mount
● └─system.slice
●   └─tmp.mount
●   └─sysinit.target
```

Rysunek 3: Lista zależności usługi

### 3.2.3. Konfiguracja

Domyślnie usługa NetData zezwala na połączenia wyłącznie z interfejsu *loopback*, czyli tylko lokalnie. Aby to zmienić edytowano plik konfiguracyjny usługi komendą **sudo nano /etc/netdata/netdata.conf** zmieniając parametr **bind socket** z 127.0.0.1 (loopback) na 0.0.0.0 co pozwoliło na połączenia zewnętrzne. Po zapisaniu pliku usługa została zrestartowana komendą **systemctl restart netdata**.

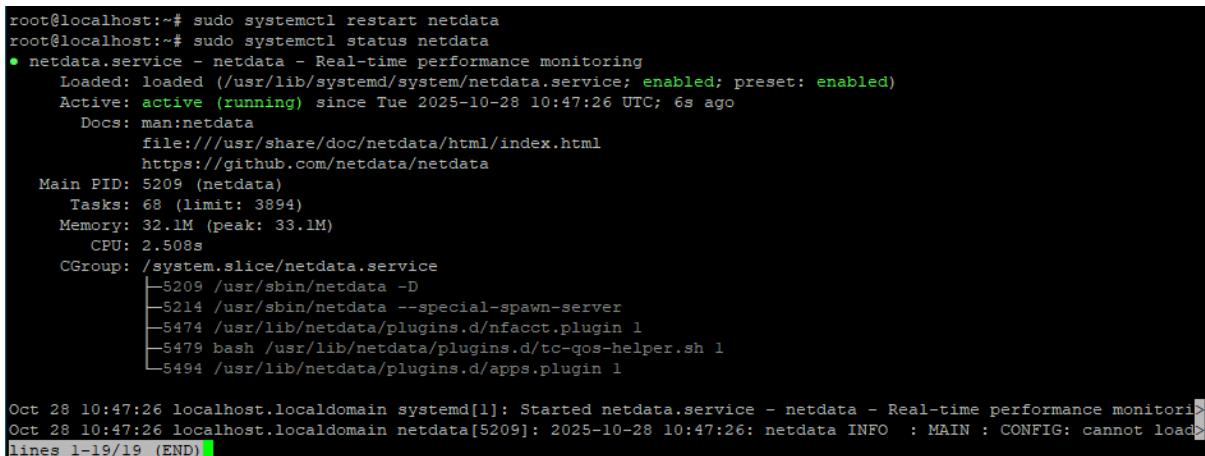


```
root@localhost: ~
GNU nano 7.2                               /etc/netdata/netdata.conf
# NetData Configuration

# The current full configuration can be retrieved from the running
# server at the URL
#
#     http://localhost:19999/netdata.conf
#
# for example:
#
#     wget -O /etc/netdata/netdata.conf http://localhost:19999/netdata.conf
#


[global]
    run as user = netdata
    web files owner = root
    web files group = root
    # Netdata is not designed to be exposed to potentially hostile
    # networks. See https://github.com/netdata/netdata/issues/164
    bind socket to IP = 0.0.0.0
```

Rysunek 4: Plik konfiguracyjny usługi ze zmienionym parametrem



```
root@localhost:~# sudo systemctl restart netdata
root@localhost:~# sudo systemctl status netdata
● netdata.service - netdata - Real-time performance monitoring
  Loaded: loaded (/usr/lib/systemd/system/netdata.service; enabled; preset: enabled)
  Active: active (running) since Tue 2025-10-28 10:47:26 UTC; 6s ago
    Docs: man:netdata
          file:///usr/share/doc/netdata/html/index.html
          https://github.com/netdata/netdata
 Main PID: 5209 (netdata)
   Tasks: 68 (limit: 3894)
  Memory: 32.1M (peak: 33.1M)
    CPU: 2.508s
   CGroup: /system.slice/netdata.service
           ├─5209 /usr/sbin/netdata -D
           ├─5214 /usr/sbin/netdata --special-spawn-server
           ├─5474 /usr/lib/netdata/plugins.d/nfacct.plugin 1
           ├─5479 bash /usr/lib/netdata/plugins.d/tc-qos-helper.sh 1
           ├─5494 /usr/lib/netdata/plugins.d/apps.plugin 1

Oct 28 10:47:26 localhost.localdomain systemd[1]: Started netdata.service - netdata - Real-time performance monitori>
Oct 28 10:47:26 localhost.localdomain netdata[5209]: 2025-10-28 10:47:26: netdata INFO  : MAIN : CONFIG: cannot load>
lines 1-19/19 (END)
```

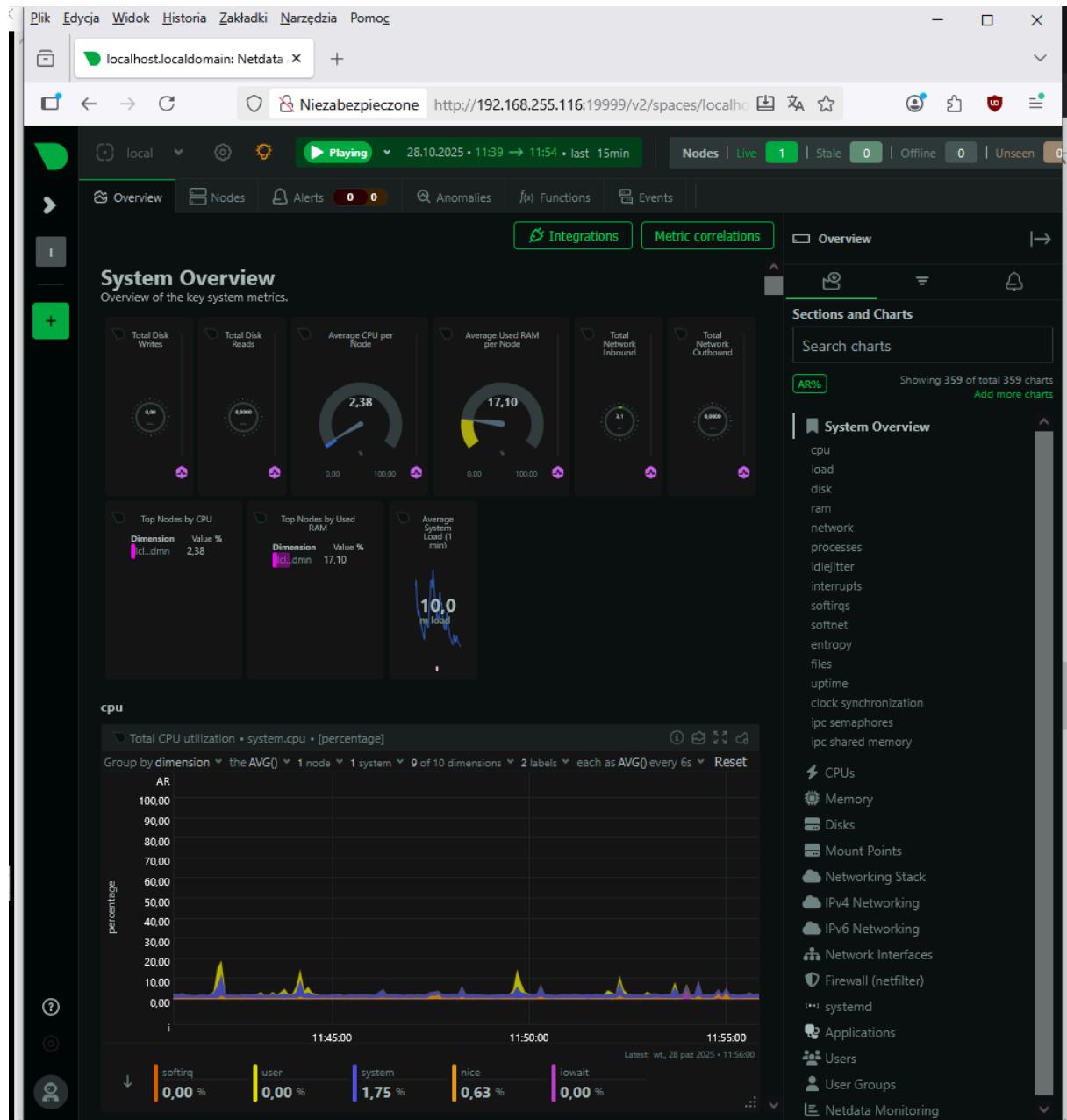
Rysunek 5: Status usługi

Warto zauważyć, że czas działania usługi jest równy 6s co potwierdza pomyślne zrestartowanie.

## 3.3. Monitorowanie zasobów

### 3.3.1. Połączenie z komputera stacjonarnego

Po prawidłowej konfiguracji usługi połączono się z komputera stacjonarnego na adres <http://192.168.255.116:19999>.



Rysunek 6: Interfejs graficzny programu

W wyniku połączenia na ekranie monitora pojawił się panel główny programu NetData. Wyszczególnić na nim można panel metryk pokazujący zużycie zasobów w czasie rzeczywistym, wykresy ze zmianami zużycia z upływem czasu oraz interfejs nawigacji.

### 3.3.2. Program stress

Aby zaobserwować zmiany w zużyciu zasobów na urządzeniu Raspberry wykorzystane zostało narzędzie symulujące obciążenie *stress*. Instalacja przebiegła w ten sam sposób co instalacji usługi NetData, komendą **sudo apt-get update && sudo apt-get install stress**. Wykorzystując to narzędzie przeprowadzono testy obciążenia na procesorze, pamięci jak i dysku.

```
root@localhost:~# apt-get update && apt-get install stress
Hit:1 https://github.armbian.com/configng stable InRelease
Hit:2 https://packages.microsoft.com/repos/code stable InRelease
Hit:4 http://ports.ubuntu.com noble InRelease
Hit:5 http://ports.ubuntu.com noble-security InRelease
Hit:6 http://ports.ubuntu.com noble-updates InRelease
Hit:3 http://armbian.lv.auroradev.org/apt noble InRelease
Hit:7 http://ports.ubuntu.com noble-backports InRelease
Reading package lists... Done
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
stress is already the newest version (1.0.7-1).
0 upgraded, 0 newly installed, 0 to remove and 116 not upgraded.
root@localhost:~#
```

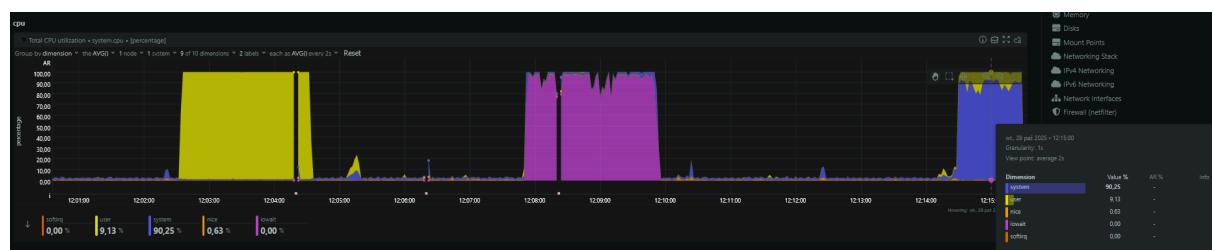
Rysunek 7: Instalacja programu stress

### 3.3.3. Stress test pamięci

W pierwszej kolejności dokonano symulacji obciążenia pamięci urządzenia wykorzystując komendę **stress -vm 10 -timeout 60**



Rysunek 8: Wykres obciążenia pamięci

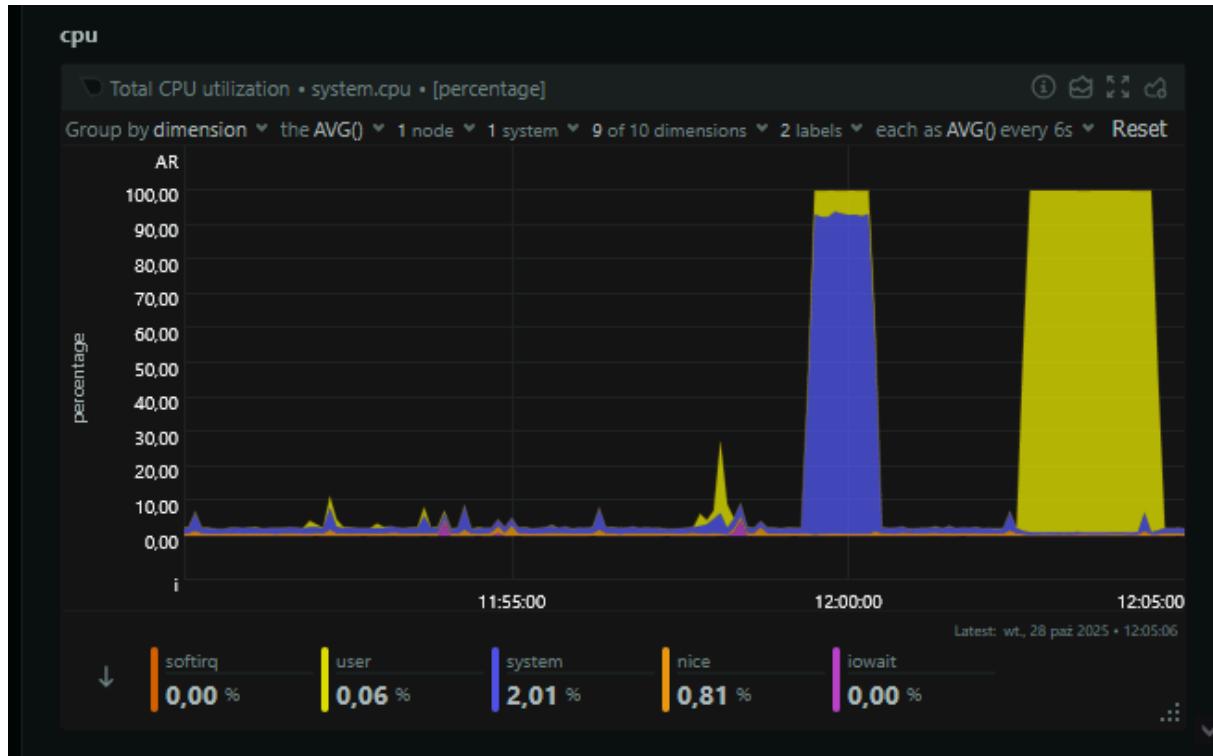


Rysunek 9: Wykres obciążenia procesora

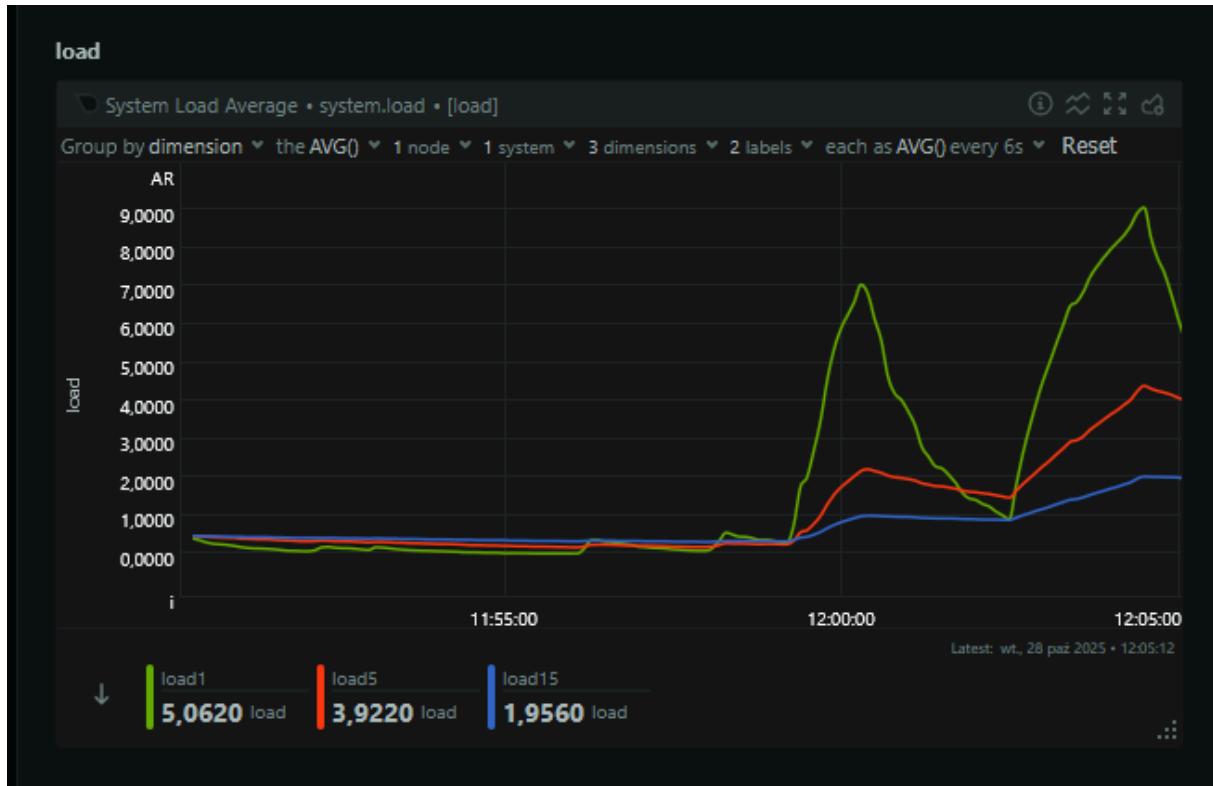
Podczas stress testu zauważono drastyczny wzrost pamięci używanej oraz buforowanej. Dodatkowo spowodowane było wysokie zużycie procesora. Powodem tego najpewniej był fakt, iż procesor jest odpowiedzialny za zarządzanie operacjami na danych w pamięci systemu

### 3.3.4. Stress test procesora

Tym razem użyta została komenda **stress -cpu 10 -timeout 120**



Rysunek 10: Wykres obciążenia procesora

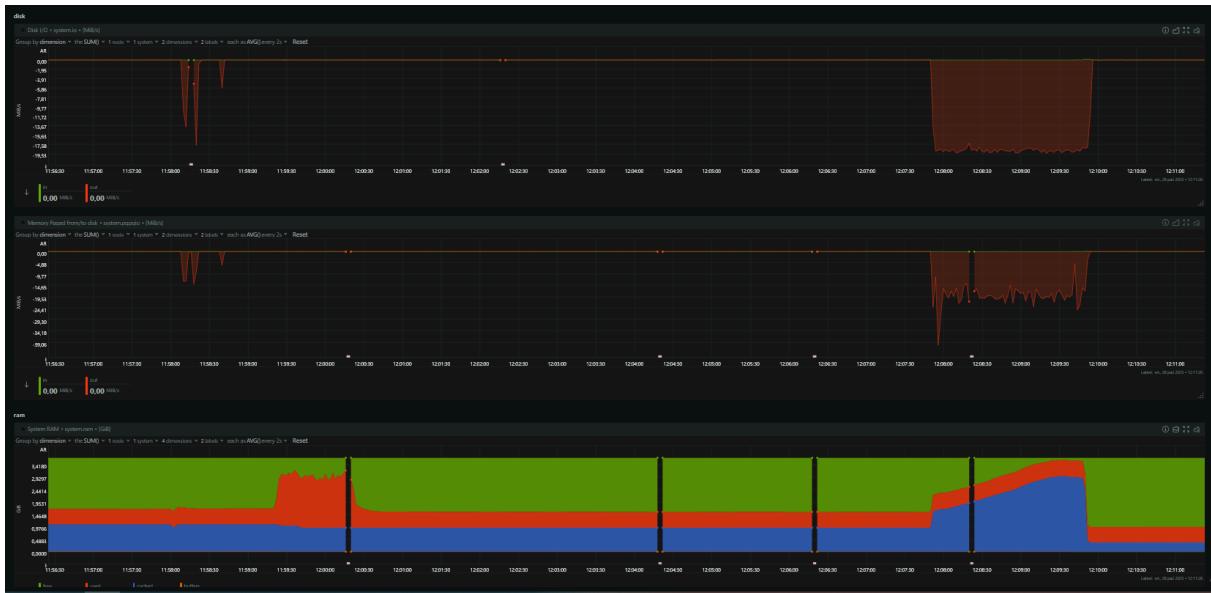


Rysunek 11: Wykres obciążenia procesora

W wyniku stress testu zauważono wysokie zużycie procesora przez procesy użytkownika oraz przez system. softirq czyli żądania przerwania pozostały bez zmian. Pozostałe zasoby urządzenia również nie zostały dotknięte gdyż praca procesora nie wymaga wysokiego udziału pamięci ani dysku.

### 3.3.5. Stress test dysku

Obciążenie dysku zasymulowane zostało komendą **stress -hdd 10 -timeout 120**



Rysunek 12: Wykresy obciążenia dysku oraz pamięci



Rysunek 13: Wykres obciążenia procesora

Końcowo stress test dysku pokazał wysoką ilość zapisu oraz odczytu danych. Poza tym zauważać można zmiany również w obciążeniu procesora jak i pamięci. Powodem tego jest to, iż procesor jest odpowiedzialny za akcje zapisu/odczytu, a w pamięci buforowane są dane używane w tych akcjach.

## 4. Wnioski

Na laboratoriach wykorzystaliśmy program NetData do badania obciążenia urządzenia Raspberry Pi w ramach testów, aczkolwiek program ten nie jest ograniczony do systemów Linux. Może być wykorzystany również na systemach Windows, freeBSD czy MacOS co rozszerza jego aplikacje. Od obserwowania nacisku na zasoby przy budowaniu oprogramowania po monitorowanie zasobów wszelakich serwerów. Uzyskano również wiedzę, jak obciążenie różnych komponentów urządzenia wpływa na resztę systemu.

## **5. Odpowiedzi na dodatkowe pytania**

### **5.1. Pytanie 1.**

**Czy możesz wskazać jakąś sytuację ze swojego doświadczenia, w której takie narzędzie mogłoby być przydatne? Na przykład podczas debugowania aplikacji C++?**

Dobrym przykładem takiej sytuacji może być hostowanie publicznych serwerów na wysoką lub niską skalę. Stałe monitorowanie zasobów jest niezbędne do zapewnienia stałej i płynnej pracy serwera.

### **5.2. Pytanie 2.**

**Jakie rodzaje błędów programistycznych można wychwycić za pomocą narzędzi monitorujących?**

- Wysokie wykorzystanie procesora może wskazywać na nie wydajne algorytmy lub zbyt wysokie obciążenie pojedynczych wątków.
- Obciążenie pamięci może oznaczać, że limity struktur danych nie są określone lub zbyt wysoka ilość obiektów jest buforowana w pamięci do późniejszego użycia.
- Abnormalne zużycie dysku może być spowodowane złym zarządzaniem operacji zapisu oraz odczytu, niepotrzebny powtarzaniem tych operacji lub braku czyszczenia plików tymczasowych.

### **5.3. Pytanie 3.**

**Jak zorganizowałbyś logowanie metryk w dużej infrastrukturze? A co by było, gdyby było 5 serwerów? 50? 500?**

Przy małej skali najpewniej wystarczyłoby zainstalowanie narzędzia lokalnie na każdym z serwerów i monitorowanie każdego z paneli osobno. Jak mówimy o większej skali, indywidualne monitorowanie nie wystarczy. Najefektywniej byłoby skonfigurować wysyłanie wyników obciążenia zasobów do chmury lub centralnego serwera, po czym dokonać agregacji na tych danych dla prostego odczytu.