Monitoring

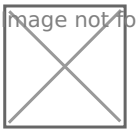
Different tools to monitor your Homelab and Network services.

- Install Glances to Monitor Open Media Vault Server
- Install Uptime Kuma - A fancy self-hosted monitoring tool
- Install Umami to Monitor your Website Traffic
- Install Netdata to Monitor Debian or Ubuntu servers
- Command Line Utilities
    - Install Bpytop for Monitoring Linux
- Monitor Your System with Grafana using Netdata and Prometheus

Install Glances to Monitor Open Media Vault Server

Glances is a cross-platform system monitoring tool written in Python. It can be installed on pretty much any Linux system but this is a great tool for monitoring NAS servers like Open Media Vault. You can see vital information like CPU temps, disk space, RAM and CPU usage and much more.

You can see most of this from the OMV dashboard but I wanted something that could read temperature sensors for the CPU and this is perfect. Plus, looking at the bright white dashboard on OMV can be strenuous on the eyes as they do not have a dark mode.




```
apt install python3
```

```
apt install python3-pip
```

```
pip3 install glances
```

Usage

```
glances
```

Faster refresh

```
glances -t0
```

View the web UI.

```
glances -t0 -w
```

Did you find this helpful? Subscribe to me on Youtube for more content!

Install Uptime Kuma - A fancy self-hosted monitoring tool

Uptime Kuma is a great way to monitor your self hosted apps and services.

- Monitoring uptime for HTTP(s) / TCP / Ping / DNS Record.
- Fancy, Reactive, Fast UI/UX.
- Notifications via Telegram, Discord, Gotify, Slack, Pushover, Email (SMTP), and 70+ notification services, click here for the full list.
- 20 seconds interval.

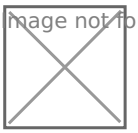
Installation

Paste this docker compose stack into Portainer

```
version: '3.3'

services:
  uptime-kuma:
    image: louislam/uptime-kuma
    container_name: uptime-kuma
    volumes:
      - /docker/uptimekuma:/app/data
    ports:
      - 3001:3001
```

Did you find this helpful? Subscribe to me on Youtube for more content!

Install Umami to Monitor your Website Traffic



Umami is a simple, easy to use, self-hosted web analytics solution. The goal is to provide you with a friendlier, privacy-focused alternative to Google Analytics and a free, open-sourced alternative to paid solutions. Umami collects only the metrics you care about and everything fits on a single page. You can view a live demo here or read more about Umami here.

Install Filebrowser (if you want to cheat and not use CLI)

```
version: "2.1"
services:
  filebrowser:
    image: hurlenko/filebrowser:latest
    container_name: filebrowser
    environment:
      - FB_BASEURL=/f
    volumes:
      - /:/data
      - /docker/filebrowser:/config
    ports:
      - 8081:8080
    restart: unless-stopped
```

Create the schema.postgresql.sql file and place it in /docker/umami

Paste the following into the schema.postgresql.sql file.

```
schema.postgresql.sql
```

```
drop table if exists event;
drop table if exists pageview;
drop table if exists session;
```

```
drop table if exists website;
drop table if exists account;

create table account (
    user_id serial primary key,
    username varchar(255) unique not null,
    password varchar(60) not null,
    is_admin bool not null default false,
    created_at timestamp with time zone default current_timestamp,
    updated_at timestamp with time zone default current_timestamp
);

create table website (
    website_id serial primary key,
    website_uuid uuid unique not null,
    user_id int not null references account(user_id) on delete cascade,
    name varchar(100) not null,
    domain varchar(500),
    share_id varchar(64) unique,
    created_at timestamp with time zone default current_timestamp
);

create table session (
    session_id serial primary key,
    session_uuid uuid unique not null,
    website_id int not null references website(website_id) on delete cascade,
    created_at timestamp with time zone default current_timestamp,
    hostname varchar(100),
    browser varchar(20),
    os varchar(20),
    device varchar(20),
    screen varchar(11),
    language varchar(35),
    country char(2)
);

create table pageview (
    view_id serial primary key,
    website_id int not null references website(website_id) on delete cascade,
    session_id int not null references session(session_id) on delete cascade,
```

```
    created_at timestamp with time zone default current_timestamp,
    url varchar(500) not null,
    referrer varchar(500)
);

create table event (
    event_id serial primary key,
    website_id int not null references website(website_id) on delete cascade,
    session_id int not null references session(session_id) on delete cascade,
    created_at timestamp with time zone default current_timestamp,
    url varchar(500) not null,
    event_type varchar(50) not null,
    event_value varchar(50) not null
);

create index website_user_id_idx on website(user_id);

create index session_created_at_idx on session(created_at);
create index session_website_id_idx on session(website_id);

create index pageview_created_at_idx on pageview(created_at);
create index pageview_website_id_idx on pageview(website_id);
create index pageview_session_id_idx on pageview(session_id);
create index pageview_website_id_created_at_idx on pageview(website_id, created_at);
create index pageview_website_id_session_id_created_at_idx on pageview(website_id, session_id,
created_at);

create index event_created_at_idx on event(created_at);
create index event_website_id_idx on event(website_id);
create index event_session_id_idx on event(session_id);

insert into account (username, password, is_admin) values ('admin',
'$2b$10$BUli0c.muyCW1ErNJc3jL.vFRFtFJWrT8/GcR4A.sUdCznaXiqFXa', true);
```

Run the docker stack and install

```
version: '3'
services:
  umami:
    image: ghcr.io/mikecao/umami:postgresql-latest
```

```
    ports:
      - "3000:3000"
    environment:
      DATABASE_URL: postgresql://umami:umami@db:5432/umami
      DATABASE_TYPE: postgresql
      HASH_SALT: H6ei6O1tdLNxIQLRs4Mw
    depends_on:
      - db
    restart: always
  db:
    image: postgres:12-alpine
    environment:
      POSTGRES_DB: umami
      POSTGRES_USER: umami
      POSTGRES_PASSWORD: umami
    volumes:
      - /docker/umami/schema.postgresql.sql:/docker-entrypoint-
initdb.d/schema.postgresql.sql:ro
      - /docker/umami/db:/var/lib/postgresql/data
    restart: always
volumes:
  umami-db-data:
```

Connect to the web UI

Go to your.server.ip.here:3000 and log in using admin as the username and umami as the password.

Video tutorial

https://www.youtube.com/embed/nUjDGxazkOQ?ab_channel=Geeked

Did you find this helpful? Subscribe to me on Youtube for more content!

Install Netdata to Monitor Debian or Ubuntu servers

Netdata is a great way to monitor your Debian based systems and servers. It's feature rich and packs all the punches any system administrator needs. Installation is a breeze and it can be up and running in less than 2 or 3 minutes.

Netdata can even be used to monitor Virtual Machines and Containers on a Proxmox server as you can see in the screenshot below. See the lxc containers on the right side navigation menu.

Install Netdata

A simple one line command. This requires curl to be installed on the system.

```
apt install curl -y
```

```
bash <(curl -Ss https://my-netdata.io/kickstart.sh) -y
```

Start Using Netdata

To start using Netdata, open a browser and navigate to http://NODE:19999, replacing NODE with either localhost or the hostname/IP address of a remote node.

Where you go from here is based on your use case, immediate needs, and experience with monitoring and troubleshooting.

Installing lm Sensors

To view CPU temperatures, you will have to install lm sensors. lm-sensors provides a hardware health monitoring driver for Linux. It's used by system administrators to check the health status of their hardware. It is also used to monitor the hardware infrastructure in servers and be very valuable in mission critical applications.

```
apt install lm-sensors
```

Refresh the Netdata web UI and you should see a new section called Sensors. There you can see the CPU temperatures.

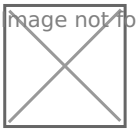
Image not found or type unknown

Command Line Utilities

Command Line Utilities

Install Bpytop for Monitoring Linux

Bpytop is amazing. I love this utility for monitoring CPU temps/usage, memory usage, network throughput and processes. It also has a wonderful display for hard drive usage.



Just look at how beautiful it is. Click the image above to enlarge.

Install

Make sure python3 is installed. It should be on Debian. If not run

```
apt install python3-pip
```

```
apt install bpytop
```

Usage

open a terminal ans imply type:

```
bpytop
```

Press esc on your keyboard to view and navigate options such as themes and other customizations.

Example video:

Monitor Your System with Grafana using Netdata and Prometheus

Prometheus is quite amazing! Here I show you how to use it to pull metrics from the Netdata API to display in a custom dashboard with Grafana.

The idea behind this project is to have a custom dashboard that displays only things you really want to see at a glance. You can choose to dig through the Prometheus Netdata metrics more to display anything you wish but below is the example I setup for this tutorial.

Prometheus does all the work. The only thing required to be installed on the machine being monitored is Netdata. No other agents or workers are needed that could utilize more system memory on your servers. That's the beauty of Prometheus. Rather than coming to your machine to collect data, Prometheus waits for data to come to it instead, using API calls.

> Netdata is required to be installed on any machine that will be monitored. Please see this video before moving forward here.

I chose to host Grafana and Prometheus on their own separate LXCcontainer using Proxmox. You can use a RPi or any host you wish. I do this so I can use one central host to keep things more organized. You only need one host for Grafana and Prometheus.

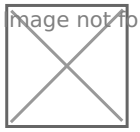
Install Grafana

I use a docker stack through Portainer.

```
version: '3.3'
services:
    grafana:
```

```
        ports:
            - '3000:3000'
        container_name: grafana
        image: grafana/grafana
```

Grafana will then be accessible on port 3000. Login with admin/admin then choose a stronger password.



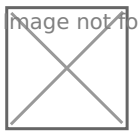
Install Prometheus

Again, I use a docker stack through Portainer.

```
version: '3.3'
services:
    prometheus:
        ports:
            - '9090:9090'
        volumes:
            - '/docker/prometheus:/etc/prometheus'
        image: prom/prometheus
```

This will create a folder on your host machine at /docker/prometheus

Prometheus will then be accessible on port 9090.



Create the prometheus.yml file

```
cd /docker/prometheus
```

```
touch prometheus.yml
```

Edit the prometheus.yml file

```
nano /docker/prometheus/prometheus.yml
```

Below is my example prometheus.yml file. You should change the IP to match that of your Netdata web UI. You should not have to change anything above the pound line, only that in between.

```
# my global config
global:
  scrape_interval:     15s # Set the scrape interval to every 15 seconds. Default is every 1
minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.
  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration
alerting:
  alertmanagers:
  - static_configs:
    - targets:
      # - alertmanager:9093

# Load rules once and periodically evaluate them according to the global
'evaluation_interval'.
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this
config.
  - job_name: 'prometheus'

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

    static_configs:
    - targets: ['localhost:9090']

################################################################################################
###
```

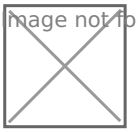
```
  - job_name: 'netdata-scrape'

    metrics_path: '/api/v1/allmetrics'
    params:
      # format: prometheus | prometheus_all_hosts
      # You can use `prometheus_all_hosts` if you want Prometheus to set the `instance` to
your hostname instead of IP
      format: [prometheus]
      #
      # source: as-collected | raw | average | sum | volume
      # default is: average
      #source: [as-collected]
      #
      # server name for this prometheus - the default is the client IP
      # for Netdata to uniquely identify it
      #server: ['prometheus1']
    honor_labels: true
    static_configs:
      - targets: ['192.168.1.13:19999']

##############################################################################################
###
```

If you decide to monitor more than one Netdata instance just copy and paste from each pound line then edit the job name and target IP.

Prometheus Netdata Metrics

You can browse the metrics explorer by pressing the small globe icon next to the search bar or use the metrics I used in the video to start your dashboard. You can see those below.

On the Prometheus webui, copy and paste the following metrics in the finder then copy the query you wish to display in Grafana.



System Uptime

```
netdata_system_uptime_seconds_average
```

When adding this to Grafana be sure to select "seconds (s)" as the unit of measurement under Standard options.

CPU Temperature (requires sensors to be installed on the server using lm-sensors)

```
netdata_sensors_temperature_Celsius_average
```

When adding this to Grafana be sure to select "Celsius (°C)" as the unit of measurement under Standard options.

CPU Usage

```
netdata_cpu_cpu_percentage_average
```

When adding this to Grafana be sure to select "Percent (0-100)" as the unit of measurement under Standard options.

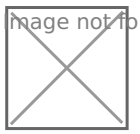
Memory Used

This was a tricky one. I had to take total memory and subtract the available memory from it to get an accurate number.

```
31970 - netdata_mem_available_MiB_average{instance="192.168.1.13:19999", job="netdata-scrape"}
```

When adding this to Grafana be sure to select "mebibytes" as the unit of measurement under Standard options.

As you can see, I have 32GB of RAM. I had to play with the numbers and used bpytop to compare. I was able to get the RAM spot on to match the same output in bpytop.



Hard Drive Space

```
netdata_disk_space_GiB_average
```

When adding this to Grafana be sure to select "gibibytes" as the unit of measurement under Standard options.

If you'd like to look at the JSON for my Grafana dashboard, please view it here.

Video Tutorial

https://www.youtube.com/embed/uimGcQVRaqI

Did you find this helpful? Subscribe to me on Youtube for more content!