munin2smartphone

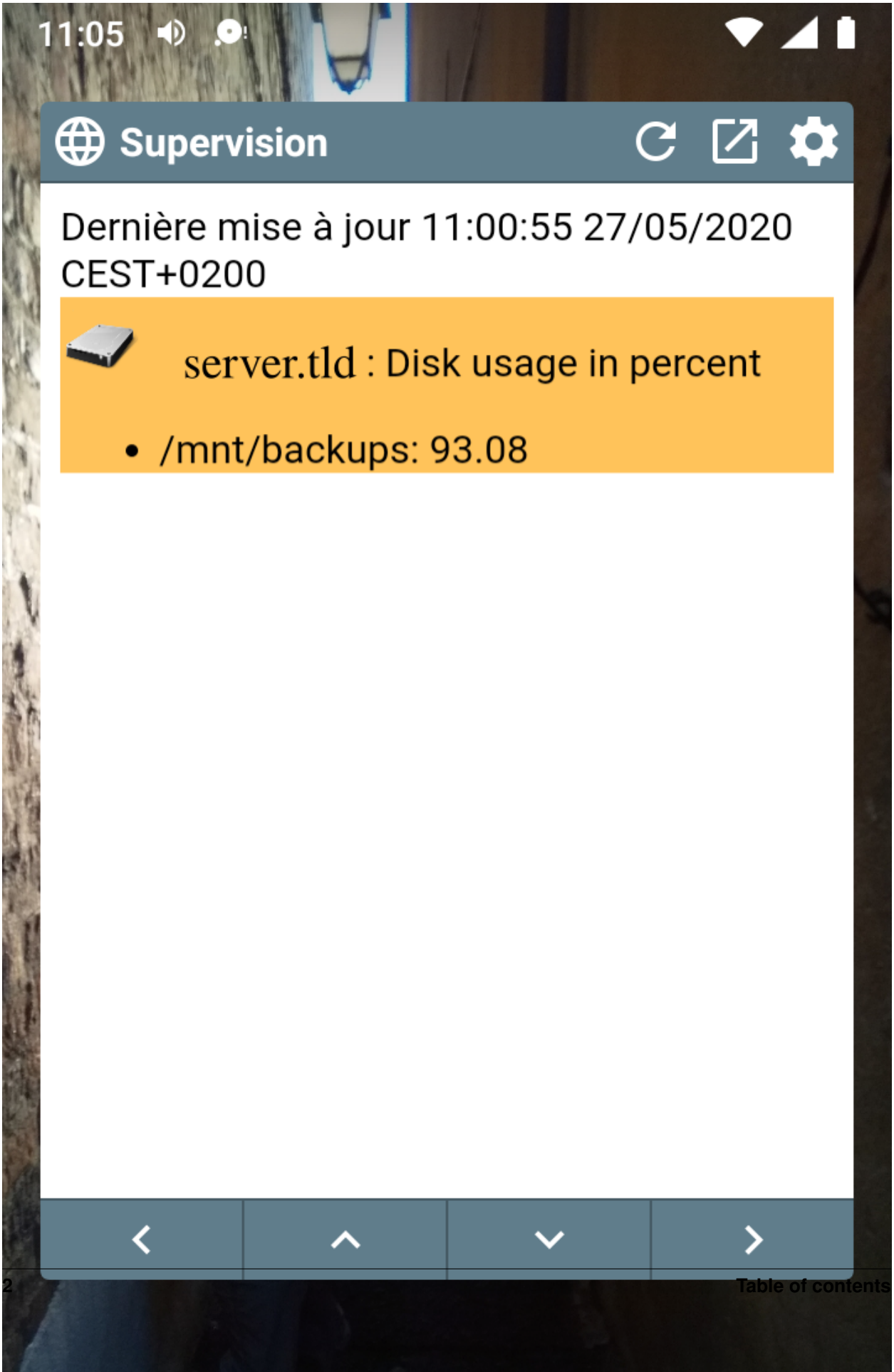
Release 0.0.1

May 28, 2020

Table of contents

Demilitarized HTTP server eating monitoring data[1] and producing static pages that can be displayed securely (ie. without javascript).

[1] Currently supported: munin. Any stream of monitoring data should be parseable by *munin2smartphone*.


11:05
Supervision
Dernière mise à jour 11:00:55 27/05/2020 CEST+0200
server.tld : Disk usage in percent
/mnt/backups: 93.08

CHAPTER 1

Overview

1.1 What you get

A non intrusive, permanent view of the important things.

You get to select what you see and what is filtered out.

1.2 Architecture

A *munin2smartphone* setup consists of 3 bricks:

a Munin supervision server.

We add a trigger and a bit of configuration to your stock munin installation, so that it pushes the state of all known checks to the next brick.

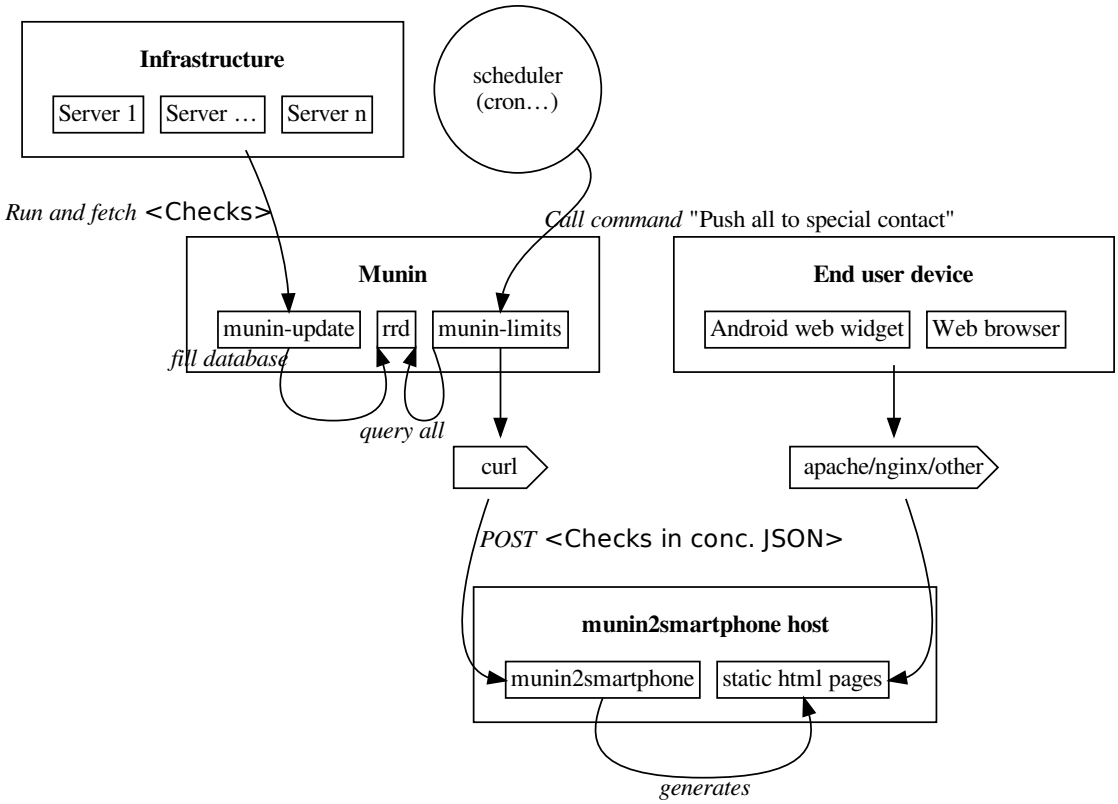
a *munin2smartphone* daemon.

The daemon is receiving authenticated data and generating static web pages.

a frontend to view the web pages.

Whether a web browser or an android widget.

1.3 Information flow



CHAPTER 2

Installation guide

Table of Contents

2.1 On the munin2smartphone + static html server

Note that this guide:

- Uses nginx as a static web server and reverse proxy,
- Stores static HTML pages to `/var/lib/munin2smartphone`,
- Serves those pages on URL `https://your.fqdn.example.htmlreports`,

- Makes *munin2smartphone* listen on address `127.0.0.1` and port `8765`,
- Receives data on URL `https://your.fqdn.example/pushdatahere`.

2.1.1 Build the Public Key Infrastructure

We are going to authenticate our data broker with an HTTPS-client certificate.

Here is a step-by-step using Debian 10 and easy-rsa 3 as a crypto-tools wrapper. Feel free to adapt (hopefully this is accurate).

```
sudo apt install easy-rsa
mkdir munin2smartphone-pki
cd munin2smartphone-pki
cp /usr/share/easy-rsa/easyrsa .
./easyrsa init-pki
./easyrsa build-ca
```

Deploy the CA to nginx:

```
sudo mkdir -p /etc/nginx/client_certs/
sudo cp ./pki/ca.crt /etc/nginx/client_certs/
```

Build the client cert (you will be prompted for a password, which you will need later):

```
./easyrsa build-client-full httpmunin
```

You want to transfer `./pki/private/httpmunin.key`, `./pki/issued/httpmunin.crt` and `./pki/ca.crt` to the munin host. You may safely delete `./pki/private/httpmunin.key` from this place.

2.1.2 Configure 2 nginx locations

We will serve the static pages and forward ssl-verified requests to *munin2smartphone* through nginx.

We need to add 2 location directives and specify the ssl CA, something like:

```
server {
    index index.html index.htm index.nginx-debian.html;

    server_name your.fqdn.example;

    # [...]

    # ------ Copy-paste and adapt below ------

    # Here, the static html location
    location ~ ^/htmlreports(/.*)$ {
        alias /var/lib/munin2smartphone$1;
    }

    # Now, make client ssl verification optional
    # and define the forwarding location

    # client certificate
    ssl_client_certificate /etc/nginx/client_certs/ca.crt;
    # make verification optional, so we can display a 403 message
```

(continues on next page)

(continued from previous page)

```
    # to those who fail authentication (= forbidden)
    ssl_verify_client optional;

    location /pushdatahere/ {
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header        X-Forwarded-Proto $scheme;
            proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
            if ($ssl_client_verify != SUCCESS) {
                    return 403;
            }
            proxy_pass      http://127.0.0.1:8765;
    }

    # ------ End of the new sections ------

    # [...]

}
```

2.1.3 Install *munin2smartphone*

You probably want to use a virtualenv with Python 3.8 (3.6 at least) and

```
pip install munin2smartphone
```

Now test-run *munin2smartphone* in a terminal!

Real system installation: To be documented (help welcome). I plan on:

- using a systemd unit to start the daemon,
- creating a debian package of *munin2smartphone* and deps.

2.1.4 Run munin2smartphone with relevant options and configuration

This is covered in another section of the documentation: *options_config*.

2.2 On the munin host

2.2.1 HTTPS POST with curl

At the *end of the PKI step-by-step*, we transfered 3 files to this host.

Now ensure that their names and location match the script below, that is:

- `/etc/munin/httpmunin-ca.crt` (initially `ca.crt`)
- `/etc/munin/httpmunin.crt`
- `/etc/munin/httpmunin.key`

Listing 1: `/usr/local/bin/push-munin.sh` - This file should be readable and executable by the user `munin`.

```
#!/bin/bash

# We'll receive data from stdin (piped into this script).
sed "s,/,\\/,g" | \
/usr/bin/curl \
--cacert /etc/munin/httpmunin-ca.crt \
--cert /etc/munin/httpmunin.crt \
--key /etc/munin/httpmunin.key \
--pass superpass \
https://your.fqdn.example/pushdatahere/ \
-d @-
```

2.2.2 Call to munin-limits every 5 minutes

Let's code a very simple daemon.

`munin-limits` will evaluate the state of every known check and take the configured action.

Listing 2: `/usr/local/bin/call-munin-limits.py` - This file should be readable and executable by the user `munin`.

```
#!/usr/bin/python3

import time
import subprocess

INTERVAL = 300 # seconds = 5 minutes
COMMAND = [
    "/usr/share/munin/munin-limits",
    "--contact",
    "widget",
    "--force",
    "--always-send",
    "warning,critical",
]


def main():
    while True:
        print("Calling {}".format(' '.join(COMMAND)))
        process = subprocess.Popen(COMMAND)
        process.wait()
        print("Exit code: {}".format(process.returncode))
        time.sleep(INTERVAL)

if __name__ == "__main__":
    main()
```

2.2.3 Configure the service

Listing 3: /etc/systemd/system/push-munin.service

```
[Unit]
Description=Triggers a push of all munin states in json

[Service]
ExecStart=/usr/local/bin/call-munin-limits.py
User=munin

[Install]
WantedBy=multi-user.target
```

Then run

```
systemctl daemon-reload
systemctl enable push-munin
systemctl start push-munin
```

2. Add a contact in munin, and specify the concatenated JSON format

```
contact.widget.command /usr/local/bin/push-munin.sh
contact.widget.text \
        { \
                "group":"${var:group}", \
                "host":"${var:host}", \
                "graph_category":"${var:graph_category}", \
                "graph_title":"${var:graph_title}", \
                "warning":[  ${loop<,>:wfields { \
                        "label":"${var:label}", \
                        "value":"${var:value}", \
                        "w":"${var:wrange}", \
                        "c":"${var:crange}", \
                        "extra":"${var:extinfo}" \
                } } ], \
                "critical":[ ${loop<,>:cfields { \
                        "label":"${var:label}", \
                        "value":"${var:value}", \
                        "w":"${var:wrange}", \
                        "c":"${var:crange}", \
                        "extra":"${var:extinfo}" \
                } } ], \
                "unknown":[  ${loop<,>:ufields { \
                        "label":"${var:label}", \
                        "value":"${var:value}", \
                        "w":"${var:wrange}", \
                        "c":"${var:crange}", \
                        "extra":"${var:extinfo}" \
                } } ] \
        }
```

2.2.4 On the smartphone

On android, you can use this widget.

Configure a view to your static web pages with an automatic reload.

2.3 Notes

on the refresh interval

I chose to use a 5 minutes interval everywhere. 5 minutes is the default polling interval for munin.

Should you wish to change this, you need to change the interval in:

- munin update (cron or systemd timer),
- *call-munin-limits.py*,
- your web viewer (browser or *smartphone*).

on the munin contact name

We are going to configure a contact named "*widget*" in `munin`. This contact will trigger a shell script pushing data over HTTPS with curl. Every 5 minutes, we will shoot an event so that munin processes the data and sends it to our `munin2smartphone` daemon.

Should you want to use another name than *widget* for the contact, change it

- in the *munin config*
- and in *call-munin-limits.py*,

CHAPTER 3

Running munin2smartphone

Table of contents

3.1 Command line options

(section in the works)

optional arguments:

-h, --help show this help message and exit

-V, --version show program's version number and exit

-v, --verbose increase output verbosity -can be called multiple times: Levels: 0 time: error, 1 time: warning, 2 times: info, 3 times:debug

--coloredlogs terminal logs are colored (using coloredlogs): default

--no-coloredlogs terminal logs are NOT colored: default is colored logs

-o OUTPUTDIR, --outputdir OUTPUTDIR html output directory (default: /home/feth/munin2smartphone)

–configfile CONFIGFILE –logfile LOGFILE –cache_directory CACHE_DIRECTORY

munin2smartphone cache directory (defaults to /home/feth/.cache/munin2smartphone)

--port PORT listening TCP port (defaults to 8765)

--listening-address LISTENING_ADDRESS listening IP address (defaults to 127.0.0.1)

--timezone TIMEZONE Timezone for html output -internal dates are UTC (defaults to Europe/Paris)

3.2 Configuration file

Configuration files are on the roadmap but are not handled yet.

CHAPTER 4

How to hack this project

4.1 Setup your environment

This project is managed with *poetry*, you don't need an explicit virtualenv.

- Install poetry.
- then fork the project here https://framagit.org/feth/munin2smartphone (or ask me for dev status, or clone my repo and send me diffs)
- then

```
git clone git@path.to.your/repo/munin2smartphone.git
cd munin2smartphone
poetry install
```

4.2 Run your code

Generally: .. code-block:: bash

> poetry run <whatever is installed with the project> [options] poetry run munin2smartphone [options] poetry run python # This will run the Python version associated with the project.

4.3 Run the tests

There are no tests! This is a shame, please help us!

CHAPTER 5

munin2smartphone

5.1 munin2smartphone package

5.1.1 Submodules

5.1.2 munin2smartphone.config module

5.1.3 munin2smartphone.datastore module

5.1.4 munin2smartphone.entrypoints module

5.1.5 munin2smartphone.exceptions module

exception munin2smartphone.exceptions.**ConfigError**
Bases: *munin2smartphone.exceptions.Munin2SmartphoneException*

exception munin2smartphone.exceptions.**Munin2SmartphoneException**
Bases: Exception

5.1.6 munin2smartphone.server module

5.1.7 munin2smartphone.utils module

munin2smartphone.utils.**purge_none**(*data*)
Delete keys with the value None in a dictionary, recursively.

fixed a stackoverflow algo

5.1.8 munin2smartphone.widget module

5.1.9 Module contents

munin2widget

HTTP server converting munin-json to html reports that you can display on your smartphone (html widget)

CHAPTER 6

Glossary

Munin Munin is an IT monitoring tool. It is very easy to extend in order to monitor anything.

RRD Round Robin Database - actually the concept is not used here but we use the word.

Python Module Index

m

Index