Netwark Documentation

Release 1.0.0

Michael Vieira

May 30, 2019

Contents

CHAPTER 1

Architecture design

CHAPTER 2

Installation

2.1 Prerequisites

Netwark is designed for Linux/Unix systems. Running the webserver or a worker on Windows are not officially supported and we haven't tested it. Running the worker on the *Windows Subsystem Linux* not work because some tools need *raw sockets manipulation*, not available on the current version of WSL.

If you still want to deploy Netwark on a Windows environment, you can use Docker. It can resolve the problem of OS incompatibility, but it can be painful for deploying the networking part.

Because Netwark is mainly focused for an internal use, it doesn't have (yet) any authentication et authorization mechanisms. We do not recommend to expose the webserver on Internet to avoid attacks.

If your infrastructure allows you to create a dedicated network connected to all your regions, we recommend to do it.

2.1.1 Infrastructure prerequisites

Netwark need some external services for storing the data and for communicating with the entire network of workers.

Netwark need these services installed on the same host of the webserver or on separate servers (recommended):

- RabbitMQ >= 3.5
- PostgreSQL >= 9
- Internet connectivity. This point seems stupid, but if you want to retrieve information from public resources, you need Internet. More information are added on the next sections.

For each nodes, you need to install:

- Python >= 3.5, we use python types, not working with previous versions
- Poetry, Poetry seems largely better than Pipenv/Pipfile
- PostgreSQL libs (needed for communicating with the database)
- Supervisord or Systemd

- Ping utility (we use it for… pinging machines)
- WHOIS utility (we use it in synchronous and asynchronous tasks for retrieving information of a public resource of Internet)
- dig utility (we use it for retrieving informations from DNS zones and for reverse DNSs)

If you want to use Docker, install on your machines:

- Docker >= 18.08
- Docker-compose

2.1.2 Webserver prerequisites

Because the webserver doesn't run magic tools (only normal stuff), the webserver can be installed on a Windows machine but it's recommended.

You need to install on the host in addition of the packages specified in the last section:

- NodeJS (for retrieving and handling frontend assets)
- UWSGI with uwsgi-python
- A reverse proxy server (e.g. Apache, Nginx). Exposing the uwsgi/waitress are not recommended.
- Mapbox account for showing the maps
- Internet connectivity. We need Internet for all *synchronous tasks* and for updating the database (MAC OUI database) and retrieving new versions of *Maxmind databases*.

2.1.3 Worker prerequisites

The worker is a magical part of the project that listen constantly RabbitMQ queues that his assigned, ready to run the instructions sent through into the queues.

The worker need some magical tools and the listen can increase with the time and the next releases.

For this release, each machine hosting a worker need:

- *mtr*, basically a much better *traceroute/ping* utility.
- *ping*, for receiving `pong` from other machines
- Internet connectivity. Unlike the webserver, the worker doesn't need big requirements in term of bandwidth and traffic needs. In case you are in a cloud environment, you can dedicate few gigabytes (1-2GB) of traffic per months. Of course, it will depend with the usage you will have with Netwark.

2.2 Install on a server

We consider you already have deployed RabbitMQ and PostgreSQL and have installed everything on your servers.

2.2.1 General instructions

The very beginning step is to clone or download an archive of the latest version of Netwark. If you want to use git, simply clone the repo by using:

```
git clone --branch <tag_name> https://github.com/themimitoof/netwark netwark
cd netwark
```

The list of tags are available on GitHub.

If you prefer using `tar` or `zip` archives, you can download one by using the release page on GitHub.

You can now create a *Python virtualenv* and activate it by using:

```
python3 -m venv venv && source venv/bin/activate
```

Warning: If the module `venv` is not available, you can install it by installing the packet `python3-venv` on Ubuntu/Debian, shipped by default on Fedora/CentOS.

We need now to download `Poetry` for downloading all our dependencies:

```
pip install poetry && \  # Install poetry on the virtualenv
poetry install --no-dev  # Retrieve all dependencies only needed for a production␣
↪environment.
```

Note: If you want to test `master` branch, a in-progress feature or simply contribute to the codebase, you should remove `--no-dev` to the list of arguments sent to `poetry`.

A good part is now finished. You can now follow next sections to finish the deployment of each part of Netwark.

2.2.2 Deploy the Webserver

Before continuing the deployment, we need to create a `.mapbox-token` file at the root folder of Netwark that contain the **access token** of your Mapbox account.

Now, you can download all *front-end* dependencies by using `npm` and *bundle* all resources with:

```
npm i && \
./node_modules/.bin/gulp all
```

You can go now into the `config` folder and create a copy of all files or `netwark_backend.yaml.example` and `<environment>.ini.example`, and remove the `.example` in the extension.

For now, you can edit in `netwark_backend.yaml` the line `broker_url:` and replace the connection string by your *RabbitMQ* credentials.

In the `<environment>.ini` file, replace the connection string in `sqlalchemy.url` with our *PostgreSQL* credentials.

Your installation is almost ready to use! If you want, you can pause the deployment to continue to configuring the webserver by following the configuration page.

Now, we need to run our database migrations scripts. For this, we only need to run the command `alembic -c config/<environment>.ini upgrade head`.

After that, we need to retrieve latest version of *MaxMind DB* and updating the *MAC OUI* database. For this, run theses two commands:

```
python netwark/bin/update_oui_vendor_table.py config/<environment>.ini
python netwark/bin/update_maxmind_db.py config/<environment>.ini
```

Everything is now configured! Congratulations! But we need to configure a last thing system side. Before continuing, you can test if the webserver works by typing the command:

```
pserve config/<environment>.ini
```

You can now open your browser and go to http://localhost:6543.

Use supervisord

You can use supervisord as daemon manager. For this, create a new `netwark-webserver.conf` in `/etc/supervisor/conf.d` folder or add at the end of `/etc/supervisord.conf` file, the below content:

```
[program:netwark-webserver]
command=<uwsgi command>
directory=/opt/netwark ; Replace with the good path

autostart=true
autorestart=true
startretries=20
stdout_logfile=/var/log/netwark/netwark-webserver.log
redirect_stderr=true
```

You can now reload the configuration or restart `supervisord` by typing:

```
pkill -SIGHUP -x supervisord
# or
systemctl restart supervisord
# or
service supervisor restart
# or
/etc/init.d/supervisor restart
```

Now, you should have access to the webserver through your web browser by accessing to http://localhost:6543. If is not, check the logs specified in the `supervisord` configuration file.

Use systemd

The main Linux distributions embed `systemd` by default. To use it, create a new service by creating a new file on `/etc/systemd/system/netwark-webserver.service` and add the below content:

```
[Unit]
Description=Netwark webserver
Requires=Network.target
After=network.target

[Service]
Type=simple
ExecStart=<uwsgi command>
StandardOutput=file:/var/log/netwark/netwark-webserver.log
StandardError=file:/var/log/netwark/netwark-webserver-errors.log
```

You can now check if the service start and work well by using the command `systemctl start netwark-webserver` and by accessing to http://localhost:6543 with your browser.

If the webserver works, you can enable the service to start automatically on boot:

```
systemctl enable netwark-webserver
```

Voilà! You have done the deployment of the werbserver! We recommand now to configure your **reverse proxy** and follow the configuration page to adjust your installation settings.

2.2.3 Deploy the worker

The deployment of the worker is more easier than the webserver because it doesn't need much steps.

First above, you need to go into the `config` folder and create a copy of all files or `netwark_backend.yaml.example` and `<environment>.ini.example`, and remove the `.example` in the extension.

For now, you can edit in `netwark_backend.yaml` the line `broker_url:` and replace the connection string by your *RabbitMQ* credentials.

In the `<environment>.ini` file, replace the connection string in `sqlalchemy.url` with our *PostgreSQL* credentials.

If you dont have runned the database migrations wet, you need to run it by using the command:

```
alembic -c config/<environment>.ini upgrade head
```

The worker is now ready to start! To test before creating the service configuration, you can start it by using the command:

```
python netwark/bin/celery_backend.py config/<environment>.ini
```

Use supervisord

You can use supervisord as daemon manager. For this, create a new `netwark-worker.conf` in `/etc/supervisor/conf.d` folder or add at the end of `/etc/supervisord.conf` file, the below content:

```
[program:netwark-worker]
command=<path to bin folder of your virtualenv>/python /opt/netwark/netwark/bin/
↪celery_backend.py /opt/netwark/config/<environment>.ini
directory=/opt/netwark ; Replace with the good path

autostart=true
autorestart=true
startretries=20
stdout_logfile=/var/log/netwark/netwark-webserver.log
redirect_stderr=true
```

Note: Please take care to replace the path in the configuration file by the good path used in your server.

You can now reload the configuration or restart `supervisord` by typing:

```
pkill -SIGHUP -x supervisord
# or
systemctl restart supervisord
# or
service supervisor restart
# or
/etc/init.d/supervisor restart
```

You can now `tail` the logs file and run a new operation through the web interface or via the *REST API*.

Use systemd

The main Linux distributions embed `systemd` by default. To use it, create a new service by creating a new file on `/etc/systemd/system/netwark-worker.service` and add the below content:

```
[Unit]
Description=Netwark worker daemon
Requires=Network.target
After=network.target

[Service]
Type=simple
ExecStart=<path to bin folder of your virtualenv>/python /opt/netwark/netwark/bin/
↪celery_backend.py /opt/netwark/config/<environment>.ini
StandardOutput=file:/var/log/netwark/netwark-worker.log
StandardError=file:/var/log/netwark/netwark-worker-errors.log
```

Note: Please take care to replace the path in the configuration file by the good path used in your server.

You can now start the worker by using `systemctl start netwark-worker` and follow the logs by using `tail` or with `journalctl` and run a new operation through the web interface or via the *REST API*.

```
tail -f /var/log/netwark/netwark-worker.log /var/log/netwark/netwark-worker-errors.log

# or

journalctl -f netwark-worker
```

If the worker execute the task without error, you can enable the service to start automatically on boot by using:

```
systemctl enable netwark-worker
```

Voilà! You have done the deployment of the worker! We recommand now to follow the configuration page to adjust your installation settings.

2.3 Install on Docker

The deployment in Docker is much easier than the deployment on multiple machines but it can be painful if you want create a dedicated network across all your machines.

Note: We uses *alpine* images to have a minimum as possible footprint on the host machine and uses intermediate builds for not storing useless parts on our images (e.g. *node* and his *node_module* folder).

First above, you need to go into the `config` folder and create a copy of all files or `netwark_backend.yaml.example` and `production.ini.example`, and remove the `.example` in the extension.

Editing theses files are not needed for now, they already are configured for using the *PostgreSQL* and the *RabbitMQ* configured in the `docker-compose.yml` file.

Before building the base image, we need to create a `.mapbox-token` file at the root folder of Netwark that contain the **access token** of your Mapbox account.

Now, everything is ready to build the base image for the webserver and the worker. To do that, simply run the command:

```
docker-compose build
```

Now, we need to run some scripts on the *webserver* container to run *database migrations* and fill the *MAC OUI table*, retrieve *MaxMind DBs*. To do this, run theses three commands:

```
# Run database migrations
docker-compose run --rm webserver poetry run alembic -c config/production.ini upgrade␣
↪head

# Fill the `MAC OUI` table
docker-compose run --rm webserver poetry run python netwark/bin/update_oui_vendor_
↪table.py config/production.ini

# Update MaxMind DBs
docker-compose run --rm webserver poetry run python netwark/bin/update_maxmind_db.py␣
↪config/production.ini
```

Warning: If you want to mount the folder that contain maxmind databases, please execute theses commands before running the command for updating maxmind DBs:

```
mkdir maxmind_db
chown 1100:1101 maxmind_db  # 1100 and 1101 are GID/UID inside the container
```

At this point, the installation is finished! You can now execute `docker-compose up -d` to start the complete stack.

If you want Docker starts the whole stack on boot, replace `restart: on-failure` in the `docker-compose.yml` file by `restart: always`.

To test if everything works like a watch, watch the logs with `docker-compose -f` and open your browser and test if Netwark respond by accessing to http://localhost:6543.

Voilà! The complete stack is running on a single machine! You can now use *swarm*, *kubernetes* or create a tunnel on the host to communicate with other machines and add more workers to your Netwark installation. We recommand now to configure your **reverse proxy** and follow the configuration page to adjust your installation settings.

2.4 Configuration

This page contains all information you need to add additional settings to your installation.

2.4.1 <environment>.ini files

This configuration file contains all information the worker and the webserver need to work.

In general, two keys are important:

- *sqlalchemy.url*: contain the connection string to the PostgreSQL server
- *backend.config*: contain the path to the *netwark_backend.yaml* file

Note: SQLAlchemy give you the possibility to use several databases types. We do not recommend to use other database engine. If you still want to use for example MySQL, please make a pile of tests and send us a *Pull request* to integrate the database engine to the list of compatible database engines.

For the webserver, some additional keys are important to edit/keep update:

- *session.token*: this is the token for signing your user sessions. Please change his value before exposing the webserver to your network.
- *geoip_database.city*: contain the path to the MaxMind City database. The presence of the database is needed to start the webserver because is a key feature for retrieving additional informations when you WHOIS IP addresses and ASN.
- *geoip_database.ASN*: same as *geoip_database.city* but for the MaxMind ASN database.

If you are not satisfied with the output of the logs of the webserver or for the worker, you can edit the *logging* configuration by following the guide available on the *pyramid* documentation: https://docs.pylonsproject.org/projects/pyramid/en/latest/narr/logging.html

2.4.2 netwark_backend.yaml file

This configuration file is specific to asynchronous tasks. Both our components need them to send operations to queues (webserver) and listen queues for receiving and executing operations (worker).

Every keys in the object *celeryconfig* are configuration keys of Celery. The complete list are available on https://docs.celeryproject.org/en/latest/userguide/configuration.html.

This part is not recommended to modify except three keys:

- *broker_url*: contain the connection string to the RabbitMQ broker.
- *timezone*: update it to use your timezone/server timezone
- *broker_heartbeat*: specify the interval time between the worker need to send a *heartbeat* signal (we recommend to keep 60 seconds).

The *netwark_queues* are more important to configure. It concern all the queues of the worker can listen tasks. By default, every workers listen *netwark* queue and *netwark.broadcast* queue. *netwark* queue is a simple *direct* queue that RabbitMQ dispatch the task like *round-robin*. The *netwark.broadcast* queue is a *broadcastable* queue. All tasks sent to this queue are automatically played by **all workers** connected to the queue.

We have choosed to purpose a way to seperate where you want to execute your tasks by purposing the creation of custom queues. The best scenario is creating multiple queues for separating each offices/datacenters/circuits.

To do this, we have a template available on the example file.

```yaml
netwark_queues:
    - queue: lc_eqx_pa2
      name: 'Equinix PA2'
```

(continues on next page)

(continued from previous page)

```
      location: '114 Rue Ambroise Croizat, 93200 Saint-Denis, France'
      broadcast: true
```

- *queue*: is the name of the queue in Celery. To avoid errors, we only recommend to follow AMQP recommendations and uses `[a-zA-Z0-9-_.:]` characters.
- *name*: this is the name of the queue/location. This label will be soon visible on the frontend.
- *location*: this key give location informations to the frontend. Same as the name label, it will be soon visible on the frontend.
- *broadcast (not mandatory - default: False)*: specify if the queue need to be configured as a *broadcast* queue.

Warning: All queues you specify on your workers need to be specified in the *netwark_backend.yaml* file of the *webserver*. If you don't fill with all queues, you will do able not send operations to theses queues.

CHAPTER 3

REST API documentation

All functionnalities of Netwark are available in a REST API. To access it, simply access it by using the `http[s]://{instance_url}/api/v1/{resource}`.

> **Warning:** The REST API document are currently in progress. Status codes and the return content are currently not documented.

`GET /ip-calc/{resource}/{cidr}`

Parameters

- **`resource`** (*string*) –
- **`cidr`** (*string*) –

Status Codes

- **default** – UNDOCUMENTED RESPONSE

`GET /mac-oui/{resource}`

Parameters

- **`resource`** (*string*) –

Status Codes

- **default** – UNDOCUMENTED RESPONSE

`GET /mac-oui`

Status Codes

- **default** – UNDOCUMENTED RESPONSE

`GET /management/backend/queues`

Status Codes

- **default** – UNDOCUMENTED RESPONSE

`GET /operations/{operation_id}`

Parameters

- **`operation_id`** (*`string`*) –

Status Codes

- **default** – UNDOCUMENTED RESPONSE

`GET /operations`

Status Codes

- **default** – UNDOCUMENTED RESPONSE

`POST /operations`

Status Codes

- **default** – UNDOCUMENTED RESPONSE

Netwark is a web-based toolkit for lazy systems and network administrators that want to run parellized tools on multiple servers.

Netwark can help you to run ping or mtr on a single machine, all machines of the network or a group of machines. It embed some tools like IPv4/IPv6 calculator, MAC OUI Lookup and can WHOIS domains, ASN and ip addresses.

In the future, it's planned to add more tools and the capability to create smoke pings graphs and alerts.

CHAPTER 4

Features

Today, Netwark is capable to:

- Run asynchronous tasks (operations) with the possibility to broadcast (or not) the tasks to one or multiples machines/queues:
  - Can run `mtr` and retrieve a graph
  - Can run a simple `ping`
- **Run synchronous tasks for retrieving:**
  - WHOIS informations from *domains names*, *IP addresses* and *ASN numbers*
  - Retrieve informations about the manufacturer of a device by looking on the *MAC OUI* table
  - Calculate IPv4/IPv6 subnets with a *IP calculator*
- Access to all features with a **REST API** or through a sweet web interface

More *network/discovery* tools will be added to the list of available tools on asynchronous queues.

CHAPTER 5

Deployment informations

Netwark is completly written in Python using Pyramid Framework, Cornice, Celery and uses PostgreSQL has database.

Requirements:

- **Linux/Unix host** the server can works on Windows but the worker need some commands that only work on a true Linux/Unix environment (WSL don't allow to play with raw sockets).
- **PostgreSQL** 9.5+
- **RabbitMQ**
- **Node.JS** LTS (only for needed for npm and gulp)

You can also deploy Netwark on Docker and scale as you want.

For more informations, check the documentation.

CHAPTER 6

Contributions

Netwark is free and open source software licensed under **MIT** license.

This product includes GeoLite2 data created by MaxMind, available from https://www.maxmind.com

You can open issues to report a bug, suggest a new feature/enhancement or open a pull request to contribute to the codebase.

Please ensure you have black, pylint, pycodestyle and ESLint installed on your machine and ensure that no errors are returned by theses tools. Please create or adapt tests units for all your modifications.

HTTP Routing Table

/ip-calc

/mac-oui

/management

/operations