

# **NetGraphz2**

**Monitoring system interface**

# Introduction

NetGraphz2 — web-based interface for Icinga 2 monitoring system to provide visual graph-like overview of network, replacing static status map. It allow users interact with graph, see how nodes connected between. Such representation allows to find possible outages quickly, analyze weak points of network. Placing nodes in comfortable position using layout algorithm allows user to see clusters of nodes, which may represent connection points in real life.

Application goal is not to represent nodes on real geographical map. We believe that people remember connections much quicker than real world map. Evolution made a big gift for us humans giving us abstract associative memory abilities.

Program uses information from existing monitoring system (Icinga 2) using it's opened interface (MK LiveStatus). It stores graph separately from monitoring system configuration. It's not hardly connected to specific monitoring system.

Nodes and links are stored in graph database (neo4j) which allows application to do such actions as graph traversing really quick. It allows to easily automatically generate monitoring system configuration snippets starting from specific nodes using DFS.

Main web-interface is built on JavaScript. Rendering of graph performed by Cytoscape.js library which is leading library for graph draw and interaction. Additionally we use many other libraries under open license. Full list of used JavaScript libraries available at the document. At the same time structure of application own JS is pretty complex, but we don't use any sort of Bower, Require.js, AMD. We don't think that it's too much clever decision.

Structure of application is partial. There is no central daemon which handles all application life cycle. NetGraphz2 based on several components connected between using internal API. Messaging are passing between configured systems.

The main goal of our project is to provide people good, reliable, fast tool to find out problems in theirs difficult network. Visual interface is the primary priority of project. Without ability to see network from bird-eye view you can't make clever decision about future upgrade and development plan. When you have lack of maps, you don't know anything that might happen. We are here to solve this problem using well-tested, reliable technologies and solutions.

We want to TRY make your network management easy as possible!

*NetAssist LLC*

*2015*

# Structure

Application solution consists of several applications linked by API. As it was said above, there is no central daemon which controls application life cycle. So, we don't use such way.

At the opposite we have several parts and utilities with own role:

- Web-site (PHP 5.6, Phalcon) — Used to deliver pages, summary information, graph with correct nodes location to end users. We use it as back-end for end users. Site is written following MVC pattern, repository pattern and many others. We believe that good and documented code is what we need there. (folder: `web-phalcon`)
  - Client JS — core of visualization. Used to show graph, navigate through, search through and many more. It receives nodes from web-site, builds graph for use. It uses Cytoscape.js to perform node rendering and interaction. (folder: `web-phalcon/public/js`)
- Notifications server (Node.js) – Receives notification from monitoring system (Icinga) using it's RESTful API and monitoring notifications scripts. Gets node identifier, information and forwards notifications to users though WebSocket or AJAX LongPoll, also forwards chat messages from users. (folder: `notifications`)
- Graph API (Node.js) – Small RESTful API to manage graph nodes and links (folder: `api`)
- Icinga2 configuration generator (python) – Utility to generate configuration, traversing graph from root node using DFS. (folder: `icinga_config_generator`)
- Icinga2 notifications scripts (Node.js) – scripts to forward notifications from Icinga 2 to the notifications server
- Icinga2 include configurations example – sample configurations files to set up interactions between notifications server and monitoring system.

# Application

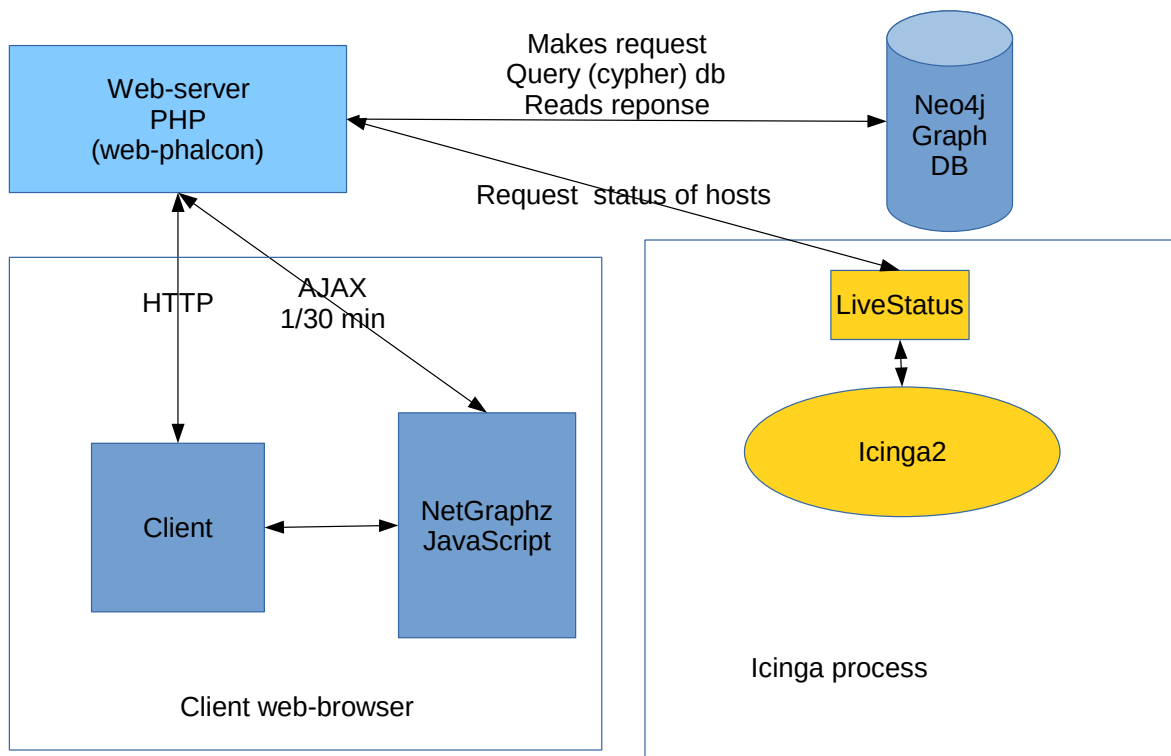
## Web front-end

Web interface front-end located in **web-phalcon** folder. It consists of several parts and uses MVC pattern. We use Phalcon 2.0.3 framework to build page, so it should be installed in order to install NetGraphz2.

In the root folder of web-front end you may find such folders:

- app — contains Phalcon applications source files (Controllers, View, Models) and code to access MKLiveStatus service.
- public — contains files to be sent to the user browser including JavaScript user interface

Let's take a look on front-end role. It is just simple, it gives user information about host from graph database and connects existing nodes to monitoring system data.



*Fig 1: Front-end site role*

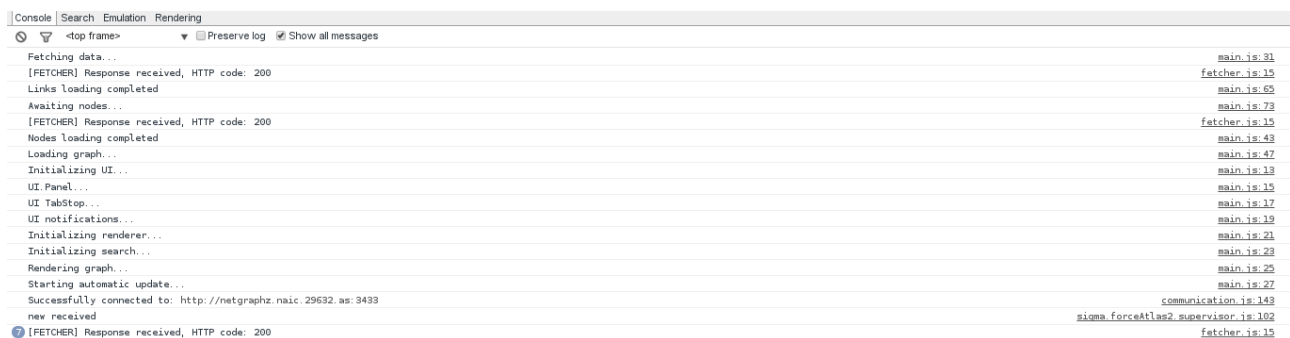
Front-end site configuration located in **web-phalcon/app/config/config.php** file. You can edit array contents of this file to change some settings like neo4j database URL, authentication parameters, MKLiveStatus connection parameters.

## Client JavaScript

One of the most important part is client JavaScript user interface. Uses Cytoscape.js, jQuery, jQuery UI, socket.io-client and few other 3-rd party libraries. All netgraphz JavaScript code located in `web-phalcon/public/js/netgraphz` folder It split into modules:

- `communication.js` – Server notifications handler
- `core.js` – Core definitions and constants
- `eventbus.js` – Event bus interface to implement publisher-subscriber pattern
- `fetcher.js` – Graph fetching functions
- `graph_utils.js` – Useful functions to work with graph states
- `main.js` – Provides application start-up sequence
- `renderer.js` – Interface for cytoscape.js renderer
- `settings.js` – **Contains settings for client scripts**
- `store.js` – In-memory storage for nodes, links and states
- `tools.js` – Basic tools like JS-object extend, deep extend, etc
- `ui.js` – User-interface functions and event-listeners
- `ui.notifications.js` – Notifications UI
- `ui.panel.js` – Node/link information panel UI functionality
- `ui.search.js` – Node search functionality (search field)
- `ui.tabstop.js` – TAB key problem navigation functionality
- `updater.js` – Automatic update of graph

Client JavaScript provides verbose error, information console output for browser. You can find out what's going wrong into browser developer console.



```
Console | Search | Emulation | Rendering
[icon] [icon] <top frame> [icon] Preserve log [x] Show all messages
Fetching data... main.js:31
[FETCHER] Response received, HTTP code: 200 fetcher.js:15
Links loading completed main.js:65
Awaiting nodes... main.js:73
[FETCHER] Response received, HTTP code: 200 fetcher.js:15
Nodes loading completed main.js:43
Loading graph... main.js:47
Initializing UI... main.js:13
UI Panel... main.js:15
UI TabStop... main.js:17
UI notifications... main.js:19
Initializing renderer... main.js:21
Initializing search... main.js:23
Rendering graph... main.js:25
Starting automatic update... main.js:27
Successfully connected to: http://netgraphz.naic.29632.as:3433 communication.js:143
new received sigma_forceAtlas2_supervisor.js:102
[FETCHER] Response received, HTTP code: 200 fetcher.js:15
```

Fig 2: JavaScript console output

## Notifications server

Forwards notifications between connected users, located in **notifications** folder. Works on Node.js IO framework which is designed for such use cases. Uses netgraphz2 node.js database library located in **node\_netgraphz2** folder

Handles web-socket or long-poll connections on port 3433 by default by sending JSON objects to connected clients about current events. Also listens 3434 port by default for Icinga script calls.

Configuration of the server located in **config.json** file.

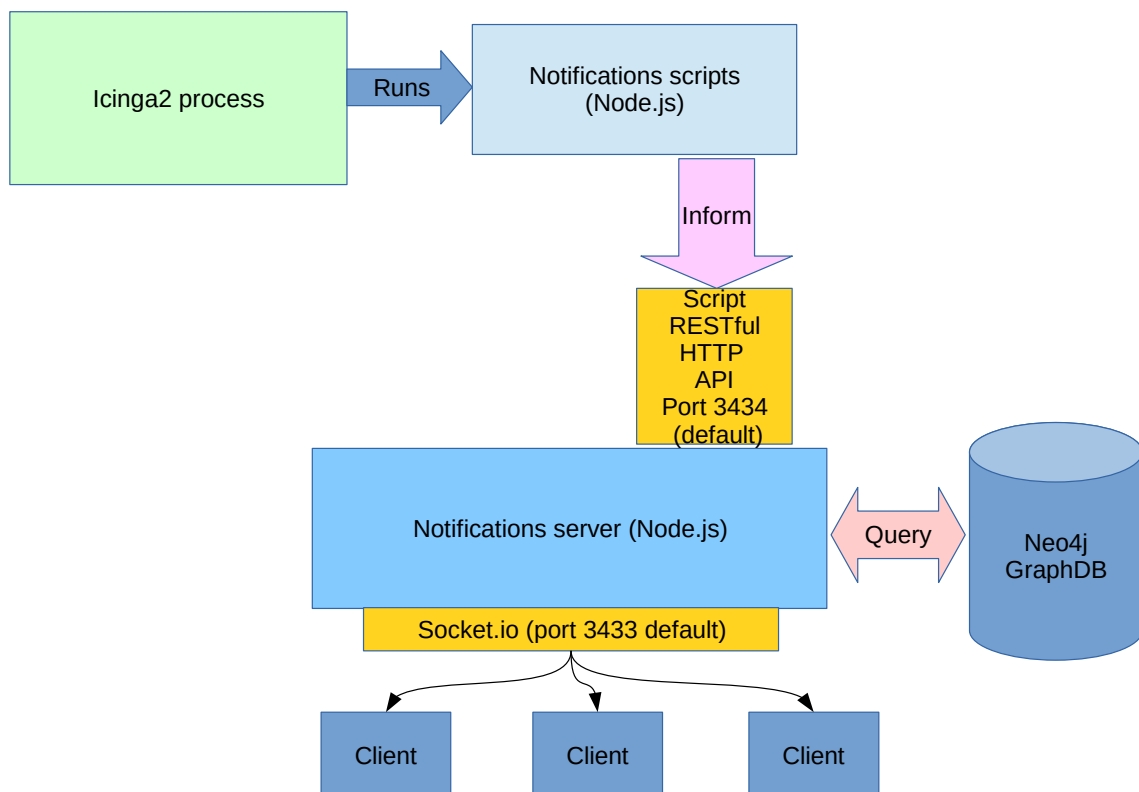


Fig 3: Notifications server flowchart

## Graph API

Still in development, will be published in **api** folder. Will use same JavaScript IO framework as run-time and same library as notifications server. Needed to update graph using RESTful API requests. Documentation about this application part will be published later...

## Tools

Application also contains few additional tools for deployment purposes.

### Icinga2 host configuration stubs generator

Can be used to create Icinga2 host configurations automatically starting from some root host using DFS. Located in `icinga/config_generator` directory, requires few additional Python modules to get started (look into installation section).

Configuration generation is long-time process: it might take 5 minutes to fully finish utility execution (depends from graph size).

Tool uses two files: `host_template.cfg`, `dependency_template.tpl` – host and host dependencies templates in order to generate configurations.

Tool configuration sample file can be found `config.json` file. Prints following help message when started with `--help` argument:

```
usage: generator [-h] [-v] config_file.json output_folder start_host
```

Icinga config generator from nodes graph database

positional arguments:

<code>config_file.json</code>	Configuration JSON file path (see README and manual)
<code>output_folder</code>	Output folder for config files
<code>start_host</code>	Starting host for DFS graph scanning

optional arguments:

<code>-h, --help</code>	show this help message and exit
<code>-v, --version</code>	show program's version number and exit

It generates only host stubs using `ip_address` and `icinga_name` graph nodes properties! So, be careful and always backup old Icinga2 host configuration!

### Icinga2 scripts and configurations

Located in `icinga` folder. Needed to connect Icinga2 hosts and services notifications mechanism to NetGraphz2 system. Configuration of scripts located in `icinga/scripts/config.json` file.

Additional required Icinga2 configurations provided in `icinga/conf.d` folder.

Script and configuration installation shown in installation section.

# Installation

System installation is easy to do, but requires few manual steps

1. Unpack distribution package into /opt/netgraphz2 directory on your server.
2. Install Icinga 2 (<https://www.icinga.org/download/>), neo4j (<http://neo4j.com/download/>), node.js >= 0.10.29 (<https://nodejs.org/download/>), Python 2.7.5 (<https://www.python.org/downloads/>), Icinga2 LiveStatus plugin service (included in distribution)
3. Install Python additional modules (needed to start toolkit). Run following command:  

```
# pip install MySQL-python py2neo
```
4. Install web-server capable of running PHP >= 5.6 (<http://php.net/downloads.php>) like Nginx (<http://nginx.org/en/download.html>) or Apache2 HTTPd (<http://httpd.apache.org/download.cgi>). There is one known issue with Apache2 httpd support: it might lead to worker process crash due and empty responses due to the strange memory leak when used in mod-php5 configuration. So we recommend to use FastCGI php configuration.
5. Install PHP 5.6 on top of the web-server to execute scripts, enable path rewriting in web-server configuration (e.g mod\_rewrite in Apache2).
6. Install PHP Phalcon module (<https://phalconphp.com/en/download>), follow instruction on Phalcon developers site. Don't forget to enable phalcon binary extension in PHP configuration and restart web-server. Phalcon PHP module should be loaded after all dependent modules (add extension=phalcon.so line into php configuration).
7. Install some PHP packages from PECL:  

```
# pecl install SPL_Types
```
8. Enable SPL\_Types module in PHP configuration (add extension=spl\_types.so into php configuration)
9. Create host definitions, allow path rewriting and set web-server root directory into /opt/netgraphz2/web-phalcon/public
10. Restart PHP FastCGI and web-server itself
11. Login into installed neo4j graph database instance and configure credentials (<http://localhost:7474>)
12. Enable Icinga2 MKLiveStatus extension (<https://github.com/Icinga/icinga2/blob/master/doc/15-livestatus.md>)
13. Place LiveStatus configuration file into /etc/icinga2/conf.d/livestatus.conf with following contents:



```

library "livestatus"
object LivestatusListener "livestatus-tcp" {
    socket_type = "tcp"
    bind_host = "127.0.0.1"
    bind_port = "6558"
}
object LivestatusListener "livestatus-unix" {
    socket_type = "unix"
    socket_path = "/var/run/icinga2/cmd/livestatus"
}

```

14. Edit LiveStatus configuration on your decision, remember configurations
15. Configure notifications server (/opt/netgraphz2/notifications/config.json).
16. Configure Icinga2 scripts (/opt/netgraphz2/icinga\_scripts/config.json). Default port of notifier API – 3434.
17. Configure web interface (/opt/netgraphz2/web-phalcon/app/config/config.php). No need to configure 'database' section
18. Configure client JavaScript (/opt/netgraphz2/web-phalcon/public/js/netgraphz/settings.js). Don't forget to change notification server URL!
19. Link Icinga2 configuration directories:
 

```
# cd <Icinga2 configuration path> - mostly /etc/icinga2
```

```
# ln -s /opt/netgraphz2/icinga/scripts/ scripts.netgraphz2
```

```
# ln -s /opt/netgraphz2/icinga/conf.d/ netgraphz2.d
```

 Edit icinga2.conf:  
 Add line:  

```
include_recursive "netgraphz2.d"
```
20. OPTIONAL: There is no API and other interfaces yet. Sorry. So use neo4j web-console (<http://localhost:7474>) to add nodes into graph (label: NetAssistNode) and links (type: LINKS\_TO). Link and nodes properties are described in the next section.
21. OPTIONAL: Generate Icinga2 configuration stubs using utility located in /netassist/netgraphz2/icinga/config\_generator directory
22. Add your hosts configuration into Icinga2.
23. Restart Icinga2
24. Start notifier daemon #[nodejs or node] /opt/netgraphz2/app.js . Add it into your autostart system.
25. Open web-browser and enter NetGraphz2 web-site address, press Enter

# neo4j Node and Relationship properties

## Hosts

Use nodes with label **NetAssistNode**. Following properties supported:

- name [string] — node name
- db\_sw\_id [int] — database identifier (if imported)
- address [string] — physical location
- comment [string]
- model [string] — Device hardware model
- **icinga\_name** [string] — links to the Icinga2 monitoring host name
- serial [string] — Serial number
- num\_ports [int] — Number of ports (for switches, routers)
- ip\_address [string] — IP address of Node (currently only IPv4)
- manage\_vlan\_id [int] — VLAN of management
- mac\_address [string] — MAC address
- loc\_id [int] — NetAssist internal
- net\_id [int] — NetAssist internal

## Links

Create relationship with type **LINKS\_TO**. Following properties supported:

- db\_sw\_id [int] - database identifier of source node
- db\_ref\_sw\_id [int] - database identifier of destination node
- speed [int] — link speed in mbps
- quality [int] — link quality for 0 to 1 [OPTIONAL]
- type [int] — link type (0-fiber, 1-copper, 2-radio, ...) [OPTIONAL]
- port\_id [int] — port number on source node
- ref\_port\_id [int] — port number on destination node
- comment [string]

# Usage

Open main page, you'll see loading graph. After loading layout start, ForceAtlas2 simulation will automatically make clusters. You can manually control layout execution pressing «Layout start» and «Layout stop» buttons. If user zoomed to much, it can restore graph position to the center using «Reset zoom» button.

Graph navigation is performed by mouse. Zooming is done by scroll wheel. Each node can be dragged in to preferred position. Single click selects node, double click focuses node in the center of the view-port.

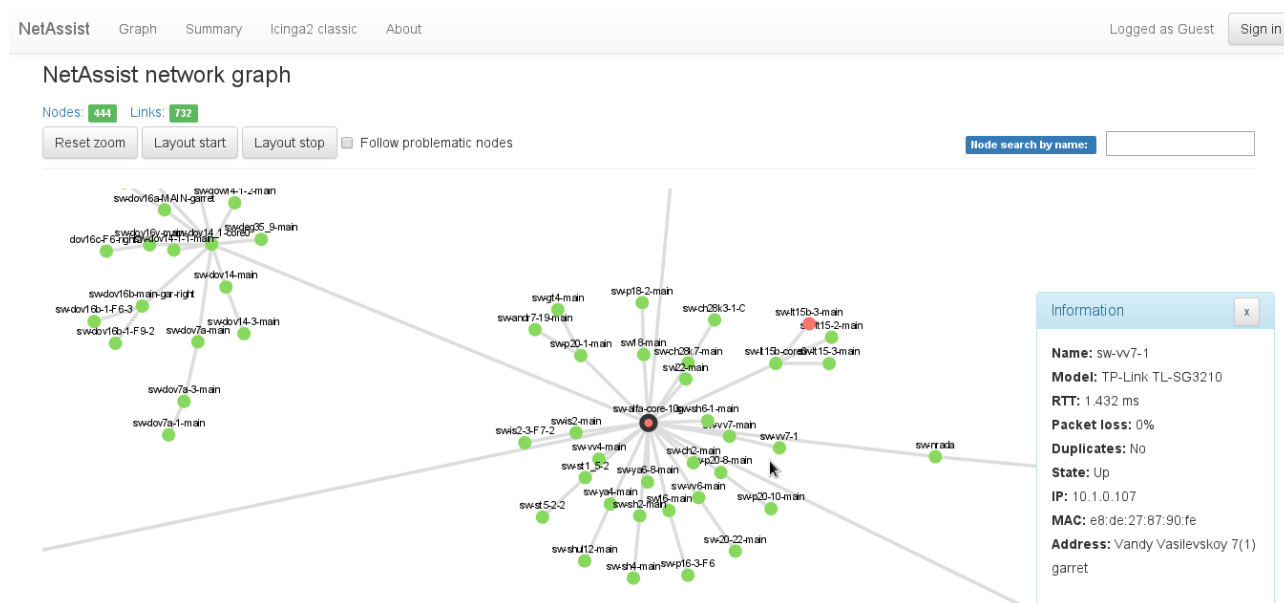


Fig 4: NetGraphz2 user interface

Nodes status are shown in different colors: red — down, green — up, warning — problematic, gray — not monitored by Icinga2.

Node information panel raises automatically when node being hovered by cursor.

Navigation through problematic and down nodes done by [TAB] and [SHIFT]+[TAB] keys. Search through nodes names done by search field (top right, [/] key). Check box on top of the screen enable automatic following to the down nodes in real time.

# Contacts

Company:

NetAssist LLC

Ukraine, Kyiv

str. Striletskaya 7/9 office 96

<http://netassist.ua>

Project leader:

Philippe Duke

NetAssist LLC

Email: [philippe46@netassist.ua](mailto:philippe46@netassist.ua)

[philippe46@pulsepad.com.ua](mailto:philippe46@pulsepad.com.ua)

[philippe46.snproject@gmail.com](mailto:philippe46.snproject@gmail.com)

[philippe46@henkuaile.cn](mailto:philippe46@henkuaile.cn)

Site: <http://henkuaile.cn>

Jabber: [philippe46@netassi.st](jabber:philippe46@netassi.st)

Facebook: <http://facebook.com/philippe46.snproject>