

Building APMv3 Map Visualization Using Nagios Host Data

Hans Sebastian Tiono^{1,a}, Jelle Oosterkamp^{2,b}, Tom Peperkamp^{3,c}

¹Department of Informatics, Petra Christian University, Surabaya, Indonesia

^{1,2}Information and Communication Technology, Fontys University of Applied Sciences, Eindhoven, The Netherlands

³Acknowledge Proactive Monitoring, Acknowledge Benelux BV, Waalre, The Netherlands

^ahanssebtino@gmail.com, ^bj.oosterkamp@fontys.nl, ^ctompeperkamp@acknowledge.nl

Abstract. Nowadays, having a network monitoring is becoming an important thing for a growing or big company which should also have a network infrastructure inside it. Network monitoring is the use of a system that constantly monitors a computer network for slow or failing components and that notifies the network administrator in case of outages [1]. Acknowledge Proactive Monitoring (APM), one of the sub-departments in Acknowledge, focuses on developing a network monitoring system using the Nagios tool. However, one of its main features called Map Visualization is not really efficient in terms of its map management. The current stable version of the system is called APMv2 and the team is developing a new version called APMv3 which needs a better Map Visualization. The application made in this project was a proof of concept that will later be used on APMv3. It is a web-based application which has a separate frontend written in HTML and JavaScript, has a separate RESTful backend written in PHP, and uses some frameworks such as jQuery UI, jCanvas, Twitter Typeahead, and Bootstrap 3. All necessary data within this application are exchanged asynchronously using AJAX.

Keywords: Map Visualization, APM, Nagios, APMv2, APMv3.

1. Introduction

This project was carried out at one of the companies in The Netherlands named Acknowledge Benelux BV.



Figure 1. Company logo

1.1 Acknowledge

Acknowledge is a company which provides business and IT solution to its clients. The company has currently 230 workers, who work in various departments. Acknowledge was founded in 1994 and was started as an IT supplier which provided box products, software licenses and small solutions. Throughout the years the company expanded its services towards its current portfolio and size.

1.2 Acknowledge Proactive Monitoring

Acknowledge Proactive Monitoring (APM) is one of the sub-department in Acknowledge which focuses on developing a system to monitor its client's computer systems, network, and infrastructure environment (usually in general they are called hosts). The client's hosts have status such as up/down/unreachable/pending, and they are visible in real time. So, whenever a host has a problem, the client will be alerted that some required actions should be performed to fix that problem. APM team does not use any local repository to store their works and code, but they use Git instead as their separate storage, so it supports the flexibility of their work.

The project was carried out in this sub-department. Tom Peperkamp is the head of APM as well as the project supervisor.

2. Project Overview

2.1 Background

Until now, APM team uses in-house developed application called APMv2 and it is the current stable version. The team is developing a new version of the application called APMv3.

2.2 APMv2 Application

APMv2 is functioning properly and has been broadly used to monitor clients' hosts.

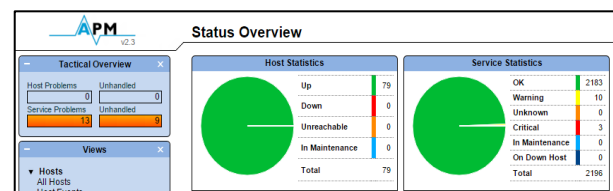


Figure 2. APMv2 status overview page

The application is web-based, written mostly in Python and partially in PHP. There are several terms in APMv2 such as host and service. Both of them are the main objects that are monitored by APMv2. A host in APMv2 basically refers to the computer, server or device which can be in physical or virtual form with a specific IP address. By contrast, a service is a process running on a host. A host can have more than one service or possibly has no services at all,

depending on the host type. There are some particular types of host which does not have any service.

Each host has a specific status (up / down / unreachable / pending). Figure 2 is APMv2 status overview page which has two pie charts indicating the hosts (left pie chart) and services (right pie chart) status overviews. Every color has a meaning: green is up, red is down, yellow is unreachable, and blue is pending. APM users can simply click on the status to see the hosts or services with the specified status.

One of the main functionalities in APMv2 is the Map Visualization. In this case, map refers to the network logical or physical diagram. Map visualization visualizes the client IT environments according to the APM standard. One of the reasons to use map visualization is to help the users better understand the connectivity between different hosts resulting in doing a faster impact analysis of the monitoring result. By visualizing the standardized map, users can notice easily how the IT infrastructure environment was constructed.

APMv2 application uses a tool called Nagios [2] for the whole monitoring functionality. For the Map Visualization functionality, APMv2 uses a tool called NagVis [3].

2.3 APMv3 Application

APMv3 has the same functions and still uses Nagios as its monitoring tool, but there will be some improvements and one of them is data storing and retrieval using an API.

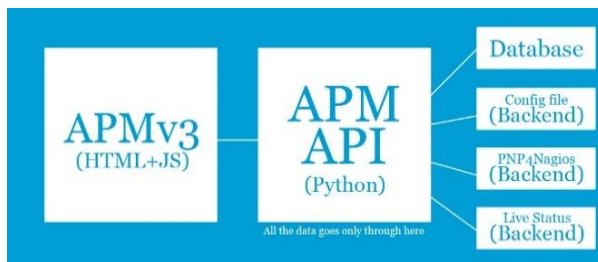


Figure 3. Expected architecture of APMv3

The main purposes of building APMv3 are to improve major functionalities, to make robust and scalable software, and to increase user experience through a well-built user interface.

APMv3 will use an API called APM API which is a door to some supportive backend data in the APMv3 application. The API also helps APM standardize the data flow (the data that goes out from database to user and vice versa), store some application protocols, and authenticate and/or authorize the user.

2.4 Current Problem

Due to a version change from APMv2 to APMv3, APM needs to rebuild the system all over again and it takes a lot of works. One of the major works needed to be done is the Map Visualization functionality in APMv3. Simple illustrations below on how APM users create a new map (Figure 4) and maintain it (Figure 5) will demonstrate why the way NagVis works in APMv2 is considered not efficient.

1. They have to make an image based file (.jpg/.png) network diagram in third party software like Microsoft Visio or any diagram builder.

2. They have to upload the image to the NagVis application.
3. They need to add a host/service symbol on each relevant host in the network diagram one by one. NagVis is just simply adding small icons identifying whether they are hosts and/or services. The icon will also indicate the status of the corresponding host or service.

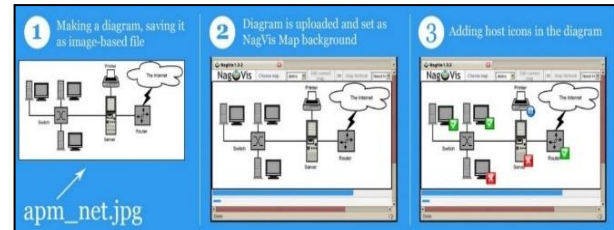


Figure 4. Creating a map in APMv2 Map Visualization

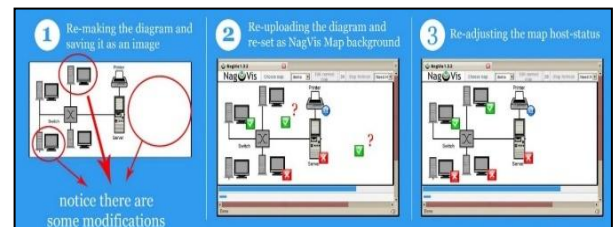


Figure 5. Modifying the created map

The map that has been created and added with some host icons will be active. NagVis will automatically relate the host icon to a specific real host status. So, the status on the map will follow the real status. However, the map will be difficult to maintain, in particular whenever a change occurs in the map (shown in Figure 5).

1. They have to edit the map in the third party diagram builder (e.g. Microsoft Visio) again.
2. They have to re-upload the image and set it as the background. In Figure 5, the host icons remain the same as the first version of the map because NagVis just saves their x and y positions.
3. The users have to re-adjust the host icons.

The map in Figure 5 is simple and it might be still durable. However, if the map had for instance 50 or 100 hosts, the re-adjustment would be a very exhausting work.

2.5 Project Description

The project was initialized with a purpose to improve the current Map Visualization on APMv2 to be applied later in the APMv3. The company hopes that APMv3 has a more efficient Map Visualization and could exchange every data via API. The project in overall is to research and build a new way in visualizing the monitoring results for APMv3 map visualization.

2.6 Functional Requirement

Here are the functional requirements of the program: add/edit/delete icons, add/edit/delete/view maps, add/edit/delete icon categories, and view activities.

2.7 Technical Requirement

APMv3 Map Visualizer is responsive because the map size follows the screen resolution. However, the application is designed in a computer environment which has the screen resolution of 1680×1050px, so it is highly recommended to use this application on the 1680×1050px screen resolution. The application has been tested on several browsers and it works well on Chrome, Mozilla Firefox, Opera, and Safari.

3. Project Initiation

There are 3 possibilities to accomplish the project. First, NagVis may still be used but it will be improved to become more like an object oriented visualization tool. Second, other completely different visualization tools or facilities are employed to replace NagVis. Third, a working visualization tool is built from scratch to be applied in APMv3.

3.1 Improving NagVis

After some research, it could be inferred that improving NagVis was possible, but it was nearly impossible to be accomplished by those who did not contribute in writing it or people outside the developer area because of the complexity of its code.

3.2 Using another Available Tool

There is another Map Visualization tool for monitoring purpose called Zabbix [4]. However, since the Map Visualization of Zabbix is Zabbix's proprietary, it was not possible to apply it for APMv3 which has a developed Nagios environment inside it.

3.3 Creating a Tool from Scratch

The only possibility would be to create a self-made map visualization feature for APMv3. The criteria to determine that this possibility is the best are the flexibility, reasonability, sustainability, and suitability to APMv3. The advantages of creating a tool from scratch are:

1. The tool is fully flexible – i.e., no need to follow some tool regulations or functions – and 100% unique so it has a sale or market value. The flexibility is measured through the programming language alternatives that the author could choose and the exact functionalities of the final product that the author himself could determine. Since the tool will be part of the whole APMv3 monitoring tool, the visualization module is considered an added value that users will find useful and can raise the application product's value.
2. The tool can be adjusted to be completely suitable to APMv3 and can exactly fulfill what APMv3 needs, and all kind of features can be easily added.
3. No need to study an available visualization tool and the way it works.
4. At the end, the tool creator can master his tool, which means he will understand the tool program as a whole.

The disadvantages of creating a self-made map visualization tool are:

1. Additional research has to be done regarding what frameworks should be employed and how to build the tool.
2. It will be a lot of works since the tool is built from zero.
3. There will be some risks that the tool does not work properly or there will be some bugs since a new freshly-built tool cannot be compared to a mature visualization tool, like NagVis, that has been through a lot of testings. A careful application development is needed.

3.4 Project Initiation Conclusion and Decision

Thus, it had been decided to create the map visualization from scratch. The advantages can outweigh the disadvantages because the final application will satisfy the most desirable purpose of APMv3 map visualization: to retrieve data only via API. The full flexibility of the application is one of its main advantages. Other alternatives may be nearly impossible to accomplish that and may require several backends besides the API.

4. Project Design

4.1 Entity Relationship Diagram (ERD)

ERD describes the data structure that is stored in the database. Figure 6 explains the ERD of APMv3 Map Visualizer which uses 7 tables: maps, maps_details, maps_relations, icons, categories, last_filenaming, and activities.

- Table maps stores the general information of a map and has a child table called maps_details which stores map host information inside a map.
- Table maps_details has a child table called maps_relations which stores optional line relations between two map hosts.
- Table icons stores the icon images' information used to represent a map host and they are categorized in table categories.
- Table last_filenaming is mainly used only to give incremental file names of the icons when a new one gets inserted by users.
- Table activities logs and tracks all user activities within the whole application.

5. Project Construction

5.1 Frameworks or Plugins Used

There are some frameworks used as plugins in the application. Those plugins are some JavaScript frameworks (jQuery, jQuery UI, jQuery Canvas [5], Twitter Typeahead, PNotify) and a CSS framework (Bootstrap 3). Bootstrap 3 [6] and jQuery are already used in APMv3.

5.2 Building Icon

The first thing to do is to build the icon management of the application. This includes add/edit/delete icons functionalities. The icon will be later used on a map as an image representation of a host. A single icon is designed to be 100×100 pixel size, but later, it can be resized to be bigger or smaller while it is being used on the map.

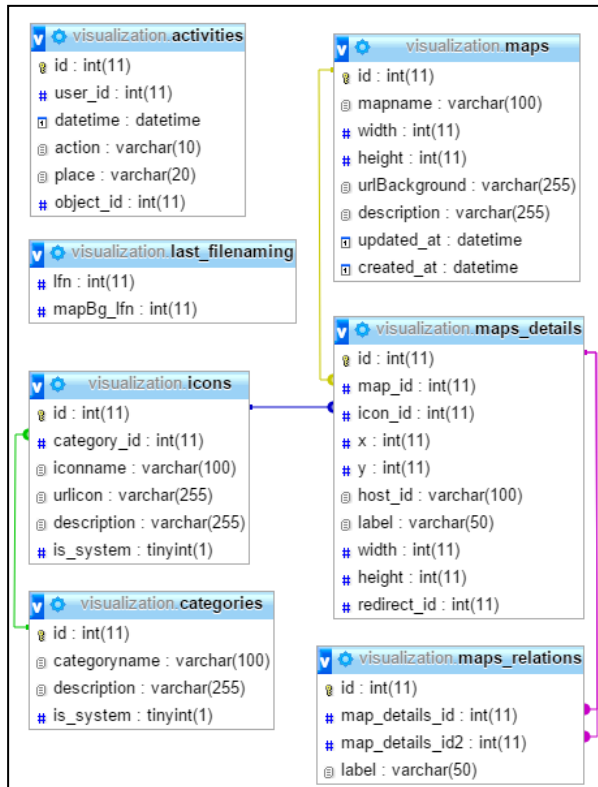


Figure 6. ERD of APMv3 Map Visualizer

5.2.1 Icon Image

Each map host must have exactly one icon image, so it is mandatory to link an icon with an image URL in the database. Figure 7 shows a clear example of how the host icon looks alike. Almost all the images are taken from Microsoft Visio [7].



Figure 7. Examples of APMv3 Map Visualizer icons

The preferable icon image file type is PNG because it supports transparency and has a better image color quality than GIF.

5.2.2 System Icon

A single icon could be either a system icon or just a normal icon. A system icon is an icon that can neither be created, edited, nor deleted by anyone except the system administrator. The purpose of the system icon is to represent the general standard icons in APMv3 Map Visualizer. The system icons have existed since the application was built. Any new icon that is created will be just a normal or custom icon, which can later be edited or deleted.

5.3 Building Category

Each icon is categorized in a category which means every icon must have exactly one category, as shown in Figure 8.





















Figure 8. Examples of APMv3 Map Visualizer categories

There is also a system category that has the same concept as the system icon. It is a category that cannot be created, edited or deleted except by the system administrator. When the application is installed the first time, there are some initial categories with some initial icons inside them. Table 1 shows all initial categories and icons inside it.

5.4 Building Map

Maps are what the application needs after the icons and categories have been created and filled with some records. A map is an empty place or field where the map host along with their relation(s) to other map host(s) can be placed and later it will become meaningful network information of a particular IT infrastructure.

Table 1. Initial Categories and Icons

Category	Icon name	Icon	System
Infrastructure (system category)	Iconless Host		Yes
	Computer		Yes
	Server		No
	Router		Yes
	Switch		Yes
	Firewall		Yes
	Storage		Yes
	Air Conditioner		Yes
	UPS		Yes
	Telephone		No
Platform (system category)	Windows Host		Yes
	Linux Host		Yes
	Database		Yes
Applications (system category)	Website		Yes
	Active Directory		Yes
Clients (system category)	Sensor		Yes
	Workplace		Yes
	Printer		Yes

The visualization of the maps is generated inside an HTML div element with an absolute positioned HTML canvas on top of it. Each canvas in the maps is maintained by jCanvas and all canvas drawing is done by it. The application uses only the line drawing method from jCanvas to draw the map host relations.

5.4.1 Map Size

Each map has a specific size depending on how big the map needs to be. The width and height attributes in table map will store the map's size in pixel (px) units. The minimum width and height of the map are set to be 100px and there is no restriction for the maximum map size. The default map size when it is first created is adjusted to the computer's resolution size.

5.4.2 Map Background

A map can have a background image or not. The background is set on the map by inserting it as a CSS background-image of the map HTML element without any repeat. There is no size restriction of the background and the background will remain at its original size. However, it is recommended to use an image smaller than 3MB, so the image upload would be fast. If the background is too small or too big comparing to the map size, the background will not be stretched on the map.

5.4.3 Map Host

A map can have zero or many map hosts. A map which does not have any host is an empty map or a new map that has just been created. All map host data are saved in a table called map_details.

5.4.4 Map Host Relation

A map host can have no relation or many relations to another map host. If there is a relation between two hosts in a map, then there will be a straight line drawn between those two hosts using jCanvas, a JavaScript plugin for HTML5 canvas. The relation data are stored in table maps_relations. Figure 9 clearly illustrates the whole components of a map.

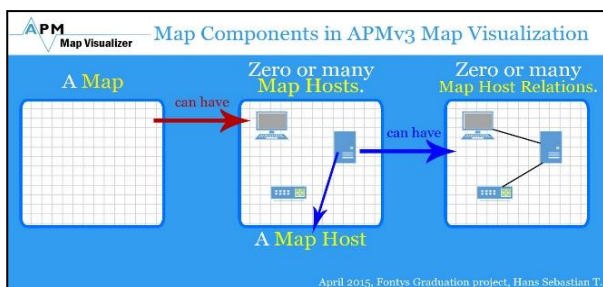


Figure 9. Map components

5.5 Building Asynchronous Backend Connection on APMv3 Map Visualization

Ideally, after the application has finished, all data connections between the application and the database have to go through the APM API. However, the APM API is still being built and it is not possible to be used as intended. At first, APMv3 Map Visualizer used HTML, JavaScript, and

PHP altogether, resulting in a very complex web application, and later it would be hard to integrate it with APM API because it does not use PHP. So, it had been decided to take all the PHP code to be a separated temporary API (backend) and the remaining code would be only HTML and JavaScript (frontend). The final application would be much easier to migrate to the finished real API as the main advantage of building the temporary API. Figure 10 shows three possibilities in building the application backend.

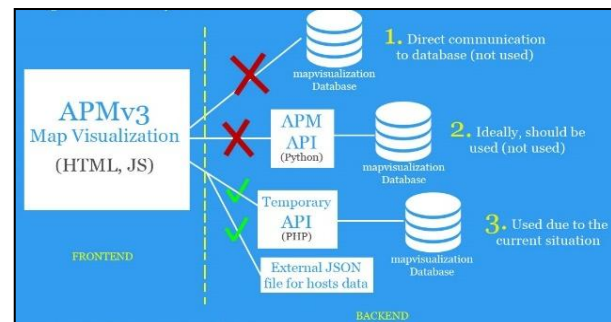


Figure 10. Possible backend communications

The temporary API will process all the server-side data. All of the application webpages are the frontends, which will display the client-side information as interactively as possible. The only data ready to be used from the APM API are the host data, so the application will use a JSON external file [8], which is derived from the APM API. After the APM API is ready to be used, the temporary API and the external JSON file will be migrated fully to the APM API.

The frontend communicates with the backend using AJAX (Asynchronous JavaScript and XML). AJAX is asynchronous data communication protocol. With Ajax, web applications can send data to and retrieve from a server asynchronously (in the background) without interfering with the display and behavior of the existing page [9].

5.5.1 RESTful Architecture Approach

The AJAX connections will be built according to the RESTful architecture, which means the connections use a set of HTTP methods: GET for selecting data, POST for inserting data, PUT for updating data, and DELETE for deleting data.

5.5.2 Application Endpoints

An endpoint is the URL needed to be specified within the AJAX call in order to do a correct intended action. To give illustration, the endpoint to select maps is different with the endpoint to select icons. The temporary API needs to use fifteen endpoints with different HTTP methods from the total of eight tables. Table 2 lists all the endpoints along with their explanation.

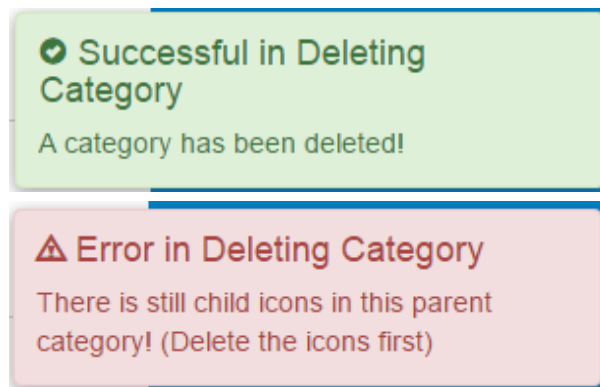
5.5.3 Data Structure in JSON

All AJAX connections will return a JSON formatted data, even if the connections are meant to insert, update or delete from a table. The select operation will obviously return the specified table contents in JSON but The JSON data returned from insert, update, or delete operations are different

Table 2. List of APMv3 Map Visualizer endpoints

Method	Endpoint	Usage
GET	<i>/visualization/activities/</i>	Get the list of all activities in JSON
GET	<i>/visualization/categories/</i>	Get the list of all categories in JSON
GET	<i>/visualization/icons/</i>	Get the list of all icons in JSON
GET	<i>/visualization/maps/</i>	Get the list of all maps (without map hosts and relations) in JSON
GET	<i>/visualization/maps/{id}</i>	Get a specific map along with its map hosts and relations in JSON
POST	<i>/visualization/categories/</i>	Insert a new custom category
POST	<i>/visualization/icons/</i>	Insert a new custom icon
POST	<i>/visualization/maps/</i>	Insert a new map
POST	<i>/visualization/images/</i>	Upload a new image for a map back-ground or an icon image depending on the value of the <code>_POST['imgtype']</code> variable
PUT	<i>/visualization/categories/</i>	Edit a specific category
PUT	<i>/visualization/icons/</i>	Edit a specific icon
PUT	<i>/visualization/maps/</i>	Edit a specific map along with its map hosts and relations
DELETE	<i>/visualization/categories/</i>	Delete a category
DELETE	<i>/visualization/icons/</i>	Delete an icon
DELETE	<i>/visualization/maps/</i>	Delete a map along with its map hosts and relations

The application frontend will receive all kinds of notification messages and will display them in the client-side interface using PNotify, as shown in Figure 11.

**Figure 11.** Examples of notifications displayed

5.5.4 Data Retrieval Method

The map data are retrieved on the frontend page by using both `jQuery.getJSON()` and `jQuery.ajax()` request methods. Both of them are AJAX request methods. In the APMv3 Map Visualizer, they are used together because they have their own advantages and disadvantages.

`jQuery.getJSON()` does not need many lines of codes. Another advantage of using `jQuery.getJSON()` is that you can actually store the result in a global JavaScript variable to be used later on. The `jQuery.ajax()` method returns a result which can only be used inside the function of the success handler due to the asynchronous nature of AJAX. So the result should be displayed immediately, and later if we need to refresh it, we have to recall the `jQuery.ajax()` request again, causing more page loadings and slowing down the application. That is one disadvantage of using the `jQuery.ajax()` request method.

However, there are at least two advantages of using the `jQuery.ajax()` method. First, unlike `jQuery.getJSON()` that can only use GET HTTP method, `jQuery.ajax()` can use all

four HTTP methods (GET, POST, PUT, and DELETE). So, `jQuery.ajax()` will support the RESTful architecture which is already being built in the APM API. It is possible to use GET HTTP method to serve POST, PUT, and DELETE purposes, but the application then will not have the RESTful architecture anymore. The second advantage of using the `jQuery.ajax()` request method is that the error can be returned in a very simple way using the error setting from the `jQuery.ajax()` itself which is written above, `error: function(the_error){}`.

Therefore in the APMv3 Map Visualizer, all insert, update, and delete operations use `jQuery.ajax()` to maintain the application to be RESTful, whereas the select operations usually use `jQuery.getJSON()` because the returned results are usually needed to be used again later.

5.6 Building Activities Tracker

The application has a feature to log every single action that is performed. The logging function is in the backend and the activity data are stored in table **activities**. The table contains what the activity was, where and when it was done. The activity tracker is displayed in a separate webpage and user can sort the activities according to their table header (ascending or descending) using `jQuery` table sorter. Figure 12 shows how the activity tracker looks like.

Activity	ID	Date
+ Add category	31	2015-04-22 15:43:47
+ Add category	31	2015-04-22 15:43:48
+ Edit category	31	2015-04-22 15:44:01
+ Delete category	31	2015-04-22 15:44:14
+ Add icon	31	2015-04-24 16:48:55
+ Delete icon	31	2015-04-24 16:49:03
+ Add category	30	2015-04-22 15:42:51

Figure 12. Activities Tracker display interface

6. Map Editor and Map Viewer

Map editor and map viewer are the two most important pages in the APMv3 Map Visualizer. As the names already imply, the map editor page is where a user can edit a map and the map viewer page is where a user can view a map.

6.1 Map Editor

The map editor has a page layout such as shown in Figure 13. This page uses Bootstrap 3 JavaScript (like popover for the map options, tooltip for icon explanation, collapse for icon selection in side bar, and modal for displaying map menu) to support its functionality and user-friendliness.



Figure 13. A preview of map editor page layout

One component in the map editor is the map host. When a map host is clicked, it will show a popover (shown in Figure 14) to adjust or change its host data.

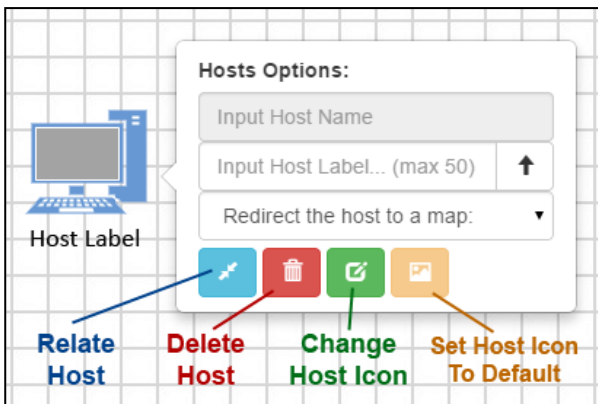


Figure 14. A popover detailing the map host data

There are several actions that a map host can do. Figure 15 displays some actions that can be applied to a map host.

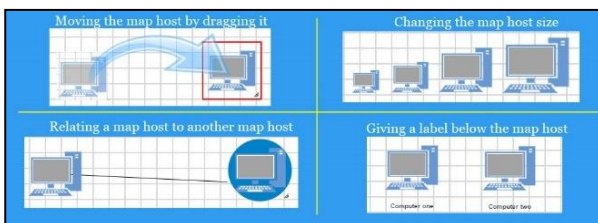


Figure 15. Actions that can be applied to a map host

6.2 Map Viewer

The map viewer shows the whole map along with its map hosts, map host relations, and status of every map host.

A map can be categorized as a logical map which does not have a background (shown in Figure 16) or a physical map which has an image background (shown in Figure 17).

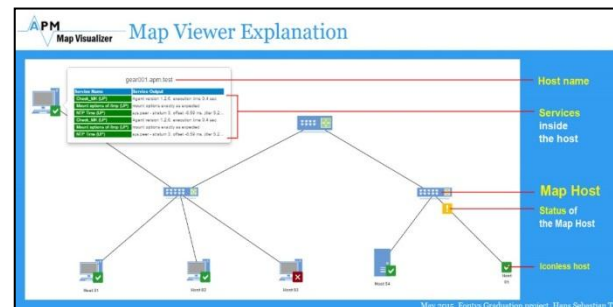


Figure 16. A preview of a logical map



Figure 17. A preview of a physical map

There are five possible host statuses (illustrated in Figure 18). The green host indicates “up” status, the red host “down” status, the yellow host “unreachable” status, the blue host “pending” status, and the black host “unknown” status. All statuses are retrieved from `hostresources.json`, which came from the APM API.

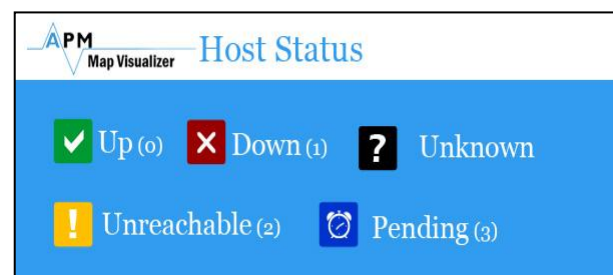


Figure 18. All host statuses

7. Conclusion and Recommendation

All mandatory requirements of the application have been fulfilled. An optional requirement, which is to provide authentication and authorization features within the final application, is not done because the process has to follow APMv3 user data (retrieved from APM API) and the API is still being built right now, so it is not possible to complete this requirement. Thus, the use case diagram is also discarded because it explains the authentication and authorization part.

APMv3 that is still being developed will later incorporate the APMv3 Map Visualizer for its map visualization feature. Through some research, the application was decided to be built from scratch. It is a web-based application, which has a separate frontend (written in HTML and JavaScript) and a separate RESTful backend (written in PHP). All necessary data within this application are exchanged asynchronously (in and out) using the AJAX method.

The APMv3 Map Visualizer can overcome the current map visualization's problem, which requires APM users to recreate the image-based map whenever a change occurs in the map, because the application can create the map itself directly on the webpage and it does not need any third party software like Visio to design the map. Compared to NagVis, which is used in APMv2, the final application has two major advantages: 1) it is fully flexible to be integrated with APM API (NagVis needs a huge customizations to integrate it with APM API) and 2) it uses a separate frontend and the API will be the only backend, which means there is no need to install any web service (NagVis needs to install PHP).

Further recommendation would be completing the application with authentication and authorization features after it has been integrated with the APM API and implementing security within the application.

References

1. Wikipedia, *Network Monitoring*. Available: https://en.wikipedia.org/wiki/Network_monitoring [Jun 2015].
2. Nagios Enterprise, *Nagios – The Industry Standard in IT Infrastructure Monitoring*. Available: <http://www.nagios.org> [Jun 2015].
3. NagVis Project Team, *NagVis 1.8 Documentation*. Available: http://docs.nagvis.org/1.8/en_US/index.html [Jun 2015].
4. Zabbix, *Zabbix – The Enterprise-Class Open Source Network Monitoring Solution*. Available: <https://www.zabbix.com> [Jun 2015].
5. Evans, C., *jCanvas – jQuery Meets the HTML5 Canvas*. Available: <http://calebevans.me/projects/jcanvas/> [Jun 2015].
6. Otto, M. and Thornton, J., *Bootstrap 3*. Available: <http://getbootstrap.com> [Jun 2015].
7. Microsoft, *Microsoft Office Visio Professional 2013*, 2013 [computer software].
8. ECMA International, *The JSON Data Interchange Format*, Standard ECMA-404 1st ed., 2013. Available: <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf> [Jun 2015].
9. Garrett, J.J., *Ajax: A New Approach to Web Applications*, 2005. Available: <http://adaptivepath.org/ideas/ajax-new-approach-web-applications/> [Jun 2015].