## Design

The system provides a simple cli interface that allows deploying networks (cloning stores), deploying stores (setup of vlan, static routes, vpn and l3fwrules) as well as updates of l3 firewall and s2svpn rules to store lists.

The system uses templates for vlans and l3 firewall rules, and after using in production Safeway proprietary Netx conversion generates actual absolute vlans and l3 firewalls rules.

The documentation of the conversion process and logic is contained in the relevant files. (netx.py, vlans.py and vlans\_handlers, and firewall\_handlers.py) and is outside the scope of this document.

Additionally, the system provides csv to json conversion and validation (schema and firewall semantics) for the required files to sanitize input data.

Another aspect is keeping extensive logs for traceability, auditability and future enhancements.

At this point the system does not support multi-user, but the basic design was done considering the need for this enhancement.

## Functionality

### Overview

A cli controller controls and dispatches commands to the lower layers.

The structure is such that handlers take care of manipulating the data and calling API drivers. These API drivers essentially interface directly to Meraki-API and provide logging and isolation.  
  
An automation layer contains handlers for the various modules. Automation layer is mostly concerned about generating proper API-Calls with proper inputs. It is also responsible for bulk and aggregated operations.

### CLI

This is a layer which receives cli commands, does some initial parsing and passes valid commands down to the automation layer.  
  
A text UI (tui.py) framework was developed and would need to be incorporated into the system if deemed required. This could benefit usability.

### Automation Layer

This layer composed of several handlers is mainly concerned with bulk-updates (multi-store) and template specialization using Safeway’s proprietary Netx Logic for the stores (l3 and vlans)

* Bulk update
* Firewall handler (l3rwrules)
* Network handler  
  Access to the network/store layer, mostly bulk support
* Static route handler
* Store Orchestration
* Vlan Handler
* Vpn firewall handler (s2svpnrules)
* Vpn handler

### API Layer

This layer contains the drivers to interface to external API’s and logic. It is mostly composed of Meraki API wrappers, but it also contains the Men and Mice interface and the Netx (Safeway’s proprietary logic for store subnet generation)

* firewall (layer 3)
* men-and-mice
* meraki
* meraki-patch

In order to improve error logging and error logging capabilities the open-source Meraki-API driver is hot-patched (python monkey patching). This allows for the automation scripts to be run in dry-mode, which means that scripts and input data can be exercised without incurring delays due to real Meraki-API calls and being totally transparent to the Meraki network.

* network
* netx
* static route
* vlans
* vpn

## Directory Structure

### Code Directories:

Below directories that contain code:

1. base directory

cli.py – cli handler

global\_var.py – caters for global flags used in debugging. (See debugging hints for more info)

tui.py (text ui handler framework, which was tested but no integrated into the system)

Contains also ancillary files:

“get, deploy, select, convert”: linux scripts to simply use of the cli for the get, deploy, select and convert functions.  
“VERSION.sh” – displays the git branch label for the current deployment

1. /automation sub-directory  
   contains all the handler files afore-mentioned.
2. /api sub-directory  
   contains all the API drivers afore-mentioned.

men\_and\_mice.py, network.py, vlans.py

devices.py, meraki.py, netx.py, vpn.py

firewall.py, meraki\_patch.py, static\_route.py, vpn\_firewall.py  
Please note that meraki.py is simply a copy of a Meraki release.  
(This is done so that Meraki updates do not impact the code, and also that there more control over the stability of the code, at any point a new version can be pulled in, but it is advisable that full regression testing should be done)

1. /utils sub-directory

contains general use utilities such as:

logging, csv, json, jinja support, global variables.

1. /GUI sub-directory  
   flask python web-server, which integrated for testing.   
   No longer being supported, however it could be useful in for REST-API support development.
2. /menAndMice sub-directory  
   Python 2.7 SOAP based men and mice client. Tested but not operational. Will need run as an external service periodically in order to provide full Men and Mice integration.

### Data Directories:

* /config sub-directory  
  Contains system level configuration.  
  At this point the main function for this settings file is to support new Orgs.  
  The process of setting up a new org is described in a separate document. (JASS>>>>>>)
* /data sub-directory

contains run time data and logs generated by the scripts.  
Useful for debugging and auditing.

* /runtime sub-directory

contains run time data and logs generated by the scripts.  
Useful for debugging and auditing.

* /templates sub-directory

Contains the in-use definitions for store-lists, org-lists, l3 firewall rules, s2svpn rules.

Please note the “valid\_inputs. json” file contains a list of valid schemas that is used in the convert utility (“csv-to-json”)

* /doc sub-directory

Used for storage of development relevant documents.

## DebugGING in Productions

### Increase the log messages verbosity.

When running in production log messages a kept to a minimum. In the case of errors where the messages are not sufficient it is possible to increase the verbosity by changing a variable in the global\_vars.py

Simply change the line

**log\_verbose = False**

to

**log\_verbose = True**

Do not forget to change it back once the problem has been resolved.

### Debugging the ISSUE on PYCHARM

Given that debug capabilities on the automation server are very limited, it might be necessary to resort to Pycharm in order to seek a faster resolution/debug to the problem.

The steps should be the following:

* An existing Pycharm environment should be already setup. It is outside the scope of this document how to setup such an environment.
* The version of code in nsmk-qa/code should be the same as the one which is being used in production.

These are the steps to obtain a sync-up version on your environment.

1. On the server run ./VERSION.sh

(same as doing a git describe –tags)  
This will provide which version is running on the server.

<VERSION\_IN\_PRODUCTION> e.g. P25

1. On your machine clone the same version:  
   git clone -b <VERSION\_IN\_PRODUCTION> [https://github.com/<repo](https://github.com/%3crepo)> code
2. Now you are ready to start debugging the code.
3. If the problem is a crash, simply put a breakpoint just before the crash and attempt to find the problem that way.

### Common PRODUCTION issues and possible remedies

It is always advisable to deploy l3 firewall and s2svpn rules against a test org.

1. Deploy l3fwrules fails  
   a ) The networks/stores have been corrupted.  
   Delete and re-deploy networks and stores again.   
   b) An actual problem with the l3fwrules\_template.json file.  
   Ensure the convert csv-to-json is passing without errors, watch for the actual error messages from the

Script. (The message will point to a line that can be offset but one of two depending if you are using csv or excel)

1. Deploy s2sfwrules fails

a) An actual problem with the l3fwrules\_template.json file.  
Ensure the convert csv-to-json is passing without errors, watch for the actual error messages from the

Script. (The message will point to a line that can be offset but one of two depending if you are using csv or excel)

1. Deploy