Dimension Data Contact Details

We welcome any enquiries regarding this document, its content, structure or scope. Please contact:

Linus Vidal - Sr. Consultant,

Dimension Data North America, Inc.  
5000 Hopyard Road,  
Suite 450,  
Pleasanton, CA 94588  
  
  
  
🕾 1 925 226 8378  
🖂 linus.vidal@dimensiondata.com

Please quote reference in any correspondence or order.

This document is valid until February 18, 2015 and is subject to the confidentiality and terms and conditions as per the formally agreed contract between Dimension Data North America, Inc., (“Dimension Data”) and Albertsons Group (“Albertsons”) or Dimension Data’s standard terms and conditions (which are available on request) if no formal contract has been agreed.

Document Configuration Management

Document Identification

|  |  |
| --- | --- |
| File Name | Albertsons-Meraki-Network-Automation-Design-v1.1.docx |
| Version | Version 1.0 |
| Sensitivity Classification | Company Confidential - Client / Vendor Information |
| Document Owner | Linus Vidal |

Preparation

|  |  |  |  |
| --- | --- | --- | --- |
| Action | **Name** | **Role / Function** | **Date** |
| Prepared by: | Linus Vidal |  | 2015-April-14 |
| Reviewed/Approved by: | Customer Team |  | 2015- |

Release

|  | Date Released | Change Notice | Remarks |
| --- | --- | --- | --- |
| 0.01 | 2015-April-21 |  | 1st Draft |

Contents

[1. Executive Summary 5](#_Toc511634040)

List of Figures

**No table of figures entries found.**

List of Tables

**No table of figures entries found.**

# Executive Summary

Dimension Data has been engaged to design the customer’s new LAN/WAN

# 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 in order 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

* /templates 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”)

* /data directory

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

* /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>>>>>>)
* /runtime sub-directory

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

* /doc sub-directory

Used for storage of development relevant documents.

# Subnetting and templates

The Meraki API for vlan creation and updates, uses a format, which requires absolute values which are specific to each store.

Additionally, the Meraki API for l3 firewall rule updates also requires absolute values.

One of the purposes of the automation scripts is to automatically generate the above files/formats based on templates and store name. This allows for bulk deployments and consistency across the network.

The subnetting logic is as follows:

Stores map to device names, which translates into an IP address.  
Based on this IP address the netx logic (explained below) creates eight subnets.

A men and mice (external utility) funnel file (explained below) contains the last octet and mask, together with the netx this provides all the information required for sub-netting. We will call this store-subnetting information

The template use is as follows:

The store subnetting information is used to specialize a VLAN jinja template, which produces the required vlans\_generated\_<xx>, used to call the Meraki API

So, with inputs netx + funnel and using /config/jinja\_vlans\_template we obtain vlans\_generated which is used by the Meraki API to update and or create the vlans.

For l3 firewall rules the Meraki API call process follows the same logic as explained previously but uses an l3fwrules\_template\_<xxx> provided and constantly updated by the Safeway Security Team. This template follows the Meraki standard for firewall templating standard, but we do not use it to make Meraki API calls directly. This choice of template format was done for the sake of consistency. Additionally, this template is specialized by actual automation scripts code as jinja was not suitable for this task.  
The firewall\_handler.py contains the code that specializes this template.

# NETX/IP SUBNET Generation WORKFLOW

Store names will always be in the format <ABC>\_<nnnn>, where ABC is the Safeway Division (e.g. SHA, JEW, etc.) and <nnnn> is the actual store four-digit number.  
  
Store numbers are unique, and each store has a network controller cc<nnnn> associated with it. For example, the SHA\_0012 store will have a cc0012 controller which should be visible on the Safeway network.

The VLAN generation follows the following steps:

* From the Men and Mice external utility a vlans\_funnel.csv file is obtained. This file is patched using the config/vlan\_funnels.patch.csv.

The first colum is the VLAN number, the second line which follows the format

10.x.a.nn/nn is the subnetting.

10.x.a is translated into a select for the subnet “a” (three octets subnet created by the netx function)

Conversely 10.x.b. selects subnet “b”, and so forth up to 10.x.h which selects subnet “h”.

The last octet is exactly that the last octet (4th octet, plus mask)

## vlans\_netx.json subnet file generation.

* Find the ip for the store  
  ping cc8501

Pinging cc8501 [10.218.31.5] with 32 bytes of data:

* “upper” first and second octets
  + copy from the ip   
    “upper” = “10.218.xx”
  + third octet is ip’s third octet – 3; 31-3 = 28

“upper” = “10.218.28.0”

* “lower” first and second octets

  For “lower” the second octet is subtracted by 64 (218-64=154)

* + copy from the ip   
    “lower” : “10.154.xx”
  + third octet is ip’s third octet – 3; 31-3 = 28

” lower” : “10.154.28.0”

* “a” , “b”, “c” and “d” are three octets based on “upper”

“a” is a copy of “upper” first three octets

“b” has the last octet same as “a” incremented by one

“c” has the last octet same as “b” incremented by one

“d” has the last octet same as “c” incremented by one

So:

“a” : “10.218.28”

“b” : “10.218.29”

“d” : “10.218.30”

“d” : “10.218.31”

* “e”, “f”, “g” and “h” are three octets based on “lower”

“e” is a copy of “lower” first three octets

“f” has the last octet same as “e” incremented by one

“g” has the last octet same as “f” incremented by one

“h” has the last octet same as “g” incremented by one

So:

“e” : “10.154.28”

“f” : “10.154.29”

“g” : “10.154.30”

“h” : “10.154.31”

Sample vlans\_netx.json file

{  
 *"upper"*: "10.218.28.0",  
 *"lower"*: "10.154.28.0",  
 *"a"*: "10.218.28",  
 *"b"*: "10.218.29",  
 *"c"*: "10.218.30",  
 *"d"*: "10.218.31",  
 *"e"*: "10.154.28",  
 *"f"*: "10.154.29",  
 *"g"*: "10.154.30",  
 *"h"*: "10.154.31"  
}

## vlans\_funnel.csv file.

From Men and Mice we get funnel\_vlans.csv

Vlan,Subnet,Description  
1,10.x.a.16/27,Network Management  
4,10.x.a.32/27,Network Management  
6,10.x.a.64/27,Extranet Vendors  
7,10.x.a.96/27,Pharmacy  
8,10.x.a.128/27,Backstage Users  
14,10.x.a.248/29,Store Content Engine  
16,10.x.b.0/24,Store Wireless  
19,10.x.c.0/24,General Store LAN  
24,10.x.d.0/24,POS LAN  
35,10.x.e.0/27,Digital Signage  
40,10.x.e.32/27,Thin Wireless Mgmt  
45,10.x.h.192/26,Cisco Wireless Management  
60,10.x.h.128/27,Vendor VLAN II (2nd Subnet)  
70,10.x.e.64/26,New Pharmacy  
75,10.x.e.128/25,Macauthenticated Wireless Clients  
79,10.x.g.192/26,VOIP Clients  
80,10.x.f.0/24,Secure Wireless  
81,10.x.g.160/28,Field Service Wireless  
82,10.x.g.176/28,iPad quarantine vlan  
85,10.x.h.0/26,Retail Cluster Management VLAN  
95,10.x.g.128/27,Printer VLAN

Vlans funnel\_funnel is patched with the patch below:

Vlans\_funnel\_patch has vlans where subnets do not changer per store

992,"192.168.192.0/24","Digital Signage - I"  
995,"192.168.1.0/24","Guest WIFI"  
996,"192.168.100.0/24","Cache VPN"  
997,"192.168.101.0/24","Cache Internet"

## 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 converted csv-to-json is passing without errors, watch for the actual error messages from the convert script. (A message will point you to a line that can be offset but one of two depending if you are using csv or excel source input)
2. 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)

The vlans\_funnel.csv (copied from ./menAndMice/funnel.csv and patched with ./config/vlans\_funnel\_patch.csv is an old-style template.   
The first column is the vlan number, the second column contains the 3 octect subnet reference and the actual last octect + mask.   
(e.g. 10.x.a.16/27 , translates to use netx subnet “a” with the last octect as 16 and mask 27)

The netx logic explained below produces 8 subnets (named “a”, “b”… “g”,”h”) for a given absolute fixed ip.

So netx logic and funnel-vlan logic above provide the subnet which is used in VLAN generation and Firewall Rule generation.

Vlan,Subnet,Description  
1,10.x.a.16/27,Network Management  
4,10.x.a.32/27,Network Management  
6,10.x.a.64/27,Extranet Vendors  
7,10.x.a.96/27,Pharmacy  
8,10.x.a.128/27,Backstage Users  
14,10.x.a.248/29,Store Content Engine  
16,10.x.b.0/24,Store Wireless  
19,10.x.c.0/24,General Store LAN  
24,10.x.d.0/24,POS LAN  
35,10.x.e.0/27,Digital Signage  
40,10.x.e.32/27,Thin Wireless Mgmt  
45,10.x.h.192/26,Cisco Wireless Management  
60,10.x.h.128/27,Vendor VLAN II (2nd Subnet)  
70,10.x.e.,New Pharmacy64/26  
75,10.x.e.128/25,Macauthenticated Wireless Clients  
79,10.x.g.192/26,VOIP Clients  
80,10.x.f.0/24,Secure Wireless  
81,10.x.g.160/28,Field Service Wireless  
82,10.x.g.176/28,iPad quarantine vlan  
85,10.x.h.0/26,Retail Cluster Management VLAN  
95,10.x.g.128/27,Printer VLAN  
96,10.x.g.178/27,Printer VLAN  
992,"192.168.192.0/24","Digital Signage - I"  
995,"192.168.1.0/24","Guest WIFI"  
996,"192.168.100.0/24","Cache VPN"  
997,"192.168.101.0/24","Cache Internet"

x.e.64/26,New Pharmacy  
75,10.x.e.128/25,Macauthenticated Wireless Clients  
79,10.x.g.192/26,VOIP Clients  
80,10.x.f.0/24,Secure Wireless  
81,10.x.g.160/28,Field Service Wireless  
82,10.x.g.176/28,iPad quarantine vlan  
85,10.x.h.0/26,Retail Cluster Management VLAN  
95,10.x.g.128/27,Printer VLAN  
992,"192.168.192.0/24","Digital Signage - I"  
995,"192.168.1.0/24","Guest WIFI"  
996,"192.168.100.0/24","Cache VPN"  
997,"192.168.101.0/24","Cache Internet"

The vlan\_netx.json describes 8 subnets for a give

It is easier to understand the logic by example:

How to generate netx by example  
  
a) ping cc8501  
b) Pinging cc8501 [10.218.31.5] with 32 bytes of data:  
c) netx["upper"] = {1:10, 2:218, 3:28, 4:0} (31-3 = 28 for 3rd octet)  
d) netx["lower"] = {1:10, 2:154, 3:28, 4:0} (218-0x40=154 for 2nd octet)  
  
e) next{"a"] = {1:10. 2:218. 3:28} (copy 1,2,3 octets from netx["upper"]  
f) next{"b"] = {1:10. 2:218. 3:29} (copy 1,2 octets from netx["a"] and 3rd octet = netx["a"][3] + 1  
g) next{"c"] = {1:10. 2:218. 3:30} (copy 1,2 octets from netx["b"] and 3rd octet = netx["b"][3] + 1  
e) next{"d"] = {1:10. 2:218. 3:31} (copy 1,2 octets from netx["c"] and 3rd octet = netx["c"][3] + 1  
  
e) next{"e"] = {1:10. 2:218. 3:28} (copy 1,2,3 octets from netx["lower"]  
f) next{"f"] = {1:10. 2:218. 3:29} (copy 1,2 octets from netx["a"] and 3rd octet = netx["f"][3] + 1  
g) next{"g"] = {1:10. 2:218. 3:30} (copy 1,2 octets from netx["b"] and 3rd octet = netx["g"][3] + 1  
e) next{"h"] = {1:10. 2:218. 3:31} (copy 1,2 octets from netx["c"] and 3rd octet = netx["b"][3] + 1  
   
So for  
ip = 10.218.31.5  
netx = {  
 "upper": "10.218.28.0",  
 "lower": "10.154.28.0",  
 "a": "10.218.28",  
 "b": "10.218.29",  
 "c": "10.218.30",  
 "d": "10.218.31",   
 "e": "10.154.28",  
 "f": "10.154.29",  
 "g": "10.154.30",  
 "h": "10.154.31"  
}

Below the generated vlans\_netx.json

{  
 *"upper"*: "10.195.200.0",  
 *"lower"*: "10.131.200.0",  
 *"a"*: "10.195.200",  
 *"b"*: "10.195.201",  
 *"c"*: "10.195.202",  
 *"d"*: "10.195.203",  
 *"e"*: "10.131.200",  
 *"f"*: "10.131.201",  
 *"g"*: "10.131.202",  
 *"h"*: "10.131.203"  
}

The cleaned up vlan\_funnel template is called vlan\_funnel\_netx and is shown below:

Vlan,Subnet,Description  
1,a.16/27,Network Management  
4,a.32/27,Network Management  
6,a.64/27,Extranet Vendors  
7,a.96/27,Pharmacy  
8,a.128/27,Backstage Users  
14,a.248/29,Store Content Engine  
16,b.0/24,Store Wireless  
19,c.0/24,General Store LAN  
24,d.0/24,POS LAN  
35,e.0/27,Digital Signage  
40,e.32/27,Thin Wireless Mgmt  
45,h.192/26,Cisco Wireless Management  
60,h.128/27,Vendor VLAN II (2nd Subnet)  
70,e.64/26,New Pharmacy  
75,e.128/25,Macauthenticated Wireless Clients  
79,g.192/26,VOIP Clients  
80,f.0/24,Secure Wireless  
81,g.160/28,Field Service Wireless  
82,g.176/28,iPad quarantine vlan  
85,h.0/26,Retail Cluster Management VLAN  
95,g.128/27,Printer VLAN  
992,192.168.192.0/24,Digital Signage - I  
995,192.168.1.0/24,Guest WIFI  
996,192.168.100.0/24,Cache VPN

Vlan\_funnels\_subnet is the specialization of vlan\_funnel\_netx using the vlans\_netx.csv

Vlan,Subnet,Description  
1,10.195.200.16/27,Network Management  
4,10.195.200.32/27,Network Management  
6,10.195.200.64/27,Extranet Vendors  
7,10.195.200.96/27,Pharmacy  
8,10.195.200.128/27,Backstage Users  
14,10.195.200.248/29,Store Content Engine  
16,10.195.201.0/24,Store Wireless  
19,10.195.202.0/24,General Store LAN  
24,10.195.203.0/24,POS LAN  
35,10.131.200.0/27,Digital Signage  
40,10.131.200.32/27,Thin Wireless Mgmt  
45,10.131.203.192/26,Cisco Wireless Management  
60,10.131.203.128/27,Vendor VLAN II (2nd Subnet)  
70,10.131.200.64/26,New Pharmacy  
75,10.131.200.128/25,Macauthenticated Wireless Clients  
79,10.131.202.192/26,VOIP Clients  
80,10.131.201.0/24,Secure Wireless  
81,10.131.202.160/28,Field Service Wireless  
82,10.131.202.176/28,iPad quarantine vlan  
85,10.131.203.0/26,Retail Cluster Management VLAN  
95,10.131.202.128/27,Printer VLAN  
992,192.168.192.0/24,Digital Signage - I  
995,192.168.1.0/24,Guest WIFI  
996,192.168.100.0/24,Cache VPN  
997,192.168.101.0/24,Cache Internet

Meraki Specialization/Accepted format for VLANS.

Jinja is used for the conversion:  
The Jinja template is in /config/jinja\_vlans\_template.json  
The input data is generated by converting

Vlans\_funnel\_subnet (which is a list) into a table

{  
 *"1"*: "10.195.200.16/27",  
 *"4"*: "10.195.200.32/27",  
 *"6"*: "10.195.200.64/27",  
 *"7"*: "10.195.200.96/27",  
 *"8"*: "10.195.200.128/27",  
 *"14"*: "10.195.200.248/29",  
 *"16"*: "10.195.201.0/24",  
 *"19"*: "10.195.202.0/24",  
 *"24"*: "10.195.203.0/24",  
 *"35"*: "10.131.200.0/27",  
 *"40"*: "10.131.200.32/27",  
 *"45"*: "10.131.203.192/26",  
 *"60"*: "10.131.203.128/27",  
 *"70"*: "10.131.200.64/26",  
 *"75"*: "10.131.200.128/25",  
 *"79"*: "10.131.202.192/26",  
 *"80"*: "10.131.201.0/24",  
 *"81"*: "10.131.202.160/28",  
 *"82"*: "10.131.202.176/28",  
 *"85"*: "10.131.203.0/26",  
 *"95"*: "10.131.202.128/27",  
 *"992"*: "192.168.192.0/24",  
 *"995"*: "192.168.1.0/24",  
 *"996"*: "192.168.100.0/24",  
 *"997"*: "192.168.101.0/24"  
}

vlans\_funnel\_table (note that the description info is being dropped and the description is being picked up from the jinja template file), this maps vlans to respective subnets.

Also note that the jinja template uses the 3 octets for the appliance-ip and this is generated by the code on the flight, so there is no file mapping for this.

The network ID is also picked up on the flight and uses the correct Meraki-netid for that particular store.

So now we have to generate a proper specialized version of the vlans which can be applied to Meraki.  
The template for Meraki Vlans is in /config/jinja\_vlans\_template.json

{  
 *"id"*: 19,  
 *"networkId"*: "{{networkid}}",  
 *"name"*: "generalstorelan",  
 *"applianceIp"*: "{{vlan[19]['octets']}}.1",  
 *"subnet"*: "{{vlan[19]['subnet']}}",  
 *"dnsNameservers"*: "upstream\_dns",  
 *"fixedIpAssignments"*: {},  
 *"reservedIpRanges"*: []  
},  
{  
 *"id"*: 24,  
 *"networkId"*: "{{networkid}}",  
 *"name"*: "storelan",  
 *"applianceIp"*: "{{vlan[24]['octets']}}.1",  
 *"subnet"*: "{{vlan[24]['subnet']}}",  
 *"dnsNameservers"*: "upstream\_dns",  
 *"fixedIpAssignments"*: {},  
 *"reservedIpRanges"*: []  
},

The final output is vlans\_generated\_N\_686798943174007640.json and is used against the Meraki API

[  
 {  
 *"id"*: 4,  
 *"networkId"*: "N\_686798943174007640",  
 *"name"*: "networkmgmt",  
 *"applianceIp"*: "10.195.200.33",  
 *"subnet"*: "10.195.200.32/27",  
 *"dnsNameservers"*: "upstream\_dns",  
 *"fixedIpAssignments"*: {},  
 *"reservedIpRanges"*: []  
 },  
 {  
 *"id"*: 6,  
 *"networkId"*: "N\_686798943174007640",  
 *"name"*: "extranetvendor",  
 *"applianceIp"*: "10.195.200.65",  
 *"subnet"*: "10.195.200.64/27",  
 *"dnsNameservers"*: "upstream\_dns",  
 *"fixedIpAssignments"*: {},  
 *"reservedIpRanges"*: []  
 },  
 {  
 *"id"*: 7,  
 *"networkId"*: "N\_686798943174007640",  
 *"name"*: "pharmacy",  
 *"applianceIp"*: "10.195.200.97",  
 *"subnet"*: "10.195.200.96/27",  
 *"dnsNameservers"*: "upstream\_dns",  
 *"fixedIpAssignments"*: {},  
 *"reservedIpRanges"*: []  
 },