Självständigt arbete på grundnivå

Independent degree project - first cycle

Datateknik
Computer Science

Network Automation – the power of Ansible

Markus Borgenstrand
Abstract

This report discusses network automation primarily with Ansible. Ansible is a software from Red Hat that can be used for network automation. The report also goes through YAML which is a standardized way of exchanging data, Jinja2 that is a templating language, Python as well as the security with Ansible. The report also goes through why network automation is needed as well as how much time might be saved with Ansible. Ansible ships with modules for Cisco IOS such as ios_config and ios_command and for Cisco ASA asa_config, asa_command and asa_acl as well as many other modules for Arista, Juniper and for other vendors. Ansible can use new APIs by creating new modules for handling that particular API, which means that the only change needed in the playbooks is to change the module name. Ansible can handle NETCONF API using the netconf_config module or various Juniper modules. Ansible is used in this report to perform certain tasks such as adding VLAN’s, close ports on ASA’s, audit network devices configuration as well as to create network diagram using the information from CDP. Ansible can be made as secure as manually doing the tasks except that Ansible can do it faster and more consistently. For connecting to normal Linux servers Ansible uses OpenSSH which is a default SSH client on most Linux systems and for connecting to network devices it uses Paramiko. The security in Ansible depends on SSH and may or may not have passwords stored locally. Ansible can be as secure as the administrator wants it to be such as using RSA key-pair to authenticate, using vault encrypted credentials or asking the administrator about which username and password to use. Using Ansible network automation can save time, the amount saved depends on what is being done, how many devices it is doing it on as well as how the playbook is written.

Keywords: Ansible, Network automation, YAML, Jinja2, Python, SSH, API
Sammanfattning


Nyckelord: Ansible, Nätverksautomation, YAML, Jinja2, Python, SSH, API
# Appendix

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1 Ansible files</td>
<td>58</td>
</tr>
<tr>
<td>A.2 Ansible VLAN</td>
<td>58</td>
</tr>
<tr>
<td>A.3 Ansible ASA</td>
<td>63</td>
</tr>
<tr>
<td>A.4 Ansible Active</td>
<td>67</td>
</tr>
<tr>
<td>A.5 Ansible audit</td>
<td>69</td>
</tr>
<tr>
<td>A.6 Ansible with Python Layer2 diagram</td>
<td>86</td>
</tr>
<tr>
<td>A.7 Ansible with Python Layer2 diagram with STP</td>
<td>90</td>
</tr>
<tr>
<td>A.8 Saving configuration</td>
<td>98</td>
</tr>
<tr>
<td>A.9 Ansible time</td>
<td>99</td>
</tr>
<tr>
<td>A.10 Python graphs</td>
<td>114</td>
</tr>
</tbody>
</table>
1 Introduction

This project will investigate why we have a need for network automation, if network automation can be done using Ansible, investigating the security aspects with manual work compared to automation in Ansible and lastly, can automation help save time. This report also goes through different APIs and what that means for network automation.

1.1 Background

Ever since the start of computer communications there has been a need for the administrator to manually login to the device to verify, update or fix the network. This process is slow and error prone. With automation this might be able to be done faster, better and cheaper.

1.2 Aim

The aim for this project is to investigate network automation primarily with Ansible and different aspects of it such as security, different APIs, connecting to different types of devices as well as expanding Ansible by using Python scripts.

1.3 Delimit

This project will only research network automation with Ansible core, there are other tools like Salt/Chef/Puppet as well as Ansible Tower but they will not be considered in this report.

1.4 Problem statement

This report aims to answer the following questions:

1. Why would anyone implement network automation?
2. Is it possible to implement network automation using Ansible?
3. Compare security aspects with manual work versus automation with Ansible
4. Can Ansible be used to audit the network devices and their configurations?
5. Can Ansible with Python be used to create a network topology?
6. How Ansible help save time?
2 Theory

The theory section will go through the needed theory to understand the project, it goes through YAML, Jinja2, Python, SSH, Network automation, API, Ansible, Cisco ASA and Cisco switches.

2.1 YAML

YAML (YAML Ain’t Markup Language [1, p.156]) is a data format that multiple different applications/devices can use as a standardized way of exchanging information. YAML is a superset of JSON (JavaScript Object Notation [1, p.170]), meaning what is written in JSON can be understood by YAML. YAML was created to be easy to read and write. A small data model in JSON taken from [1, p.170].

```json
{
  "authors":[
    {
      "firstName": "Jason",
      "lastName": "Edelman"
    },
    {
      "firstName": "Scott",
      "lastName": "Lowe"
    },
    {
      "firstName": "Matt",
      "lastName": "Oswalt"
    }
  ]
}
```

Listing 1: Small JSON data file

Rewritten in pure YAML:

```yaml
authors:
  - firstName: "Jason"
    lastName: "Edelman"
  - firstName: "Scott"
    lastName: "Lowe"
  - firstName: "Matt"
    lastName: "Oswalt"
```

Listing 2: Small YAML data file

YAML seems to be more readable by humans. YAML can have strings, integers, boolean values as well as arrays and dictionaries (dictionaries is the name used when discussing it with Python in mind, but it can also be called hashes, hash maps and they have a key and a value). It can also have different types of arrays such as lists, sets and tuples [1 155-156].

All YAML documents should start with − which symbolize the start of the document and should end with ... which symbolize the end of the document [3]. One example of a list is:

```yaml
Network devices:
  - Routers
  - Switches
```

Listing 3: YAML list

Another example of a list is:

---

1 Superset is a term when a set includes a set and additional entries [2 p.138]
Network devices: ['Routers', 'Switches']

Listing 4: YAML list

One example of a dictionary

Network devices:
  Name: R1
  Type: Router

Listing 5: YAML dictionary

Another example of a dictionary is:

Network devices: {Name: R1, Type: Router}

Listing 6: YAML dictionary

It is possible to verify if the syntax is correct using [http://www.yamllint.com/](http://www.yamllint.com/) Usually it is not enough using just one of the list or dictionary types but we can mix them to solve complex problems.

2.2 Jinja2

Jinja2 is a templating language made for Python. In itself it does not do much, as it needs data from a source like YAML/JSON to actually produce something valuable. Python can be used to take the data from YAML and combine it with Jinja2 to produce individual configuration to send to a device. Below there is a YAML file with hostnames, links, interface name as well as the ip address on the interface.

```yaml
nodes:
  R1:
    links:
      s1/0:
        ip: 10.1.0.1 255.255.255.0
        passive interface: False
      f0/0:
        ip: 192.168.101.11 255.255.255.0
        passive interface: False
      l0:
        ip: 1.1.1.1 255.255.255.255
        passive interface: True
  R2:
    links:
      s1/1:
        ip: 10.1.1.2 255.255.255.0
        passive interface: no
      f0/0:
        ip: 192.168.101.12 255.255.255.0
        passive interface: no
      l0:
        ip: 2.2.2.2 255.255.255.255
        passive interface: yes
  R3:
    links:
      s1/1:
        ip: 10.1.1.3 255.255.255.0
        passive interface: no
      s1/0:
```

Page 9
Below is the template to use with the YAML file to create configurations for the different nodes in the YAML file.

```
{% for node in Variables.nodes.keys() %}
  {{ node }}:
    {%- for interface in Variables.nodes[node].links.keys() | sort %}
      interface {{ interface }}
        ip address {{ Variables.nodes[node].links[interface].ip }}
    {%- endfor %}
{%- endfor %}
```

Listing 8: Jinja2 file using the data from the YAML file

The resulted output when Python combines them (the Python script can be found in the next section):

```
./PythonOutputOfConnectedJinja2AndYAML.py jinjatest.j2 nodes.yml
```

```
R1:
  interface f0/0
    ip address 192.168.101.11 255.255.255.0
  interface lo
    ip address 1.1.1.1 255.255.255.255
  interface s1/0
    ip address 10.1.0.1 255.255.255.0
R2:
  interface f0/0
    ip address 192.168.101.12 255.255.255.0
  interface lo
    ip address 2.2.2.2 255.255.255.255
  interface s1/1
    ip address 10.1.1.2 255.255.255.0
R3:
  interface lo
    ip address 3.3.3.3 255.255.255.255
  interface s1/0
    ip address 10.1.0.3 255.255.255.0
  interface s1/1
    ip address 10.1.1.3 255.255.255.0
```

Listing 9: Output of using the Jinja2 template with the YAML data
Jinja2 can have loops, if statements as well as variables. A variable names are taken from the data source (in above case YAML). {{}} is used around the variables to make substitution. The good thing about this is that the CLI knowledge can be directly applicable to the template. Loops in Jinja2 can loop over arrays and dictionaries (although in different ways) and then use if statements to check if something should be written or not. Loops and if statements have {% %} surrounding it.

If you want all the keys in a dictionary (to loop through all elements one by one) we can use (Jinja2 usage of the JSON/YAML case from YAML section):

```jinja
{% for author in Variables.authors %}

Listing 10: Jinja2 loop example

In above case the variable author holds the firstName and lastName of that particular part of the loop (meaning first loop through it has Jason, second time it is Scott and lastly it is Matt). The entire Jinja2 template for that case is:

```jinja
{% for authors in interface_list.authors %}
First name is: {{ authors.firstName }}
Last name is: {{ authors.lastName }}
{% endfor %}
```

Listing 11: Jinja2 using YAML data file

If we loop through over a dictionary we can do that using:

```jinja
{% for node in Variables.nodes.keys() %}
```

Listing 12: Jinja2 loop start

In above case Variables.nodes is a dictionary holding the different nodes (R1 with its data, R2 with its data and R3 with its data), but if we loop over the keys then the the variable node only holds the name of the node. Not the data of the node, meaning we have to use Variables.nodes[node] to get the data for the node in the node variable. We need to end the loop using:

```jinja
{% endfor %}
```

Listing 13: Jinja2 loop end

If we only wanted to extract the data from R1 we can use an if-statement:

```jinja
{% if node == "R1" %}
```

Listing 14: Jinja2 if statements start

and ending the if-statement with:

```jinja
{% endif %}
```

Listing 15: Jinja2 if statements end

To make Jinja more flexible and less copy-paste like it is possible for one template to read another template and incorporate it into the own template. Switches and routers could share some configuration such as SNMP/way of authentication among others, we could add that in a file called basecisco.j2 and then have another template for switches called ciscoswitches.j2 that starts with this line:

```jinja
{% include "basecisco.j2" %}
```

Listing 16: Jinja2 inheritance

and when rendering the ciscoswitches.j2 it will start with the basecisco file. This way if we need to update something for both the switches and routers we only need to update it in one place. This way the setup becomes more modular. We could also make this more complex by using operating system name, if we have Juniper routers and Cisco routers we could have a variable in the YAML file called OS: IOS or OS:

\[2\] Command Line Interface
Junos and then use if statements in the Jinja2 configuration for when they are different for the Juniper routers and the Cisco routers. A better solution could be to just split them up into different Jinja2 files altogether and have a base Jinja2 template that checks what OS it is and then choose another Jinja2 file to load depending on the answer.

### 2.3 Python and combining YAML and Jinja2

If we want to verify that the YAML and Jinja2 can work together then we do that using Python using this script (Slightly adapted from [1] p.193):

```python
#!/usr/bin/python
from jinja2 import Environment, FileSystemLoader
import yaml
import sys

# The base of this script is written in Networking Programmability and
# Automation book from O'Reilly (ISBN: 9781491931257), I adapted it
# slightly to my needs.
# This script takes in two variables during the script call, first the
# template to use and then the data to use with the template. If sys.
# argv is empty then the script says that and runs with standard
# template.j2 and data.yml. Script fails if template or data does not
# exist or if they are not correct Jinja2 and YAML documents. The
# reason why the variables can be hardcoded at the beginning is if the
# same ones are always checked then it might be easier if it is
# possible to possibly hardcode it.
Template="jinjatest.j2"
Data="nodes.yml"

if (len(sys.argv)==3):
    Template=sys.argv[1]
    Data=sys.argv[2]
elif (len(sys.argv)==2 or len(sys.argv)>3):
    print("Wrong amount of command-line arguments, run with 0 or 2
    arguments. If 2 then please add template to use and data to try
    out."");
    print("Trying with hardcoded template and data")

ENV = Environment(loader=FileSystemLoader('.'))
template = ENV.get_template(Template)

with open(Data) as f:
    interfaces = yaml.load(f)
    print(template.render(Variables=interfaces))
```

Listing 17: Python script for using YAML and Jinja2 files

Running the script using YAML and Jinja2 from the Jinja2 section:

```bash
./PythonCheckYAMLJinja2.py jinjatest.j2 nodes.yml
```

Listing 18: Executing Python script

and get this output:

```
R1:

interface f0/0
ip address 192.168.101.11 255.255.255.0
```
### Listing 19: Python script output

There is a use of Jinja templates and YAML in network automation, by using Ansible we can automate this so Ansible can take the YAML data and combine it with the Jinja2 template and save the configuration and sending it to the devices in question. Not all data from the YAML file must be used, like the passive interface part it could have been added for the routing protocol if needed.

### 2.4 SSH

Ansible uses SSH to communicate with the devices. Three ways of authenticating the client can be done with SSH and that is public-key, password and/or hostbased [4, p.215]. With public key authentication it is sending a public key and signing it with the private key, when the SSH server gets the message it can verify that the signature for the message can only be done with the private key connected to that public key. The server needs to know which keys to trust however. This is possible with Cisco routers/switches using the ip ssh pubk ey-chain command. If the client has created a public/private key-pair (can be created using PuTTYgen, ssh-keygen, using openssl or other methods. PuTTYgen will be shown here) then it can be added locally to /ssh/id_rsa for the private key and /ssh/id_rsa.pub for the public key. After opening PuTTYgen it is possible to choose what kind of key (RSA, DSA, ECDSA among others) as well as the length. 2048 bits RSA keys were chosen for this test, after deciding which key and strength pressed generate. When that was done it shows the key fingerprint and the public key, the fingerprint needs to be added to the Cisco router/switch like this (the colons needs to be removed first from the fingerprint from PuTTY):

```
ip ssh pubkey-chain
username admin
key-hash ssh-rsa C97EF10C55D8F7D7324602A2C6716287 admin
```

### Listing 20: Public key authentication Cisco IOS
It is possible to add it to the router using key-string instead of key-hash and then adding the entire public key, the router automatically changes that to the fingerprint for the running configuration. The public key needs to be added to the device that needs to be authenticated using just public-keys, it should be added to `/ssh/id_rsa.pub`.

The private key can be exported in PuTTYgen through the menu Conversions -> Export OpenSSH key. That file needs to be added to `/ssh/id_rsa` on the device that needs to be authenticated using just public-keys.

![PuTTY Key Generator](image)

Figure 1: PuTTY Key Generator

The second way of authenticating in SSH is password-based, a username and password could be setup on the server that the client needs to use to authenticate. For the router to actually allow SSH it first needs a device name, domain-name, public/private key-pair as well as allowing it on the VTY ports.

The last can be done using `aaa new-model` (it enables authentications using the local database for the vty ports) or by going to line vty 0 15 and transport input ssh as well as login local (last cannot be done when `aaa new-model` is enabled).

```
hostname R1
ip domain-name AnsibleLab
crypto key generate rsa modulus 2048 exportable
ip ssh version 2
```
enable secret cisco123
username Admin secret cisco123
aaa new-model
aaa authentication login default local enable

Listing 21: SSH authentication Cisco IOS

If we have admin account enabled using ip ssh public-key-chain and a local username Admin, we can use SSH to the device in two different ways:

```
marius@Borgen-kali:~ $ ssh Admin@192.168.101.11
R1> who
Line  User  Host(s)  Idle  Location
0 con 0  admin  idle  00:10:02
* 2 vty 0  Admin  idle  00:00:00 192.168.101.105

Interface  User  Mode  Idle  Peer Address
R1> Connection to 192.168.101.11 closed by remote host.
Connection to 192.168.101.11 closed.
marius@Borgen-kali:~ $ ssh Admin@192.168.101.11
Password:
R1> who
Line  User  Host(s)  Idle  Location
0 con 0  Admin  idle  00:00:18
* 2 vty 0  Admin  idle  00:00:00 192.168.101.105

Interface  User  Mode  Idle  Peer Address
R1> exit
Connection to 192.168.101.11 closed.
marius@Borgen-kali:~ $  
```

Listing 22: SSH authentication with Cisco IOS, password and public-key

Hostbased authentication is similar to public-key based, it is just that a client is authenticated and that client might have multiple users using it sharing the keys [4, p.215].

Why would we like to be authenticated using just a public/private key-pair? It is more secure (as long as the private-key file is properly secured) as well as it is needed for event driven automation (or a password saved locally on the device). Do we want scripts to run automatically or just when an administrator is present? [5, p.1016-1017] recommends public key authentication on most sites as it offers the best balance of strong security and convenience.

2.5 XML and data models

XML stands for eXtensible Markup Language. XML share some similarities to YAML, such as being inherently hierarchical. While YAML might be easier to read for a human, XML or JSON tend to be favored as data representation choices when software need to communicate with each other [1, p.162]. We can embed data within a parent construct like this[1, p.162]:

```xml
<device>
  <vendor>Cisco</vendor>
  <model>Nexus 7700</model>
  <osver>NXOS 6.1</osver>
</device>
```

---

3Not the most secretiv password but it is only in a testlab
4Not the most secretiv password but it is only in a testlab
5Default for AAA authentication for the VTY ports is to use the local database, it does not need to be specified in the VTY section of the configuration
XML share some similarities to HTML, but they were designed with different purposes in mind. XML was designed to carry data and HTML was designed to display data. XML has no predefined tags, the author needs to define both the tags and the structure of the document itself. Like YAML, XML does nothing on its own. To use XML we need something to take the data from the XML file to make something, like Jinja2 could take YAML data to make individual configurations. One such example is XSLT. XSLT is a language for applying transformations to the XML data.

One example of YAML data file:

```yaml
authors:
  - firstName: "Jason"
    lastName: "Edelman"
  - firstName: "Scott"
    lastName: "Lowe"
  - firstName: "Matt"
    lastName: "Oswalt"
```

The same written in XML:

```xml
<interfaces>
  <interface>
    <name>GigabitEthernet0/0</name>
    <ipv4addr>192.168.0.1 255.255.255.0</ipv4addr>
  </interface>
  <interface>
    <name>GigabitEthernet0/1</name>
    <ipv4addr>172.16.31.1 255.255.255.0</ipv4addr>
  </interface>
  <interface>
    <name>GigabitEthernet0/2</name>
    <ipv4addr>10.3.2.1 255.255.254.0</ipv4addr>
</interfaces>
```

It seems like YAML is a bit easier to read and write than XML or JSON. XML can be used to hold data about network devices as well such as XML interface data.
To make use of above XML file we could use an XSLT template like this[1, p.168]:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.example.org/
  → routerconfig">
  <xsl:template match="/">
    <xsl:for-each select="interfaces/interface">
      <interface><xsl:value-of select="name"/></interface>
      <ip address><xsl:value-of select="ipv4addr"/></ip>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

Listing 27: XSLT template

The resulting output of these two files will be[1, p.168-169]:

```
interface GigabitEthernet0/0
ip address 192.168.0.1 255.255.255.0
interface GigabitEthernet0/1
ip address 172.16.31.1 255.255.255.0
interface GigabitEthernet0/2
ip address 10.3.2.1 255.255.254.0
```

Listing 28: Result from using the XSLT and XML files

Even though it works, seems like YAML with Jinja2 is easier to use with a less effort. While XML, JSON and YAML are data formats, that’s not all that is needed. How can we be sure that the data in the variable is allowed? We could use something that is called a data model. Data models describe a constrained set of data with well-defined types and parameters [1, p.174]. Data models do not transport data and don’t care about the underlying transport protocols in use. XML has XSD [1] JSON has JSON Schema and then we have the network focused YANG that works with both JSON and XML. YANG is defined in RFC 6020 [7]. YANG provides the ability to define syntax and semantics, such as enforcing the VLAN IDs must be between 1 and 4094 or that the operational state of an interface must be "up" or "down" [1, p.174]. The data model basically defines the truth of what’s permitted.

We could have following data in XML:

```xml
  <tc:allvlans xmlns:tc="http://borgenstrand.com/ns/yang/YANGModel">
    <tc:vlans>
      <tc:id>100</tc:id>
      <tc:name>Data VLAN</tc:name>
    </tc:vlans>
  </tc:allvlans>
</ns:data>
```

Listing 29: XML example

With the following YANG statement:

```yaml
list vlan {
  key "id";
  leaf id {
    type uint32 {
      range "1..4094";
```
Listing 30: YANG data model

Above YANG statement we use the range keyword [7, p.114] and is used to restrict which integer (as the type is uint32) is allowed on the VLAN ID. Another keyword is the container that is mapped directly to hierarchy in XML and JSON [7, p.51-52]. To link the YANG model to something we use the concept of a namespace [7]. Both the client and server uses the namespace during the XML encoding and private modules are assigned without a central registry so the name must be chosen so they do not collide with standard or other namespaces [7, p.29]. The argument to the namespace statement is the URI of that particular namespace [7, p.42].

To validate an XML file with an YANG model we could install the Python library pyang (pip install pyang), it comes with multiple validation tools such as yang2dsdl and pyang. Another validation tool to check the YANG model is pyang and it can be executed using pyang YANGmodelfilename.yang and pyang will verify if the YANG model is written correctly or not. If we want to verify if the model and the data is compliant we could use yang2dsdl with the argument -v XMLfile YANGfile.

We will try this using below XML file:

```
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <allvlan xmlns="http://borgenstrand.com/ns/yang/YANGModel">
    <vlan>
      <id>100</id>
      <name>Data VLAN</name>
    </vlan>
  </allvlan>
</data>
```

Listing 31: XML file holding VLAN data

With below YANG model:

```
module YANGModel {
  namespace "http://borgenstrand.com/ns/yang/YANGModel";
  prefix "YANGModel";
  container allvlan {
    list vlan {
      key "id ";
      leaf id {
        type uint32 {
          range "1..4094";
        }
      }
      leaf name {
        type string;
      }
    }
  }
}
```

Listing 32: YANG model for verifying VLAN data

And then we use the yang2dsdl:

```
markus@Borgen-kali:~ $ yang2dsdl v YANGXML.xml YANGModel.yang
```

*Uniform Resource Identifier, defined in RFC3986*
Thus we showed that the data from the XML file was done correctly. Changing the VLAN number to 10000 and trying again with this result:

```
markus@Borgen-kali:~ $ yang2dsdl -v YANGXML.xml YANGModel.yang
```

Listing 34: Second test using yang2dsdl with a YANG model and a XML file

Even though YANG can enforce certain rules, if the VLAN number was wrong but still within the 1-4094 interval YANG will not find any errors, such as writing 10 instead of 100. YANG can also enforce that the VLAN ID must be unique by using the unique statement such as the line unique "id" [7, p.69-70].

If we wanted to take this even further we could create a valid YAML document from the YANG model. Following the instructions in [8] and then running pyang -f yaml -p release/models ../YANGModel.yang and we get a YAML document created from the YANG model.

```
markus@Borgen-kali:~ /openconfig$ pyang -f yaml -p release/models ../YANGModel.yang
```

Listing 35: Creating a YAML data file from a YANG model

It can also provide a skeleton XML document from the YANG file using pyang -f sample-xml-skeleton -p release/models ../YANGModel.yang.
Listing 36: Creating a XML data file from a YANG model

This means that we could start with the YANG model and later translate it to a valid data format document and then start populating the data in XML/YANG file.

To enforce JSON we could use a YANG model with Python library yangson.

2.6 Application programming interface

The traditional way of communicating with a device is using SSH and CLI for configurational (configuration commands) and operational (various show commands) data. This way is highly proprietary. This leads to APIs. API1 is a mechanism that is used for computer software on one device to talk to a software on another device [1, p.30]. One API is SNMP10 while quite useful it was created a long time ago and was not built to be a real time programmatic interface for network devices [1, p.30-31]. SNMP do not distinguish between configurational and operational data [9].

A new API for network devices is NETCONF (new compared to SNMP, NETCONF RFC4741 was released 2006). NETCONF is a network management layer protocol. NETCONF is connection-oriented protocol and commonly leverages SSH as its transport [1, p.32]. NETCONF uses port 830 as its default port but this can be changed if both the server and the client agrees on a port. Data sent between a client and a server is encoded in XML and remote procedure calls (RPC) are encoded in the XML document and the server processes these RPC’s. NETCONF supports transaction-based changes, this means that all changes within the XML document must succeed or the complete change is not applied to the device [1, p.33]. To restrict what data is allowed with NETCONF we could use YANG. NETCONF can be enabled in Cisco IOS-XE using the command netconf-yang [10] and in Juniper the command set netconf ssh [11]. Different models might support more features with NETCONF than others, Cisco IOS-XR and Junipers Junos support candidate configuration while HPE’s Comware 7 and Cisco IOS-XE do not [1, p.207]. Candidate configuration is a third datastore that is not like the running or startup configuration, candidate configuration is doing nothing until it is committed and then it will be in the running configuration [1, p.207]. Candidate datastore is part of RFC6241 and if a part of the candidate datastore cannot be committed to the running configuration then the running configuration must not be changed [12, p.53-55].

Another API is the REST API. REST API is based on the RESTful principles [13]. REST stands for REpresentational State Transfer and a system that follows the REST-based architecture are said to be RESTful [1, p.33]. The style relies on a stateless client-server model in which the client keeps track of the session and the underlying protocol is most commonly HTTP [1, p.33]. RESTful APIs operate just like HTTP-based systems where a web server is accessible via a URI11 and a client sends the associated HTTP request to the URL. There are multiple different types of requests that can be done such as GET (Obtain configuration or operational data), PUT (Create or replace a resource), PATCH (Create or update a resource object), POST (create a resource object) as well as DELETE (Delete a specified resource) [1, p.204-205]. The communication using the REST API is using HTTPS and an reply is always sent back [13]. The format of it is in JSON or XML [1, p.33-34]. RESTful APIs are also in use in the SDN controllers, most controllers on the market exposes a northbound RESTful APIs making it easier to automate [1, p.39]. An implementation of RESTful API is RESTCONF. RESTCONF is a REST API that uses XML- or JSON-encoded data that is defined by YANG models. RESTCONF can be used on the Cisco IOS-XE devices [1, p.251]. RESTCONF is described in RFC 8040.

9Application Programming Interface
10Simple Network Management Protocol
11Uniform Resource Locator
12Software Defined Networking
2.7 Ansible

Ansible is a open source tool by Red Hat with a big user community. Ansible is often described as a configuration management tool, we can write some kind of state description for our servers and then verify the state using Ansible [14, p.2]. Ansible can also be used in deployment, meaning moving over applications and installing it on the servers and installing it [14, p.2]. Ansible works by using Ansible playbooks to create Python scripts that is moved to the remote server and executing it, the communication between the devices happens using SSH. This approach does not work on all network devices though, it is possible to change the way Ansible works by making Ansible execute the Python script locally and then the script connects to the device and runs the commands [14, p.334] [1, p.362-363].

Ansible is agentless, meaning the remote devices do not need to install anything for Ansible to control it, except for SSH as well as Python (for some devices), Windows Remote Management (for Windows devices). The Ansible modules come with a help function called ansible-doc that can read information about the different modules such as running ansible-doc nxos_config will show which arguments that can be used with the nxos_config module as well as showing some examples of this module.

While Ansible normally uses SSH, it is not an requirement that it must only be able to use SSH. Arista EOS modules can use SSH or HTTP(S) [15] for connecting to Arista devices. Nexus NXOS modules can use SSH or HTTP(S) [16] for connecting to Cisco Nexus switches. Juniper Junos modules can use SSH (CLI) or XML over SSH (NETCONF) [17]. Using NETCONF requires Python library ncclient, NETCONF can be used in the netconf_config module or the Juniper specific modules which are described in the Junos OS Platform Options [17]. Ansible can work with different APIs, if a new device gets released with a new API then a new Ansible module can be created for that API.

Ansible uses playbooks to perform the different deeds that needs to be done. One playbook can have multiple tasks that may or may not be connected. The different tasks can use different modules. If a Cisco command needs to run in privileged mode there is a module called ios_command, if a global configuration mode command needs to run module ios_config exists. There are multivendor commands that can be ran using the NAPALM module, NAPALM can be installed using pip install napalm-ansible.

The installation of Ansible is easy, if a Debian-based OS is used it is possible to use apt to install it (some other OSes might use yum), or it can be installed through Python pip (sudo pip install ansible). Ansible uses some default files but if the same files exists in the current directory the default files will not be used. Default hosts file exists in /etc/ansible/hosts and default configuration exists in

![Figure 2: Entity-relationship diagrams](https://developer.cisco.com/site/python/outside Cisco IOS-XR, Arista EOS and Cumulus Linux supports running Python directly on the devices [1, p.363] [14, p.35])
It is good to have those files in the same directory as the playbooks so version-control system can verify those files as well [14] p.16-17.

The ansible.cfg file can hold information such as passwords (which is not recommended), timeout until Ansible throws an error (such as if a task takes too long), transport system which means which SSH client should be used (such as OpenSSH or Paramiko which is a Python SSH module) or pipelining (reduces amount of SSH sessions by copying the script as well as running it in the same session [14] p.207) among other configuration variables.

The hosts file is the inventory list that Ansible can use so it knows where to do the task. The hosts file has a simple format:

```
[GNS3]
R1 ansible_ssh_host = 192.168.101.11
```

Listing 37: Ansible hosts file

If we use the above hosts file we can connect to the device R1 by using its name or by using the groupname (the name in between []). Ansible can also have an hierarchy with the groups, we could have group CiscoRouters and CiscoSwitches and then have a group called Cisco:children.

```
[CiscoSwitches]
CiscoSwitch1 ansible_ssh_host = 192.168.101.21
CiscoSwitch2 ansible_ssh_host = 192.168.101.22

[CiscoRouters]
CiscoRouter1 ansible_ssh_host = 192.168.101.31
CiscoRouter2 ansible_ssh_host = 192.168.101.32
[Cisco:children]
CiscoSwitches
CiscoRouters
```

Listing 38: Ansible hosts file

If we have a need to push some configuration to all these devices (such as SNMPv3) then we can use the groupname Cisco, and it will automatically do for the devices in the other two groups, this way we do not have to copy paste between the groups. We could also have variables within the groups that the group Cisco will inherit [15]. An example of variables is shown below, if we have a need of different servers for the different groups we could add below in the hosts file (or a variable file normally located in ./group_vars/GROUPNAME if we want it to be added to the version control system, otherwise /etc/ansible/group_vars/GROUPNAME is recommended).

```
[CiscoSwitches:vars]
ntp_server = ntp1.example.com

[CiscoRouters:vars]
ntp_server = ntp2.example.com
```

Listing 39: Ansible variable file

The playbooks themselves are written in Ansible YAML format, and the indentations are very important. Tabs are not allowed. To make the playbooks more readable comments can be added using the # sign, everything afterwards on that line is ignored.

Boolean variables can be written in different ways but mean the same thing. If we want to set a variable to true it is possible to use true, True, TRUE, yes, Yes, YES, on, On, ON, y, Y, if we use it for a Ansible module we could use yes, on, 1 or true [14] p.24. A particular issue with this is if we have an variable in YAML called PortSec: yes and want to see in the Jinja2 template if it is set to yes, then we need to do an if-statement towards True, as they change it to True.

In the Ansible playbook we have hosts variable that is which hosts we should do something on, gather_facts if we want Ansible to automatically gather facts such as what type of CPU the device uses (we do not want to gather any facts this way as we use connection: local in most network playbook which means it gathers facts about the local computer running Ansible), connection which is if we run the resulting Python script locally or on the remote device and for most network devices we chose local. Then we have different tasks that the playbook will do. The - name: part of the playbook is just for the
administrator as it shows where Ansible currently is but it has no functionally for Ansible other than that. A short playbook to make these concept come together:

```yaml
---
- hosts: R1
gather_facts: no
collection: local
tasks:
  - name: Get credentials
    include_vars: ios.yml
  - name: Define connection
    set_fact:
      connection:
        authorize: yes
        username: "{{ ANSIBLE_NET_USERNAME }}"
        password: "{{ ANSIBLE_NET_PASS }}"
        auth_pass: "{{ ANSIBLE_NET_AUTH_PASS }}"
  - name: Gathering IOS interfaces up
    ios_command:
      commands: show ip int brief | include up
      provider: "{{ connection }}"
      register: showinformation
    debug:
      msg="{{ showinformation.stdout_lines[0] }}"
```

Listing 40: Ansible playbook

The first part is discussed previously so will start with the tasks, the first task is the get credentials tasks, this takes the saved credentials and stores it in the three different ANSIBLE_NET_* variables to make an SSH connection to the hosts (it is possible to make it truly password-less by using just public keys but that will be handled in the section about the security in Ansible). The credentials are stored in the ios.yml file and it looks like this (it is not a production password which is why the passwords are shown here):

```
ANSIBLE_NET_USERNAME: "admin"
ANSIBLE_NET_PASS: "cisco123"
ANSIBLE_NET_AUTH_PASS: "cisco123"
```

Listing 41: Ansible credential file

The next part is the interesting one, it uses the module called ios_command which will run a command on the Cisco router/switch in privileged mode. This particular command will be show ip interface brief | include up which will show which interfaces are administratively up and connected.

The feedback from the router will be stored in the variable showinformation. If we wanted the information from there we could later use or we could save it locally. After this we will print it using a debug play.

We can run the playbook like this:

```bash
ansible --playbook Playbook.yml
```

Listing 42: Executing an Ansible playbook

PLAY [GNS3]

    ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛

TASK [Get credentials]

    ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ 

ok: [R1]

TASK [Define connection]

    ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛️ ⬛

Page 23
ok: [R1]

TASK [Gathering IOS interfaces up]
  ↦ ********************
ok: [R1]

TASK [debug]
  ↦ ********************
ok: [R1] => {
  "msg": [
    "FastEthernet0/0 192.168.101.11 YES NVRAM up",
    "Serial1/0 10.1.0.1 YES NVRAM down",
    "Loopback0 1.1.1.1 YES NVRAM up"
  ]
}

PLAY RECAP
  ********************
R1 : ok=4  changed=0  unreachable=0  failed=0

Listing 43: Result from executing the Ansible playbook

If we are debugging a playbook we could add the flag -v to make Ansible more verbose, and we can go up to -vvvv and then it will show a lot of information, this is needed for troubleshooting when someone finds a bug and wants to report it or just an error within the playbook itself. If we do this on the above playbook the credentials from the ios.yml file will be shown which might be good or might be bad depending on what we are debugging we could make sure Ansible does not increase the verbose by using no_log: true in that particular task or tasks.

Ansible network modules support idempotency\(^\text{14}\) that means we can run the playbooks multiple times and it should not change or break anything [14, p.336]. As usual the playbooks needs to be tested, some might break things but so can every other scripting/network automation tool. There is a difference between configuring SNMP versus ensuring the SNMP configuration exists [1, p.383]. Before pushing out the configuration Ansible collects the current configuration and comparing them to verify if there is a need to push out the configuration, this way we might be able to avoid certain issues [1, p.383]. An example is when we send out a configuration change of the hostname on a router from R1 to R11.

The play is:

```yaml
- name: changing hostname
  ios_config:
  lines: hostname R11
  provider: "{{ connection }}"
```

Listing 44: Ansible play

First time we run the playbook we get this result:

```
TASK [changing hostname]  ********************
changed: [R1]
```

Listing 45: Output from Ansible first time

Second time:

```
TASK [changing hostname]  ********************
```

\(^{14}\)A term from mathematics, one of the ten laws in Boolean algebra is the idempotent law that shows\( f + f = f \) as well as\( f \cdot f = f \) [2, p.71-713]. It can also be used in set theory as the idempotent law \( A \cup A = A \) and \( A \cap A = A \) [2, p.139]. In the laws of logic there it is also a law of idempotent, \( p \lor p = p \) and \( p \land p = p \) [2, p.58].
Listing 46: Output from Ansible second time

Ansible can only compare what they can get however, if we have a play like this:

```
- name: Setting up RSA keys
  ios_config:
    lines:
    - crypto key generate rsa modulus 2048 exportable
      provider: "{{ connection }}"
```

The above play will send the command crypto key generate rsa modulus 2048 exportable, it will create a new set of RSA keys. This is the result of running above play:

```
TASK [Setting up RSA keys]
  changed: [R1]
```

Second time it is run it will show the same result. That means that this particular time idempotency is not achieved. Why does this matter? In this particular case, if we change the key-pair on the router and trying to login again with openssh we get this:

```
markus@Borgen-kali:~ $ ssh admin@192.168.101.11
@ WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!
@ WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
SHA256: hYIFraAQHZ47ZVxFmWvPZW1/GILF/UZw16Z9oa7MYo.
Please contact your system administrator.
Add correct host key in /home/markus/.ssh/known_hosts to get rid of this
message.
Offending RSA key in /home/markus/.ssh/known_hosts:3
remove with:
  ssh -keygen -f "'/home/markus/.ssh/known_hosts" -R "192.168.101.11"
RSA host key for 192.168.101.11 has changed and you have requested strict
checking,
Host key verification failed.
```

Above is a potential man-in-the-middle attack scenario, it is when someone/something is putting themselves between A and B and then they decrypt the traffic from A and then encrypting it before sending it to B, and the opposite as well decrypting traffic from B and encrypting it to A. If this is successful the middle man can read all the data between A and B and they do not know it is happening [4, p.107-108]. In above case it is not an man-in-the-middle it is just new keys, but as an administrator it might not be possible to know the difference and if above is happening they should not connect as doing that might give away administrator rights to the middle man. This does lead to the point that even if the Ansible modules themselves have idempotency incorporated in it, it might not be true idempotency as it can only compare the configurations but some configuration is hidden by the devices and then idempotency is not achieved. Testing of the playbooks is of utmost importance.

Ansible has modules to connect to a different NMS (Network management system) to schedule downtime for the nodes, such as to Nagios [19], PRTG (not official Ansible module [20] as well as bigpanda, sensu, zabbix among others [21]. If there is a need to schedule downtime for an hour for one host monitored by Nagios it is possible using the Nagios module [19].
# schedule an hour of HOST downtime, with a comment describing the reason
- nagios:
  action: downtime
  minutes: 60
  service: host
  host: '{{ inventory_hostname }}'
  comment: Rebuilding machine

Listing 50: Ansible nagios module scheduling downtime

If we just want to disable some services monitored by Nagios [19]:

- nagios:
  action: downtime
  minutes: 60
  service: frob, foobar, quez
  host: '{{ inventory_hostname }}'

Listing 51: Ansible nagios module disable service

If the administrator wants wait a minute after a task but before the next task it is possible to run the pause module [22]:

- pause:
  minutes: 1

Listing 52: Ansible pause module to pause the execution of the Ansible playbook

This to make sure the changes has propagated as needed such as changing the STP root device as that takes time to stabilize in the network or just making sure that Nagios has understood that a particular service should not give alerts and then the playbook continues with it work.

If we wanted to communicate to a device using NETCONF we could use the netconf_config module in Ansible. If we wanted to set up an NTP server on the device we could send below Ansible task [15]

- name: set ntp server in the device
  netconf_config:
    host: 10.0.0.1
    username: admin
    password: admin
    xml: |
      <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
        <system xmlns="urn:ietf:params:xml:ns:yang:ietf-system">
          <ntp>
            <enabled>true</enabled>
            <server>
              <name>ntp1</name>
              <udp><address>127.0.0.1</address></udp>
            </server>
          </ntp>
        </system>
      </config>

Listing 53: Ansible netconf_config module

This could be a lot more flexible then using individual modules for sending out this NTP server to Cisco IOS, Cisco ASA, Juniper Junos as well as to Arista EOS (that means it will be one task rather then four tasks/plays). As seen in the 2.5 the YANG model used in the previous Ansible NETCONF example was urn:ietf:params:xml:ns:yang:ietf-system, which is defined in RFC7317 [23]. If we would rather use a specific YANG model for a specific device we need to check which YANG models have been created by

---

15 This example is from ansible-doc netconf_config
that vendor, such as in [24]. From there going to vendor/cisco we can see different YANG models for different models and operating systems of Cisco devices. Different Juniper YANG models can be found here [25]. It is recommended to use git to clone the repository. If we have an Juniper Junos 17.4R1 we could use http://yang.juniper.net/junos/conf/vlans (located in [25] under 17.4/17.4R1/junos/conf/junos-conf-vlans@2017-01-01.yang) YANG model to enforce certain data for VLANs for when NETCONF is trying to add a VLAN to a device.

2.8 Security aspects in automation in Ansible

Ansible can handle authentication through password based SSH or through public-keys. The public/private key-pair means that the client has a public/private key-pair and that the server knows the public key or its fingerprint. That way we can skip the password.

The other way Ansible can handle authentication is through password based SSH. It can handle this in various ways such as a static password in the playbook (might be good for debugging), through variables that the playbook loads, through the hosts file with a static password for the group or on each host, through Ansible Vault or that Ansible asks the administrator for the passwords (For some network devices it will be an SSH password and an enable secret password).

If ansible-playbook Playbook.yml -ask-pass is used Ansible asks the administrator for the SSH password. This way no password needs to be written in the playbook or in another file.

Password in play can be done under the provider dictionary in that play like this:

```yaml
ios_facts:
  gather_subset: hardware
  provider:
    username: admin
    password: cisco123
    authorize: true
    auth_pass: cisco123
```

Listing 54: Ansible authentication in the task

It is not a flexible way of handling authentication as if a password is updated it needs to be updated in all plays everywhere, however we could use a variable substitution instead like ANSIBLE_NET_PASS and having that variable in another YAML file or in the hosts file directly. Another way is to set a fact in the playbook, then it is possible calling that variable in all the plays. This way the password is written just once in each playbook. The way we can do this is as follows:

```yaml
tasks:
  - name: Define connection
    set_fact:
      connection:
        username: admin
        password: cisco123
        authorize: true
        auth_pass: cisco123
```

Listing 55: Ansible authentication in beginning of the play

If we do not want a static password as above we could just use a variable substitution instead like {{ANSIBLE_NET_PASS}}. This variable might be saved in an YAML file or through different hosts/group variable files. If we do above we could just use the variable called connection under the provider in the plays like this:

```yaml
provider: "{{ connection }}"
```

Listing 56: Ansible authentication using variable created in beginning of play

This way there is less lines at each play which increases readability. If we use a password like above we should add no_log: true in the set_fact play otherwise the password could be shown when running a play and debugging is used.
There is one more way of handling the passwords and that is through the ansible-vault. If we have a file called secret.yml with the username/password for the playbook we could encrypt it using Ansible-vault:

```
markus@Borgen-kali:~ $ ansible-vault encrypt secret.yml
New Vault password:
Confirm New Vault password:
Encryption successful
```

Listing 57: Ansible encrypting file using vault

The good thing about this vault is that the file is encrypted:

```
[captionpos=b,caption=Reading encrypted vault file]
$ANSIBLE_VAULT;1.1;AES256
6133365393066866838613766386328636326336332646383461383456132623562336532436
623066343064337662393532613233562383864387537390a3883936363666363636368366636367646533
65363613236453733363636261366130396243638383138731333231353366642373035
6536138666338310a61613435343633161616346638343430393133336161633654373966
3636363237643765313031393663238350366523232643983734933666437626162383135
6262353031614630663636164341338303963363636166438326663133436363636636343666
3737313336813023632616361343966332364323932323438353232632636343626333
376636163301326232831656138643861373334633431323273630631366139646639653438
3865
```

But the way we read the variables from the file in the playbook is the exact same way. If it worked before the encryption it will work after, the only difference is that we need to add a flag when using the script and provide the vault password:

```
ansible-playbook Vault.yml --ask-vault-pass
```

Listing 58: Executing Ansible playbook with a vault password

If we provide the wrong vault password we get this:

```
avatar-playbook Vault.yml --ask-vault-pass
Vault password:
ERROR! Decryption failed (no vault secrets would found that could decrypt) on /home/markus/Desktop/ansibleProject/ansibleVault/secret.yml
```

Listing 59: Executing Ansible playbook with wrong vault password

If we run the playbook without providing the –ask-vault-pass flag we get this:

```
avatar-playbook Vault.yml
ERROR! Attempting to decrypt but no vault secrets found
```

Listing 60: Executing Ansible playbook without vault password

If we have the password in a file we could do this:

```
markus@Borgen-kali:~ $ ansible-playbook Vault.yml --vault-password-file ./password.txt
```

Listing 61: Executing Ansible playbook with vault password from file

and it will run without a problem.

We could protect the RSA keys in the same way if we want, then we need to provide the vault key to decrypt the vault.

The problem with the vault key password in a file approach is that we are basically back to where we started, if we have a vault password to get the real password how can we protect that file? The keys to the kingdom is basically located in the ./password.txt file. We need to secure that file properly, only the needed administrators should have access to that file as any other could potentially steal the keys to
the kingdom. That file as well as the id_rsa private key file needs to be secured properly and only the
needed owner and possibly owner group should have access to them. This can be done by using chown, chgrp and chmod in Linux. If we want user Admin1 to be the only one with access to the file id_rsa then we
could run the command chown Admin1 id_rsa and then chgrp Admin1 id_rsa and lastly chmod 400
id_rsa, this way Admin1 user and Admin1 group is associated with the file id_rsa and that the owner
has read access and no one else has any access. First number (in binary) is associated with the user,
second with the group and last number with the rest. The number itself is that the reading has number 4, writing has number 2 and execute rights has 1. If we want all three we get 7, if only read-write is
needed we get a 6. The problem with this is that we are trusting the underlying operating system to
protect the file. If we do above for the network engineer Admin1 but user Admin2 who is an Linux
engineer has sudo rights to the box the Linux engineer could get around the above rights as with sudo
they could change the rights if needed.

For the security minded administrators it is possible to ask the administrator for both passwords, the
SSH password and the password to go into privilege mode (second password is only if needed, certain
devices could setup that logging in with a over SSH will make the user go straight into privilege mode
and then only the SSH password is needed].

```
markus@Borgen-kali:~$ ansible-playbook Vault.yml --ask-pass --u admin --ask-
< become-pass
SUDO password [defaults to SSH password]:

PLAY [R1]
< ********************************************

TASK [Define connection]
< ********************************************
ok: [R1]

TASK [Show CDP Neighbour]  ********************************************
ok: [R1]

TASK [debug]
< ********************************************
ok: [R1] => {
  "msg": {
    "Device ID Local Interface Holdtime Capability Platform"
    "Port ID"
  }
}

PLAY RECAP
< ********************************************
R1 : ok=3  changed=0  unreachable=0  failed=0

Listing 62: Executing Ansible playbook asking for username, password and privileged password

If the argument --ask-pass (for the SSH connection itself), -u admin (for the username) as well as the
--ask-become-pass (for the password to the privilege mode) is used both passwords will be asked the user.
If second password (for the privilege mode) is left blank Ansible will default to the SSH password which
might or might not be the correct password.

Part of the security is Authentication, Authorization and Accounting, hence AAA. Authentication is
who is trying to do something, authorization is if the entity is permitted what level of access should be
given and accounting is saving records of the action that was done with who actually did the record [26
p.9-10]. If proper accounting records is needed then each administrator should use these arguments with
Ansible --ask-pass -u USERNAME --ask-become-pass so the records show who actually ran the playbook
instead of only AnsibleAdmin or such an account.
2.9 Network automation

There are other tools of network automation outside Ansible. As usual with computers it possible to do something in multiple ways, if we want to send out a new local user to a Cisco Router in an Ansible play we could have this (of course we need to have a hosts file and so on):

<table>
<thead>
<tr>
<th>name: Setting up a new user.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ios_config:</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>lines:</td>
</tr>
<tr>
<td>- username Markus secret cisco123</td>
</tr>
<tr>
<td>provider: &quot;{{ connection }}&quot;</td>
</tr>
</tbody>
</table>

Listing 63: Ansible task for creating a new user in Cisco IOS

We can do the same directly in Python:

```python
# Imports the netmiko library
from netmiko import ConnectHandler

# The device information. The difference between password and secret is that
# password is the SSH password and the secret variable is the
# privilege mode password.
device = ConnectHandler(device_type='cisco_ios', ip='192.168.101.11',
                        username='admin', password='cisco123', secret='cisco123')

# the command to send to the device.
SendingCommands=['"username Markus secret cisco123"]

# Below sets of commands prints out the current prompt, first in usermode
# then in privilege mode and lastly in global configuration. This is
# shown so the administrator can see what happens, the print lines can
# easily be removed if needed.
print (device.find_prompt ())
device.enable ()
print (device.find_prompt ())
device.config_mode ()
print (device.find_prompt ())

# Sending the command to the Cisco device and saving the output on the
# device in the variable output.
output=device.send_config_set(SendingCommands)

# Printing out the output from the Cisco device
print (output)

# Closing the SSH connection.
device.disconnect ()
```

Listing 64: Python script for creating a new user in Cisco IOS

There are some big differences between above two scripts however, Ansible modules support idempotency by default (homemade modules or new modules might not support idempotency immediately) while the Python script will send out the command by default. If we need to support idempotency in the Python script we need to add more code which will make it harder to write. Another difference is that Ansible can connect to multiple devices at once (default amount of forks is 5 but can be changed to 25 or 100 if needed, all that is needed is to add line `forks = 25` in the ansible.cfg file) while the Python script by default does not. It is possible to add each SSH connection into a thread but this will make the script even harder to write and control. Ansible hides a lot of complexity so it is easier to write playbooks then trying to do the same directly in Python.

Outside the configuration part, network automation can truly help troubleshooting by connecting to multiple devices in parallel and returning the data in a structured way as well as the network administrators having the same way of doing the tasks instead of relying on each administrators own best practice [1] p.11]. Ansible also has support for asynchronous tasks using the async clause, this means that a
particular task in a play can start and Ansible will start with the next task without waiting for the async task to be done, this can decrease total running time of Ansible [14] p.213-214. If a play moves over a big file to a device or from a device it might make sense to add async to the file transfer task if the next task does not depend on the previous task. Ansible can also run a task over a single device at the time or a certain percentage. If load balancing is used in the network it is possible to use the serial command to decide the play should be done on a single number of devices at the same time or half, serial: 1 shows that only one device at a time should run this task and serial: 30% will make Ansible take half of the devices at a time [14] p.174-176. To make the same happen in Python will increase the complexity of the code. Everything Ansible can do can be done in directly in Python. Ansible is written in Python after all.

A better way is to describe the intent and not the low-level configurations, if we wish to deny an IP address through the network we could use an access-list and access-group in a Cisco device and a filter-list in a Juniper device but this means we would not use the same configuration on these devices [27] p.69. If we could describe the intent behind it it would be possible to send it to multiple types of devices at once with the same intent configuration. NAPALM goes some distance to solve this problem, if we wish to check out a routers BGP neighbors we could use this in Ansible:

```yaml
- name: Get facts with napalm
  napalm_get_facts:
    provider: "{{ connection }}"
    optional_args:
      inline_transfer: True
      secret: "{{ ANSIBLE_NET_AUTH_PASS }}"
    dev_os: {{ OS }}
    filter: "bgp_neighbors"
    register: result

- name: print data from Napalm get facts
  debug: var=result
```

Listing 65: Ansible task to gather multivendor BGP neighbor information

We need to have an variable called OS for each OS so NAPALM knows it is ios for Cisco routers and switches, junos for Juniper and eos for Arista devices for this to work. The result we get is this:

```yaml
ok: [R2] => {
  "result": {
    "ansible_facts": {
      "napalm_bgp_neighbors": {
        "global": {
          "peers": {
            "10.1.1.3": {
              "address_family": {
                "ipv4": {
                  "accepted_prefixes": 2,
                  "received_prefixes": 2,
                  "sent_prefixes": 2
                }
              },
              "description": "",
              "is_enabled": true,
              "is_up": true,
              "local_as": 10,
              "remote_as": 100,
              "remote_id": "10.1.1.3",
              "uptime": 394
            },
            "192.168.101.11": {
```
"address_family": {
  "ipv4": {
    "accepted_prefixes": 3,
    "received_prefixes": 3,
    "sent_prefixes": 2
  }
},
"description": "",
"is_enabled": true,
"is_up": true,
"local_as": 10,
"remote_as": 10,
"remote_id": "192.168.101.11",
"uptime": 417
}
"
"router_id": "192.168.101.12"
}
"
"changed": false,
"failed": false
}

Listing 66: Resulting BGP neighbor information

The good thing about above is that NAPALM parse the output from the network devices and gives the same answer on each device which means we could loop through the peers dictionary and get the information from each neighbor. This is better then manually running each show command for the BGP information from each devices and then parsing through them manually. NAPALM is a Python library so it can be used directly in Python as well.

Different devices handles network automation in different ways, Arista EOS can use configuration sessions, this means all commands in a session is applied in one transaction and this means that we could remove part of the configuration and adding some and not lose OSPF neighborships [1, p.540-541].

Following scenario is from [1, p.539-541]. If we have following configuration:

```
router ospf 100
  network 10.0.0.10/32 area 0.0.0.0
  network 10.0.1.10/32 area 0.0.0.0
  network 10.0.2.10/32 area 0.0.0.0
  network 10.0.3.10/32 area 0.0.0.0
  network 10.0.4.10/32 area 0.0.0.0
max-lsa 12000
```

Listing 67: Arista OSPF configuration

If we want to send below configuration to an Arista EOS device using NAPALM load_merge_candidate function and printing out the compared config (using compare_config in NAPALM):

```
no router ospf 100
router ospf 100
  router-id 100.100.100.100
  network 10.0.4.10/32 area 0.0.0.0
  network 10.0.5.10/32 area 0.0.0.0
max-lsa 12000
```

Listing 68: Changing Arista OSPF configuration

We could get something like this:
Because of the configuration sessions the Arista EOS device do not lose OSPF neighborships on the link with IP 10.0.4.10/32. If the device took one command at a time all OSPF neighbors would be torn down as the OSPF process itself was removed.

Another part of network automation is the event driven network automation, Python could be used to check through syslog files and running different Ansible Playbooks depending on what is found in the log files or the NMS might do it when it gets an alert. Nagios could be setup to use event handlers to run different Playbook depending on the issue at hand [28].

2.10 Cisco ASA

Cisco ASA (Adaptive Security Appliance) is a type of firewall from Cisco. The ASA CLI is similar to the Cisco route/switch series but not the same. One big difference is that the interfaces needs a logical name such as inside and outside and if these names are used inside gets a security-level of 100 and outside of 0. The security-level is used by the ASA to determine if traffic should go through or not and it goes from higher to lower by default and the return traffic is allowed. A DMZ should then have a security-level between 1 and 99. To allow other traffic it is possible to use MPF or a plain ACL. MPF has three parts to it and they are class-map, policy-map and service-policy [29, p.479]. Class-maps is where the specific traffic flows are identified or classified. Policy-map is where an action is taken on the traffic from the class-map and lastly policy-map is setting the entire policy on one or all interfaces on the ASA [29, p.479]. MPF was designed to be modular so an service-policy can contain one or more policy-maps and the policy-map can contain multiple class-maps and the class-maps can be used in multiple policy-maps. The default MPF on an ASA allows for outwards ICMP traffic but does not allow return traffic by default as ICMP traffic is not inspected. Another difference between the ASA and the routers/switches from Cisco is that no wildcard mask is used in the ASA, only the subnet mask.

ASA could be using transparent mode or routed, meaning it could be inline on layer two so the host devices do not need to be aware of it or it could be a layer three firewall [29, p.629]. ASA’s do not support CDP or LLDP which means they do not participate with switches and routers using these protocols. If it is running as a transparent firewall traffic could be blocked from going from one interface to the other, same way as outside interfaces cannot originate traffic to the inside network by default but return traffic could be allowed.

It is possible to control the ASA using CLI, Adaptive Security Device Manager (ASDM) or Cisco Security Manager [13]. ASDM and CSM are graphical user interfaces. Outside these alternatives Cisco has released a REST API for the ASA [13].

2.11 Cisco switches and VLAN

VLAN[18] is a way to segment local area networks so they become smaller and that the broadcast traffic does not reach all devices on the network [30, p.8]. Outside the standard VLAN there exists RSPAN[19] so it is possible to copy traffic on a switch to send to another switch for monitoring purposes [20, p.352]. If we need to secure hosts from reaching other hosts in the same VLAN it is possible to use Private-VLAN. Private-VLAN can segment the hosts so the isolated ports can only reach the promiscuous (Such as a link to the default-gateway) port and no other ports. If there exists a pair of servers that need to

---

16Demilitarized zone
17Modular Policy Framework
18Virtual Local Area Network
19Remote Switched Port Analyzer
have access for each other for load-balancing purposes or backup it is possible to use community ports. Members of the same community can reach each other and the promiscuous ports. The promiscuous ports can reach all other ports. Isolated and community VLANs are considered secondary VLAN and they are connected to the primary VLAN. Normal VLANs can be created with vlan X. If we have an RSPAN VLAN we need to write remote-span under the VLAN configuration. For PVLAN we need to add private-vlan isolated/community on the secondary VLAN and private-vlan primary on the primary VLAN as well as the private-vlan association X,Y where X,Y are the secondary VLANs. If VTP\textsuperscript{20} is used and the switch is configured as a client it is not possible to add VLANs and if it was server PVLAN cannot work for VTPv1 and VTPv2, PVLAN works with VTPv3 however.

\textsuperscript{20}VLAN Trunking protocol
3 Method

This section presents the technical solutions used in the project.

The code written in for this project is available at the following URL:
https://github.com/Borgenstrand/Ansible

3.1 Ansible

Network automation was implemented in Ansible version 2.4.2.0 with Python version 2.7.14+. The Ansible version was not a active choice as it was the version from the apt repository. One external Ansible module was used and that was NAPALM.

3.2 Network devices

The development of the project scripts has been done against Cisco router c7200-advipservicesk9-mz.152-4.55 in GNS3 as well as against two Cisco c3750 Catalyst switch with IOS c3750-advipservicesk9-mz.122-46.SE.bin. ASA’s used were ASA5505 Base license with operating system version 9.1(5) and virtual ASA5520 VPN Plus license with operating system version 9.1(5)21.

3.3 Operating system

The Ansible playbook has been run from a Linux server with the Kali Linux 4.14.0-kalil-amd64 operating system. The GNS3 devices has been running on a Microsoft Windows 10 Pro device.

3.4 Requirements capture

Requirement capture was done using Slack and phone calls to network engineers. Slack is a real-time collaboration app and platform and the networkto-code.slack.com forum was used.

First requirement from a CCIE in UK:

1. backup configs on kit on a regular basis
2. Use Ansible to configure templates / configurations for multiple devices
3. Check all devices are still running / connected when deployed to site

Reasoning behind the first was to have a good backup of the different configurations on the devices in the network. Reasoning behind the second was to control the configuration on the devices and that they should only be added configuration from Ansible, this is to avoid snowflake systems. Reasoning behind the third was to verify devices up and running.

Second requirement from an network engineer have also seen a need to audit the network devices and their configuration as well as configuring VLAN in a standardize way.

Third requirement was from an network engineer named Pär in Swedish Transport Administration (Trafikverket), recommended opening and closing of firewalls and comparing the time needed to do this manually versus through automation.

3.5 Time measurements

Time measurements were taken using a stopwatch for the manual tests and through the Linux command time for the automated usage.

21 Snowflakes are all unique which might be pretty to look at but might be very hard to connect to each others in a secure and stable way [27, p.379-380]
4 Implementation

This section presents the implementations details used in this project.

4.1 Network devices

Two Cisco 3750 switches were used with IOS version: c3750-advipservicesk9-mz.122-46.SE.bin. Cisco routers were c7200 with C7200-ADVIPSERVICESK9-M IOS in GNS3. ASA’s used were ASA5505 Base license with operating system version 9.1(5) and virtual ASA5520 VPN Plus license with operating system version 9.1(5)21.

4.2 Ansible

Ansible itself was installed using apt-get: `sudo apt-get install ansible`

Some dependencies of Ansible are an SSH client and OpenSSH is usually installed in Linux systems, if not it can be installed using apt or yum and if network modules are used paramiko library in Python needs to be installed. It is installed using pip install paramiko. Ansible ships with certain amount of modules such as ios_config for configuring Cisco switches and routers, asa_config for configuring ASA’s, eos_config for configuring Arista devices, junos_config for Juniper devices as well as others. Non-standard modules can be installed from Github or through pip. NAPALM was installed using pip install napalm-ansible.

Below configurations were created to test Ansible towards Cisco routers, switches and towards Cisco ASA’s. The hosts file included three groups, one for routers, one for switches and one group for ASA’s.

```
[ Routers ]
R1 ansible_ssh_host = 192.168.101.11
R2 ansible_ssh_host = 192.168.101.12
R3 ansible_ssh_host = 10.1.1.3

[ Switches ]
SW1 ansible_ssh_host = 192.168.101.91
SW2 ansible_ssh_host = 192.168.101.92

[ Firewalls ]
ASA1 ansible_ssh_host = 192.168.1.1
ASA2 ansible_ssh_host = 192.168.101.62
ASA3 ansible_ssh_host = 192.168.101.63
ASA4 ansible_ssh_host = 192.168.101.64
ASA5 ansible_ssh_host = 192.168.101.65
```

Listing 70: Ansible hosts file

The Ansible configuration file included a few lines to specify which inventory to use unless it is taken as an argument, checking the host RSA key is disabled for the lab as GNS3 devices recreates the key when GNS3 starts up. The timeout was set to 15 seconds instead of 5 as some tasks takes longer to do, certain times this might need to be increased further. Library and action_plugins is needed for NAPALM. Running napalm-ansible after installing NAPALM will show which lines to add to the ansible configuration file for NAPALM to work. ansible.cfg:

```
[defaults]
inventory = ./hosts
host_key_checking = False
timeout = 15
#vault_password_file = PassVault
library = /home/markus/.local/lib/python2.7/site-packages/napalm_ansible/
  ➳ modules
action_plugins = /home/markus/.local/lib/python2.7/site-packages/
  ➳ napalm_ansible/plugins/action
forks = 25
```

Page 36
Listing 71: Ansible configuration file

This file holds the authentication details. `ios.yml`:

```
[ssh_connection]
pipelining = true
```

Listing 72: Ansible credential file

```yaml
ANSIBLE_NET_USERNAME: "admin"
ANSIBLE_NET_AUTH_PASS: "cisco123"
ANSIBLE_NET_PASS: "cisco123"
```

### 4.3 Ansible VLAN

This section will describe the implementations of adding of VLAN on Cisco switches using Ansible. The topology in question can be found just below:

![Topology for the adding of VLAN](image)

The Ansible playbook will create two normal data VLAN, one Voice vlan, one RSPAN\(^{22}\) VLAN and one Private-VLAN with one isolated and one community VLAN associated to it. This will later be compared to writing this manually. All configuration files and script files are located in [A.2](#).

The structure of the files for this task looks like this:

```
AnsibleVLAN
  └── AddingVLAN.yml
  └── ansible.cfg
  └── configs
```

\(^{22}\)Remote-SPAN
The nodes.yml file has the data about the VLANs in question, it consists of a dictionary called VLAN (that needs to be reached from the Jinja2 file) and this dictionary holds a dictionary about all the VLANs. VLANID exists for all VLANs as well as a description. If it is an RSPAN there are an variable called RSPAN: True and the Jinja2 file has an if-statement looking for this variable. If it is part of the PVLAN there is an variable called PVLAN: True and also an variable called PVLANTYPE that shows what type of PVLAN that particular VLAN is (Isolated/community/primary).

The Jinja2 file starts of with changing the VTP mode to transparent and then looping through all the VLANs in nodes.yml file and creating configuration files for each switch. It looks if certain variables exists and if so adds configurations for that type of VLAN such as remote-span for the RSPAN VLAN and private-vlan primary/isolated/community for the PVLANs. After this it done it loops through the VLANs again to check if it is an primary PVLAN and if so it goes through all the secondary VLANs and adds them in an variable that is later used for the private-vlan association command.

AddingVLAN.yml is the Ansible playbook for creating the VLAN configuration as well as pushing it out to the devices. It starts with which devices this should be pushed out to and it is the Switches group. After this it creates the variable connection with the username and password from the ios.yml file. After this is done it uses the Jinja2 file with the nodes.yml to create the individual VLAN configurations and these files will be located in the configs folder with the hostname_OnlyVLAN.cfg name and after this is done it pushes out these configurations to the devices.

### 4.4 Ansible active

This section will describe the implementations of verifying connectivity with Cisco switches, routers as well as ASA’s using Ansible.

The structure of the files for this task looks like this:

```
AnsibleActive
    Ansible.yml
    ansible.cfg
    hosts
    ios.yml
```

The Ansible playbook is called Active.yml and all configuration files and script files are located in

```
AnsibleActive
```

The playbook starts with an play for the switches and routers and it starts with a defining the hosts as the Routers and Switches group in the hosts file. It checks if it is possible to get ios_facts (Ansible module) from each device. If not the playbook writes which device it cannot SSH into. After this play is done it goes to the next play which checks if it is possible to SSH to the devices in the Firewalls group and writes which devices it cannot access. Due to the fact that ASA’s and IOS devices do not share network modules it became two different plays in the same playbook.

### 4.5 Ansible audit

This section will describe the implementations of auditing Cisco switches, routers as well as ASA’s using Ansible.

The structure of the files for this task looks like this:

```
AnsibleAudit
    ansible.cfg
    ASAJinja.j2
    Audit.yml
    compare.bash
    configs
    firewall.yml
    hosts
    ios.yml
```
The individual data for the switches and routers are located in the nodes.yml file and the individual data for the ASA’s are located in the firewall.yml file. Basic configuration was created for the devices, the ASA’s have an site-to-site VPN to each other and the switches has two normal data VLAN as well as an VLAN for voice. They also have an RSPAN VLAN and an PVLAN. The routers each has a BGP session to each other.

The Ansible playbook is called Audit.yml and all configuration files and script files are located in A.2. The playbook starts with an play for the switches and routers and it first uses the Jinja.j2 template that in itself uses either JinjaRouters.j2 or JinjaSwitches.j2 depending on which group the current device is in with the according to the hosts file.

After this is done the play uses NAPALM to compare the created configuration to the current configuration. The difference will be added to a file called Difference_HOSTNAME.cfg in the config folder.

After the first play is done the playbook starts with the second play that is for the ASA’s. Similarly to the first play it starts with creating a configuration for each device using the ASAJinja.j2 template with the firewall.yml data source. After this is done it uses the asa_config module to run show running and save the output to a variable called showconfig. This variable will then be saved locally on the device running Ansible. After this is done the play will compare the show running to the generated template and add the comparison to a file called Difference_HOSTNAME_TemplateToShowRun.txt in the config folder and then it will do an reverse comparison meaning from the template to the show running. The comparison is done using grep and then piping it into sed and then saving the result.

4.6 Ansible ASA closing

This section will describe the implementations of opening/closing a Cisco ASA’s using Ansible.

The topology in question can be found just below:

![Topology for the ASA testing](image)

Figure 4: Topology for the ASA testing
The structure of the files for this task looks like this:

AnsibleASAWithInspection
  ├── ansible.cfg
  │    └── ASAFirewallOpening.yml
  │    └── ASAJinjaACLCreation.j2
  │    └── ASAJinjaFWObject.j2
  │    └── ASAJinjaFWRemove.j2
  ├── configs
  │    └── firewall.yml
  │    └── hosts
  │    └── ios.yml
  │    └── Jinja.j2
  │    └── JinjaRouters.j2
  │    └── JinjaSwitches.j2
  └── nodes.yml

The Ansible playbook is called ASAFirewallOpening.yml and all configuration files and script files are located in [A.2]. The individual data for the ASA’s are located in the firewall.yml file.

The playbook starts with creating the configurations that will be pushed out to the devices. It starts with the removing of old ACL line 1 (the one blocking) and then removing the object-group. The next part is creating the new object that will be pushed out and it loops through the SharedACL dictionary that holds all the ACLs and those ACLs holds their DenyObjects. In the DenyObjects dictionary it has all the DenyObjects that the ASA needs to have and it holds all the destination and source TCP/UDP ports it should block. The last template is the new ACL that needs to be created and added to the configuration.

After these three template has gone through for each device it will save the output to Ansible variables to be used later.

The playbook starts with clearing away the old ACL line 1 and the Object-group, if they do not exists the Play should continue without stopping.

Next part is pushing out the new object-group to the ASA’s and after that is done it will setting up the the ACL using the Object-group on line 1 and allowing everything else.

This Ansible playbook can use data from the firewall.yml and blocking these ports on all the ASA’s in the chosen direction.

4.7 Ansible with Python for layer 2 diagram

This section will describe the implementations of creating a layer 2 diagram using Ansible and Python. No diagram will be shown for this part as this script creates a diagram.

The structure of the files for this task looks like this:

AnsiblePythonCDPDiagram
  ├── ansible.cfg
  │    └── CDPDiagram.yml
  │    └── configs
  │    └── hosts
  │    └── ios.yml
  │    └── nodes.yml

The Ansible playbook is called CDPDiagram.yml and all configuration files and script files are located in [A.6].

The Ansible playbook starts with which hosts this should be checked upon and it is the Routers and Switches groups in the hosts file. After this the play continues by executing the show cdp neighbors | begin Device ID command and saving it to a variable called showinformation and saving this locally on the machine running Ansible in the CDP folder. After this is done the play executes the Python script called PythonDiagram.py and only does this once.

The PythonDiagram.py script checks if there are any files in the CDP folder and if so it first reads through all the information from all the files and removing the first line (line including Device ID) but saving all other rows. After this is in the Python memory it removes all the files from the CDP folder to avoid issues with false information if a device has been running but not anymore.

After this the script goes through all the lines but only saving the current hostname with local
interface and saving the remote hostname and remote port ID, the holdtimes, capabilities and platform information are not remembered. The script double checks if a link has already been saved (R1 to R2 or R2 to R1 can be the exact same link). This information is saved in a dictionary as a tuple key and tuple value and this dictionary is fed into graphviz to create the graph.

The Python script was extended to remove STP blocked ports from the diagram as well to create a diagram according to CDP and STP.

AnsiblePythonCDPDiagramWithSpanningTree

ansible.cfg

CDP

CDPDiagramWithSTP.yml

configs

hosts

ios.yml

nodes.yml

Physical network.png

PythonDiagramWithSpanningTree.py

STP

STPRoot

VLAN0001.png

The CDPDiagramWithSTP.yml playbook similarly as CDPDiagram.yml except that is in two plays, first play saves the show cdp neighbors | begin Device ID information from the Routers group. Next play is the same for the Switches group as well as checking spanning-tree blocked ports using show spanning-tree blocked ports | include VLAN and saves that in the STP folder. It also checks who is the STP root by using the command show spanning-tree | include VLAN |This bridge is the root and saves this in the STPRoot folder.

The script follows the same structure as without STP data except for going through the dictionary and removing the blocked ports so only the open ports are shown in the VLAN topologies. The playbook for the STP extension is CDPDiagramWithSTP.yml and the new Python script is PythonDiagramWithSpanningTree.py and these files are located in A.7.

4.8 Ansible security

The security aspect of Ansible was tested using simple playbooks but with different arguments and different ways of saving the password such as through the ansible-vault.
5 Results

This section has the results.

5.1 Time usage

Testing of ASA closing manually gave this result:

Table 1: ASA manually

<table>
<thead>
<tr>
<th>Amount of devices</th>
<th>Which devices</th>
<th>Total time in seconds</th>
<th>Average per device</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ASA1</td>
<td>83.48</td>
<td>83.48</td>
</tr>
<tr>
<td>2</td>
<td>ASA1-2</td>
<td>176.23</td>
<td>88.115</td>
</tr>
<tr>
<td>3</td>
<td>ASA1-3</td>
<td>232.91</td>
<td>77.64</td>
</tr>
<tr>
<td>4</td>
<td>ASA1-4</td>
<td>306.54</td>
<td>76.635</td>
</tr>
<tr>
<td>5</td>
<td>ASA1-5</td>
<td>398.51</td>
<td>79.702</td>
</tr>
</tbody>
</table>

Total average over all 15 tries is 79.845 seconds per device.

Testing of ASA closing using Ansible gave this result:

Table 2: ASA Ansible

<table>
<thead>
<tr>
<th>Amount of devices</th>
<th>Which devices</th>
<th>Total time in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ASA1</td>
<td>19.611</td>
</tr>
<tr>
<td>2</td>
<td>ASA1-2</td>
<td>49.900</td>
</tr>
<tr>
<td>3</td>
<td>ASA1-3</td>
<td>62.935</td>
</tr>
<tr>
<td>4</td>
<td>ASA1-4</td>
<td>67.872</td>
</tr>
<tr>
<td>5</td>
<td>ASA1-5</td>
<td>68.335</td>
</tr>
</tbody>
</table>

Testing of only virtual ASA opening using Ansible gave this result:

Table 3: ASA Ansible test 2

<table>
<thead>
<tr>
<th>Amount of devices running</th>
<th>Which devices tested against</th>
<th>Total time in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ASA2</td>
<td>45.606</td>
</tr>
<tr>
<td>5</td>
<td>ASA2</td>
<td>58.475</td>
</tr>
<tr>
<td>5</td>
<td>ASA3</td>
<td>57.220</td>
</tr>
<tr>
<td>5</td>
<td>ASA4</td>
<td>55.729</td>
</tr>
<tr>
<td>5</td>
<td>ASA5</td>
<td>54.875</td>
</tr>
</tbody>
</table>

Graph comparing ASA closing using manual work and Ansible.
Figure 5: ASA testing

![ASA testing chart](image1.png)

Figure 6: CPU usage running four virtual ASA's

![CPU usage chart](image2.png)

Testing of creating VLAN manually gave this result:
Table 4: VLAN manually

<table>
<thead>
<tr>
<th>Amount of devices</th>
<th>Which devices</th>
<th>Total time in seconds</th>
<th>Average per device</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SW1</td>
<td>99.23</td>
<td>99.23</td>
</tr>
<tr>
<td>1</td>
<td>SW1</td>
<td>85.73</td>
<td>85.73</td>
</tr>
<tr>
<td>1</td>
<td>SW1</td>
<td>84.56</td>
<td>84.56</td>
</tr>
<tr>
<td>2</td>
<td>SW1-2</td>
<td>196.19</td>
<td>98.095</td>
</tr>
<tr>
<td>2</td>
<td>SW1-2</td>
<td>171.39</td>
<td>85.695</td>
</tr>
<tr>
<td>2</td>
<td>SW1-2</td>
<td>170.39</td>
<td>85.195</td>
</tr>
</tbody>
</table>

Total average over all 9 tries is 89.72 seconds per device.

Testing of creating VLAN using Ansible gave this result:

Table 5: VLAN Ansible

<table>
<thead>
<tr>
<th>Amount of devices</th>
<th>Which devices</th>
<th>Total time in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SW1</td>
<td>7.752</td>
</tr>
<tr>
<td>1</td>
<td>SW1</td>
<td>8.162</td>
</tr>
<tr>
<td>1</td>
<td>SW1</td>
<td>7.945</td>
</tr>
<tr>
<td>1</td>
<td>SW2</td>
<td>7.678</td>
</tr>
<tr>
<td>1</td>
<td>SW2</td>
<td>6.679</td>
</tr>
<tr>
<td>1</td>
<td>SW2</td>
<td>6.649</td>
</tr>
<tr>
<td>2</td>
<td>SW1-2</td>
<td>7.854</td>
</tr>
<tr>
<td>2</td>
<td>SW1-2</td>
<td>6.714</td>
</tr>
<tr>
<td>2</td>
<td>SW1-2</td>
<td>7.834</td>
</tr>
</tbody>
</table>

Graph comparing VLAN creation using manual work and Ansible.

![VLAN testing graph](image-url)
5.2 Security aspects

Security in Ansible using SSH, passwords can be written during runtime, as an argument, using passwords from files, from Vault and using RSA keys.

Manually using SSH can be done using passwords during runtime, as an argument, using passwords from files and using RSA keys. Examples using manual SSH:

```
markus@Borgen-kali:~$ sshpass -p cisco123 ssh admin@192.168.101.11
R1>
R1> exit
Connection to 192.168.101.11 closed.
markus@Borgen-kali:~$ echo cisco123 > sshpass.txt
markus@Borgen-kali:~$ sshpass -f sshpass.txt ssh admin@192.168.101.11
R1>
```

Listing 73: Manual SSH testing with password and public-key authentication

If passwords from a file is used then the security hinges on the operating system itself:

```
markus@Borgen-kali:~/Desktop/AnsibleProject$ cat ios.yml
cat: ios.yml: Permission denied
markus@Borgen-kali:~/Desktop/AnsibleProject$ sudo cat ios.yml
ANSIBLE_NET_USERNAME: "admin"
ANSIBLE_NET_AUTH_PASS: "cisco123"
ANSIBLE_NET_PASS: "cisco123"
```

Listing 74: File security unencrypted

It is the same for RSA keys:

```
markus@Borgen-kali:~$ cat .ssh/id_rsa
cat: .ssh/id_rsa: Permission denied
markus@Borgen-kali:~$ sudo cat .ssh/id_rsa
------BEGIN RSA PRIVATE KEY------
```

Listing 75: RSA key file security

Within Ansible there is the ansible-vault which encrypts an password file.

```
markus@Borgen-kali:~$ ansible-vault encrypt secret.yml
New Vault password: 
Confirm New Vault password: 
Encryption successful
```

Listing 76: Encrypting file using Ansible vault

Result of reading secret.yml:

```
$ANSIBLE_Vault: 1.1: AES256
61333653930666363938613766336362383632623636326438346138343561326235623653
234366230653430643730623935326132333562383964373537390a3863663764366653665
3637646536536636313235643537333663562613661303962343634383831383731333213
53336646237303563561396666338310a6161343534633161616146683434303431333
36661130365343796663863636323764376330319336363238350305621326326493983
73439336664376261626382313562623503161643066363961643131830396336631656438
32656631333436636330633436663373731333353861303236163613439646332364323
932323438353332632626363564326333376636163303132623238316561386438538365
3463343132373630631336139646696534383865
```

Listing 77: Reading of encrypted file

To be able to run the playbook that has the Vault protected password file an additional argument needs to be added:
Running the playbook without the --ask-vault-pass argument will result in an error:

```
markus@Borgen-kali:~ $ ansible --playbook Vault.yml --ask-vault-pass
ERROR! Attempting to decrypt but no vault secrets found
```

Decrypting the password using a password file is possible with the argument --vault-password-file.

```
markus@Borgen-kali:~ $ ansible --playbook Vault.yml --vault-password-file ./password.txt
```

5.3 Ansible audit
Auditing network devices see results under A.3.

5.4 Ansible active
Output for running the Active.yml playbook

```
markus@Borgen-kali:~ $ ansible --playbook Active.yml
PLAY [Routers:Switches] *********************************
TASK [Define connection] *********************************
ok: [R1] => {"msg": ["Device ID Local Interface Holdtime Capability Platform Port ID"]}

PLAY RECAP
R1 : ok=3 changed=0 unreachable=0 failed=0
```

ASA2 was turned off to introduce an error.
## 5.5 Ansible diagrams

Testing of creating diagram using CDP information gave this result:

<table>
<thead>
<tr>
<th>Role</th>
<th>ok</th>
<th>changed</th>
<th>unreachable</th>
<th>failed</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASA2</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R3</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SW1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SW2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Testing of creating diagram using CDP and STP information gave these two diagrams:
Figure 10: VLAN0001 topology with blocked STP ports removed
6 Analysis

Ansible can connect to many different types of devices using different modules that might use different APIs and once the playbooks have been written it is very easy to run these playbooks.

It is possible to save time using Ansible as is shown in [5.1] 308.51 seconds it took doing one task manually in the five ASA while Ansible did the same using 68.335 seconds. Using Ansible on only ASA1 (physical ASA) took only about 19.611 seconds while the same on only ASA2 (virtual ASA) was 45.606 seconds if only ASA2 was running but if all four virtual ASA’s it took about it took 58.475 to do the same task on the ASA2. Running four ASA’s virtually took about 75%-85% CPU usage. The time it took to do this task over one virtual ASA when all were running compared to doing it to all four virtual ASA’s and the physical one was small but compared to doing it manually it was a big difference. This leads to the possibility of saving even more time if the ASA environment was all physical instead of mostly virtual in the same computer.

Average time creating multiple VLANs manually on two different switches 179.32 seconds while the same took Ansible about 7.47 seconds. The time saved depends on how many devices and what Ansible is doing. As Ansible forks multiple connections and runs them in parallel the time difference between one or multiple devices is not that big, it depends on CPU available in the Ansible host as well as the network devices and the amount of traffic on the network. The time used by Ansible seems to be depending on the playbooks themselves as well. In the VLAN example, a source file was just to push to the device. This did not work for the asa_config module, so it had to be split up into multiple parts which means the device could not just press out the config once and be done with it, it had to do it in steps.

The security aspects of Ansible compared to manual work share a lot of details as Ansible is using OpenSSH for clients with Python installed and Paramiko for network devices without Python. The ability to take the password from a file, authenticate using RSA keys or take the password from an administrator is the same. While it is possible to make sure normal users cannot read the non-encrypted password files, this cannot be said for users with sudo rights. With sudo rights it is possible to read the file. To avoid this it is possible to encrypt the file using the Ansible Vault. In this case only the password for the Vault password is needed. If the user want to avoid leaving any trail with the password, encrypted or not, the user could use the -ask-pass, -u, -ask-become-pass arguments to specify SSH password, username and privilege mode passwords to use.

As seen in [A.5] running Ansible to audit the devices can make sure it is the right configuration currently running on the device in question. The playbook itself compares the template to the running configuration (or vice versa) but the devices add or change certain commands such as the password for the site to site VPN on the ASA’s which will make some commands stick out. If the template says ***** then the password itself will be ***** which is not a secure password but if the running configuration says ***** it is just the ASA hiding the password. The same thing could be said about the shutdown/no shutdown, no shutdown will not be shown in the running configuration, only shutdown. Shutdown is default on interfaces in Cisco routers but it will clearly say shutdown, in Cisco switches the interfaces are no shutdown by default to make the switches more plug and play. The administrator could either change the templates to include anything or the administrator could accept this and check the resulting difference files for anything out of the ordinary. In the ASA5505 (physical ASA), switchport access VLAN 1 is standard but still added to the template as it should be VLAN 1 on that particular port. While it is possible on certain devices to do a show running-config all to show all configuration including defaults, this might make the Jinja2 template unwieldy.

Part of the template is the setup of site-to-site VPN’s between the the ASA’s, combining YAML data format with Jinja2 templates creates multiple site-to-site VPN’s so all the traffic between each inside interface gets sent over the VPN. If another ASA is needed such as a new office needs an ASA, it is easy to add the new device to the YAML file and then combining it with the Jinja2 file to get their own configuration which includes all the site-to-site VPN with each other.

Ansible pulled the CDP neighbors information to make Python create a layer two topology of that data. The results ends up with a current diagram for how the network looks according to CDP which includes all devices running CDP. This only works for Cisco devices as only Cisco routers and switches support CDP (ASA’s do not), but for an only Cisco network this is a way of getting an accurate layer two diagram. A copy of the script was then extended to include STP blocked ports and remove them from the topology so only the open ports are shown, this way the VLAN topologies shows the way the frames

---

24 `show running-config | begin interface FastEthernet1/0/44 shows only row ! under it while show running-config all | begin interface FastEthernet1/0/44 has 19 rows plus the ! row on the C3750 switch`
could move over the switched network. Transparent firewalls could stop this however, if a transparent firewall is between two switches then it could be that CDP could only flow one direction which means the CDP topology would not be complete. If the devices do not run CDP, the topology will be incomplete as well. The visual representation is not perfect either, part of the Fas number is hidden in Physical network diagram.

Ansible can easily connect to a device and saving the running configuration as seen in A.8 and this can be used to keep a copy of old working configurations.
Network automation could be used to automate repetitive task to save time and reduce mistakes as well as to avoid false alerts in the monitoring system during a change. Ansible is a tool to automate tasks on servers, clients as well as on network devices. Automation will not make the CLI disappear completely, as automation usually needs SSH/HTTPS access to the device and that needs to be setup first time using the CLI but after that automation can help out to make new changes faster and better. Network automation scripts could help out less skilled personnel doing certain tasks such as a Helpdesk agent checking if a neighborship exists between the ISP and the client. The scripts could be created by a senior network engineer for the Helpdesk team to help save time for the network engineers. If we wanted to make sure that the data used with XML or JSON is correct we could use YANG, this will help make sure the sent data is valid such as enforcing an IP address to be of a certain format, VLAN ID to be between 1 and 4094 or that certain variables must be unique.

Ansible can use all scripting languages that the device running Ansible can so it is easy to extend certain modules to Perl/Ruby (or other languages) instead of Python if needed as well as creating simple scripts to run from the lookup pipe that can run scripts directly on the local machine. It is possible for Ansible to disable alerts on the NMS before actually changing something on the devices this will avoid making false alerts due to a schedule change and afterwards enable the alerts again, this could save time and reduce call outs due to alert. Ansible can connect to a device using SSH and run command after command (module ios_confg is one example) or it can run NETCONF to send XML files containing the change needed (such as with modules netconf_confg and junos_config). Certain modules also support HTTPS (module nxos_confg is one example). With YAML, Jinja2 as well as internal and external modules Ansible is very flexible. If we add in certain devices possibility to do configuration sessions, we can reduce amount of downtime as well. One example of this is removing the OSPF configuration and recreating it with the correct network statements would not cause the neighborships to go down if the neighbors link was in the OSPF before and after the change. This leads to good network automation requires a lot from the devices itself. While it is possible to automate using the CLI, it is not as flexible as some APIs. More and more devices are enabling different APIs such as NETCONF or RESTCONF. If a device supports NETCONF and a YANG model has been created it is possible to create a skeleton XML file from the YANG model, then the administrator just needs to add the data to it and then send it. APIs makes it easier to structure the data sent and received, which will make it better for automation purposes. Using Cisco IOS, the interface given by show cdp neighbors | begin Device ID and show spanning-tree blockedports | include VLAN have different formats (Fast 1/0/22 compared to Fa0/0/22). This means that the automation tool could misunderstand it as the strings sent back from the device are different. In this case Python needed to parse through the strings and change them to a shared standard. If an update to IOS changes the output to a show command, scripts using that show command could suddenly start failing. This leads to the vendors themselves have a responsibility to not change things too drastically so all automation tools fails. This mean that the vendors themselves needs to open up the devices to be able to properly be controlled using automation tools in a structured way. While Ansible can do a lot of different things, what is possible depends on the devices themselves as well as Ansibles modules. Cisco ASA’s have a RESTful API, but as it is now, Ansible do not have an official module that supports it. For the ASA it is only asa_confg, asa_command and asa_act that are official ASA modules.

The time it took for Ansible to add VLANs on one switch versus two seems to have been basically the same. Ansible is forking these connections so they are done in parallel, the slowest device will make the entire process take longer. Doing the task manually repeatedly over multiple devices increases the time very much but increasing amount of devices in Ansible did not by the same degree. Ansible forks tasks and can run certain tasks asynchronous if needed. Asynchronous means that Ansible will not wait for the task to be done before starting the next one. As shown in the results section 5 Ansible can help save time compared to manual work, the amount depends very much on the task at hand and for how many devices it should do it on as well as how the playbook is written. During the manual testing some VLAN names needed to be input twice as they were either misspelled or a uppercase was used instead of a lowercase or the opposite, the same could not be said for the Ansible testing. Humans mistakes could always happen during manual repetitive work while the same task done by a computer will do the same every time. Ansible is consistent. If the playbook is being used multiple times then it can help save time, if we spend a lot of time doing a playbook only used once it might be better just to do it in the CLI directly.
Ansible can be made as secure as manually doing the tasks except that Ansible can do it faster and more consistently. For normal Linux servers Ansible uses OpenSSH which is a default SSH client on most Linux systems and Paramiko for connecting to network devices. In an environment with multiple network administrators I would personally use --ask-pass -u USERNAME --ask-become-pass arguments so the accounting part of AAA will document who is actually running the playbooks as otherwise it might just say it is admin running it. For event driven automation it could be username admin or EventAnsible doing it.

Outside of the managing the devices centrally and using similar templates for similar devices (to avoid having snowflake systems), sometimes configuration could be added to the device even though no person added it to the device. On C2960X with IOS 15.2(2)E3 using stackwise an access-list could be added to the running configuration after the Master gets switched over and becomes the Master, an access-list called CISCO-CWA-URL-REDIRECT-ACL will then be added to the running-configuration [31]. Using the audit playbook it is possible to see this discrepancy.

The diagrams created by CDPDiagram.yml and CDPDiagramWithSTP.yml creates diagrams but have different purposes, for an only router environment the first might be preferred but for a switched network the latter creates individual topologies per VLAN with only active links. The STP blocked ports are removed and the label of the VLAN diagrams shows the STP root switch. The script only works for Cisco devices running CDP. The Python script can be enhanced for LLDP so non-Cisco devices could participate as well, it is possible to extend it so different sections of the network running CDP and others run LLDP could save individual files and later added to the diagram. The script saves the interconnection between the nodes in an dictionary, so the script could first run all of this over the CDP information and then similarly over the LLDP information and the script would not have to be changed much. Certain devices do not participate with CDP/LLDP and will make this script incomplete, ASA’s do not participate in CDP/LLDP and might actually block it if it is running as a transparent firewall. Some devices do not support LLDP or CDP at all.

As seen Ansible can connect to many different types of devices using many different APIs which makes Ansible flexible and powerful, but everything ultimately depends on what is allowed on the device itself. What APIs do the device support? How are they implemented? Some devices support NETCONF candidate datastore while others do not. This could be a problem with the script requires a candidate datastore. If we want Ansible playbooks to be run at certain intervals we could use the Linux cron to schedule this.

7.1 Ethical and social aspects

Working computer communication is more and more important as more and more people and devices depend on it every day. Ilmola and Casti talks about food and fuel shortage in stores in Finland if the Internet goes down for just two days as well as people cannot pay using credit cards [32, p.13]. In worst case it might even be riots [32, p.13].

Due to the fact that an administrator could potentially cause disaster in networks either due to a mistake, ignorance or malice there are code of ethics IT professional could follow such as System Administrators’ Guild code of ethics [33] which goes through conflicts of interest, insider trading, bribery, corruption, fraud as well as compliance with law, expenses among others. Other code of conduits [34] talk about professionalism, personal integrity, education as well as ethical responsibilities such as maintain a safe, healthy and productive workplace.

According to a report from IBM X-Force[35], employee errors led to a 424% increase of misconfigured cloud infrastructure. According to a report from Kaspersky Lab and B2B International, 46% of IT security incidents are caused by the employees each year and the staff might be too scared to report is as they fear punishment or are embarrassed by the failure [36].

It might be human nature to make mistake and to try and hide it for fear of punishment or embarrassment, but that could make the problem itself so much worse.

Automation is part of making network changes faster and more stable such as to avoid human errors, automation might introduce further issues faster then ever before unless it is tested thoroughly. Human error might cause a serious issue on a critical router/switch while automation might do the same on all routers/switches under a company’s authority. Automation might actually be part of causing the issue Ilmola and Casti talked about in [32]. If an administrator wanted to they could do a lot more damage using automation then normally login in to one device at a time. If an administrator wanted to they could erase the startup-configuration on all network devices and then commit a reboot of all devices. As
in other fields, it is up to the individual to not cause damage.

Automation can also be used to build more resilient IT infrastructure which is part of United Nations sustainable development goals [37, Section 9]. Part of the facts from United Nations:

Inadequate infrastructure leads to a lack of access to markets, jobs, information and training, creating a major barrier to doing business.

Automation could make it easier, faster, better and cheaper to invest in countries with subpar IT infrastructure. Social sustainable development talks about workplace security, social protection, better prospects for personal development and social integration [38] and economic sustainable development talks about maintaining human and material resources in the long run. Automation might be helping humans have more time then doing repetitive tasks but might also save up so much time that there is no need for as big of a workforce or that the workforce might take on new tasks instead.

7.2 Future work

Network automation is not an end in itself, it is more of a journey. Automation needs to be adopted by everyone within the organization for it to last. Automation is here to stay, the same might not be said about Ansible. Ansible is a automation tool that works without agent and is flexible with what it can do and can easily be enhanced through new modules. But is Ansible better then Salt, Puppet, Chef or other automation tools? Future work could include more of an event-driven approach to network automation by integrating Ansible with different NMS’s.

8 Conclusion

This report has described Ansible for network automation which includes YAML for data format files, Jinja2 for templates, Python, SSH and different APIs used with automation. Ansible is an open source tool by Red Hat with a low barrier to get started with network automation and has a big user community.

The security in Ansible depends on SSH and may or may not have passwords stored locally. Ansible can be as secure as the administrator wants it to be such as using RSA key-pair to authenticate or asking the administrator about which username and password to use. Ansible defaults to OpenSSH and Python paramiko (it can use other Python libraries as well).

Using Ansible to automate creation of VLAN as well as to open/close ports in Cisco ASA’s as well as to compare the time it takes to do this task manually. Creating multiple VLAN including RSPAN VLAN and PVLAN using Ansible took between 6.714-7.854 seconds while the same task manually took between 170.39-196.19 seconds. Opening multiple firewalls at once using Ansible took about 68.335 seconds while manually it took 308.51 seconds and the Ansible time was extended as four of the firewalls was running virtually which increased the time needed by Ansible. The time saved depends on what is being done and how many devices it is doing it on. There are times when time is not the most important aspect, such as when upgrading devices currently doing load balancing. In that case we do not want all of them offline at the same time. Ansible can handle doing it on a subset of them if needed.

Ansible was used to audit network devices and compare the running configuration to a configuration created by Jinja2 templates using YAML data, this is to verify the device is running the configuration it should.

Ansible was used together with Python to create network diagrams for how the network looks right now, this includes creation of Spanning-tree diagrams which could be useful for troubleshooting layer two issues. Ansible can easily be enhanced with Python script or new modules written in Python (or other languages such as Perl or Ruby).

There are many reasons to have some kind of network automation within an organization, which includes saving time on certain tasks (one-off tasks might not save time), do it consistently to avoid human errors as well disabling and then enabling network alerts. It might also make it easier for less skilled staff to do certain tasks as the complex work has gone into the script, YAML, Jinja2 and other parts. Such as the audit example, all ASA’s have site-to-site VPN’s between each other, if needed to add another ASA to this it is possible to to add another host in the YAML file and running the audit script and the configuration for this new ASA is there including all the configuration about the site-to-site VPN’s between each other, this happens within seconds of running the Ansible Playbook. What is
shown in this report is a small part of what Ansible can do, Ansible can be extended with new modules for new APIs or new devices.

If we wanted event-driven automation we could use an NMS that could locally run different Ansible playbooks depending on the issue at hand such as Nagios getting a warning/error and then using event handlers to run a playbook to fix the issue. Network automation is here to stay.

Paraphrasing Jason Edelman [39], be the automator, not the automated.
9 Bibliography

References


A Appendix

This section has the configuration and script files.

A.1 Ansible files

hosts:

```
[ Routers ]
R1 ansible_ssh_host = 192.168.101.11
R2 ansible_ssh_host = 192.168.101.12
R3 ansible_ssh_host = 10.1.1.3

[ Switches ]
SW1 ansible_ssh_host = 192.168.101.91
SW2 ansible_ssh_host = 192.168.101.92

[ Firewalls ]
ASA1 ansible_ssh_host = 192.168.1.1
ASA2 ansible_ssh_host = 192.168.101.62
ASA3 ansible_ssh_host = 192.168.101.63
ASA4 ansible_ssh_host = 192.168.101.64
ASA5 ansible_ssh_host = 192.168.101.65
```

Listing 82: Ansible hosts file

```
[defaults]
inventory = ./hosts
host_key_checking = False
timeout = 15
#vault_password_file = PassVault
library = /home/markus/.local/lib/python2.7/site-packages/napalm_ansible/
  → modules
action_plugins = /home/markus/.local/lib/python2.7/site-packages/
  → napalm_ansible/plugins/action
forks = 25

[ssh_connection]
pipelining = true
```

Listing 83: Ansible configuration file

```
Ansible_NET_USERNAME: "admin"
Ansible_NET_AUTH_PASS: "cisco123"
Ansible_NET_PASS: "cisco123"
```

Listing 84: Ansible credential file

A.2 Ansible VLAN

Ansible script for adding VLAN on Cisco Switches that do not appear in the Ansible files section A.1.

The Ansible playbook AddingVLAN.yml:

```
- hosts: Switches
gather_facts: no
```
connection: local
vars_files:
- ios.yml
- nodes.yml

tasks:
- name: Define connection
  set_fact:
    connection:
      authorize: yes
      username: "{{ ANSIBLE_NET_USERNAME }}"
      password: "{{ ANSIBLE_NET_PASS }}"
      auth_pass: "{{ ANSIBLE_NET_AUTH_PASS }}"

# Below is just VLAN configuration
- name: Generate VLAN configuration for each device
  template:
    src: "./jinjaOnlyVlan.j2"
    dest: "./configs/{{ inventory_hostname }}_OnlyVLAN.cfg"

- name: Pushing the configs created using JinjaAndYAML
  ios_config:
    src: "./configs/{{ inventory_hostname }}_OnlyVLAN.cfg"
    provider: "{{ connection }}"

Listing 85: Ansible playbook AddingVLAN.yml

Jinja2 template for the VLANs:

vtp mode transparent
{% for VLANs in VLAN.keys() %}
vlan {{ VLAN[VLANs].VLANID }}
  name {{ VLAN[VLANs].VLANDescription }}
  {% if VLAN[VLANs].RSpan is defined %}
    remote-span
  {% endif %}
  {% endif %}
  {% endif %}
  {% if VLAN[VLANs].Pvlan is defined %}
    private-vlan {{ VLAN[VLANs].PvlanType }}
  {% endif %}
  {% endif %}
  {% endfor %}

{% for VLANs in VLAN.keys() %}
  {% if VLAN[VLANs].Pvlan is defined %}
    {% if VLAN[VLANs].Pvlan == True %}
      private-vlan {{ VLAN[VLANs].PvlanType == "primary" %}
      vlan {{ VLAN[VLANs].VLANID }}
    {% else %}
      vlan {{ VLAN[VLANs].VLANID }}
      {% set ListofPvlan = "{\"VLAN\": "nn\"} %}
    {% endif %}
  {% endif %}
  {% if VLAN[VLANs].Pvlan is defined %}
    {% if VLAN[VLANs].Pvlan == True %}
      {% if ListofPvlan .update({\"VLAN\": ListofPvlan.VLAN+VLAN[VLANs].VLANID | string +\"",}) %}
        {% endif %}
    {% endif %}
  {% endif %}
{% endfor %}

Page 59
Listing 86: Jinja2 template for playbook AddingVLAN.yml

YAML data file for VLAN and switch configuration:

```yaml
nodes:
  R1:
    links:
      Serial1/0:
        ip: 10.1.0.1 255.255.255.0
        passiveinterface: False
      FastEthernet0/0:
        ip: 192.168.101.11 255.255.255.0
        passiveinterface: False
      Loopback0:
        ip: 1.1.1.1 255.255.255.255
        passiveinterface: True
    ospf:
      rid: 1.1.1.1
    BGP:
      rid: 1.1.1.1
      AS: 10
      Network: ['10.1.0.0 mask 255.255.255.0', '192.168.101.0']
      Neighbors:
        192.168.101.12: 10
        10.1.0.3: 100
  R2:
    links:
      Serial1/1:
        ip: 10.1.1.2 255.255.255.0
        passiveinterface: no
      FastEthernet0/0:
        ip: 192.168.101.12 255.255.255.0
        passiveinterface: no
      Loopback0:
        ip: 2.2.2.2 255.255.255.255
        passiveinterface: yes
    ospf:
      rid: 2.2.2.2
    BGP:
      rid: 2.2.2.2
      AS: 10
      Network: ['10.1.1.0 mask 255.255.255.0', '192.168.101.0']
      Neighbors:
        192.168.101.11: 10
        10.1.1.3: 100
  R3:
    links:
      Serial1/1:
        ip: 10.1.1.3 255.255.255.0
```

Page 60
passiveinterface: no
Serial1/0:
ip: 10.1.0.3 255.255.255.0
passiveinterface: no
Loopback0:
ip: 3.3.3.3 255.255.255.255
passiveinterface: yes
ospf:
rid: 3.3.3.3
BGP:
rid: 3.3.3.3
AS: 100
Network: ['10.1.1.0 mask 255.255.255.0', '10.1.0.0 mask 255.255.255.0']
Neighbors:
  10.1.0.1: 10
  10.1.1.2: 10

SW1:
links:
  FastEthernet1/0/1:
    VLAN: 10
    VoiceVLAN: 11
    Type: access
    Description: Access link to Internet
  FastEthernet1/0/2:
    VLAN: 20
    VoiceVLAN: 11
    Type: access
    Description: Access link to Something else
    PortSec: yes
    PortSecViolation: restrict
    PortSecDevice: mac-address sticky
    PortSecMaximum: 3
  FastEthernet1/0/3:
    VLAN: all
    Type: trunk
    Description: Trunk link to SW2
  FastEthernet1/0/4:
    VLAN: 10
    VoiceVLAN: 11
    Type: access
    Description: Access link to Something
    PortSec: yes
    PortSecViolation: shutdown
    PortSecDevice: mac-address sticky
    PortSecMaximum: 3
Vlan1:
ip: 192.168.101.91 255.255.255.0

SW2:
links:
  FastEthernet1/0/1:
    VLAN: 10
    VoiceVLAN: 11
    Type: access
Description: Access link to Internet

FastEthernet1/0/2:
VLAN: 20
VoiceVLAN: 11
Type: access
Description: Access link to Something else
PortSec: yes
PortSecViolation: restrict
PortSecDevice: mac-address sticky
PortSecMaximum: 3

FastEthernet1/0/3:
VLAN: all
Type: trunk
Description: Trunk link to SWI

FastEthernet1/0/4:
VLAN: 10
VoiceVLAN: 11
Type: access
Description: Access link to Something else
PortSec: yes
PortSecViolation: shutdown
PortSecDevice: mac-address sticky
PortSecMaximum: 3

Vlan1:
    ip: 192.168.101.92 255.255.255.0

VLAN:
    VLANNumber10:
        VLANID: 10
        VLANDescription: VLAN_Number_10
    VLANNumber20:
        VLANID: 20
        VLANDescription: VLAN_Number_20
    VLANNumber11:
        VLANID: 11
        VLANDescription: Voice_VLAN
    VLANNumber40:
        VLANID: 40
        VLANDescription: VLAN_Number_40_RSPAN
        RSPAN: True
    VLANNumber100:
        VLANID: 100
        VLANDescription: PVLAN_100_Primary
        PVLAN: True
        PVLANType: primary
    VLANNumber110:
        VLANID: 110
        VLANDescription: PVLAN_110_Isolated
        PVLAN: True
        PVLANType: isolated
    VLANNumber120:
        VLANID: 120
        VLANDescription: PVLAN_120_Community
        PVLAN: True
        PVLANType: community
A.3 Ansible ASA

Ansible scripts for opening/closing ASA’s that do not appear in the Ansible files section A.1:

The Ansible playbook `ASAFirewallOpening.yml`:

```yaml
---
- hosts: Firewalls
gather_facts: no
collection: local
vars_files:
  - ios.yml
  - firewall.yml
tasks:
  - name: Define connection
    set_fact:
      connection: authorize: yes
      username: "{{ ANSIBLE_NET_USERNAME }}"
      password: "{{ ANSIBLE_NET_PASS }}"
      auth_pass: "{{ ANSIBLE_NET_AUTH_PASS }}"

  - name: Generate removal of old Object and Deny statement for each device
    template:
      src: "./ASAJinjaFWRemove.j2"
      dest: "./configs/{{ inventory_hostname }}_FWRemove.cfg"

  - name: Generate new of Object and Deny statement for each device
    template:
      src: "./ASAJinjaFWObject.j2"
      dest: "./configs/{{ inventory_hostname }}_FWObject.cfg"

  - name: Generate the ACLs
    template:
      src: "./ASAJinjaACLCreation.j2"
      dest: "./configs/{{ inventory_hostname }}_FWACL.cfg"

  - name: get file content FWObject
    command: 'cat ./configs/{{ inventory_hostname }}_FWObject.cfg'
    register: FWObject

  - name: get file content FWRemove
    command: 'cat ./configs/{{ inventory_hostname }}_FWRemove.cfg'
    register: FWRemove

  - name: get file content FWACL
    command: 'cat ./configs/{{ inventory_hostname }}_FWACL.cfg'
    register: FWACL

  - name: Clearing the old
    asa_config:
      lines:
      - "{{ item }}"
      provider: "{{ connection }}"
```
ignore_errors: True
with_items: "{{FWRemove.stdout_lines}}"

# asa_config did not want to work directly with SRC and Parents functions in
# the asa_config module. Workaround was to read it into a variable and
# looping over the variable
- name: Pushing the new object and deny statement created using
  JinjaAndYAML
asa_config:
  lines:
  - "{{ item }}"
parents:
  - object-group service DenyObjectAnsible_1
    provider: "{{ connection }}"
  with_items: "{{FWObject.stdout_lines}}"

- name: Pushing the new ACL
  asa_config:
  lines:
  - "{{ item }}"
pplier: "{{ connection }}"
  with_items: "{{FWACL.stdout_lines}}"

Listing 88: Ansible playbook ASAFirewallOpening.yml

Jinja2 template for the clearing of old configuration ASAJinjaFWRemove.j2:
{% for ACL in SharedACLs.keys() %}
{% for Objects in SharedACLs[ACL].DenyObjects.keys()%}
o access-list {{ACL}} line 1 extended deny object-group {{Objects}} any
  any
no object-group service {{Objects}}
{% endfor %}
{% endfor %}

Listing 89: Jinja2 template for playbook ASAFirewallOpening.yml

Jinja2 template for the creating new Object-group ASAJinjaFWObject.j2:
{% for ACL in SharedACLs.keys() %}
{% for Objects in SharedACLs[ACL].DenyObjects.keys()%}
{% for srcDest in SharedACLs[ACL].DenyObjects[Objects][srcDest].keys()%}
{% if SharedACLs[ACL].DenyObjects[Objects][srcDest][Protocol] is not none %}
{% for ports in SharedACLs[ACL].DenyObjects[Objects][srcDest][Protocol][%]
service {{Protocol}} {{srcDest}} eq {{ports}}
{% endfor %}
{% endif %}
{% endfor %}
{% endfor %}

Listing 90: Jinja2 template for playbook ASAFirewallOpening.yml

Jinja2 template for the creating new ACL ASAJinjaACLCreation.j2:
{% for ACL in SharedACLs.keys() %}
Listing 91: Jinja2 template for playbook ASAFirewallOpening.yml

YAML Data file for the ASA including data about the ASA’s, firewall.yml:

Firewalls:
  ASA1:
    links:
      Ethernet0/0:
        vlan: 2
      Ethernet0/1:
        vlan: 1
      Vlan1:
        name: inside
        IP: 192.168.1.1
        SubnetMask: 255.255.255.0
        SecurityLevel: 100
      Vlan2:
        name: outside
        IP: 192.168.101.61
        SubnetMask: 255.255.255.0
        SecurityLevel: 0
    DefaultGateway: 192.168.101.1
    LocalNetwork: 192.168.1.0 255.255.255.0
    DHCPSubnet: 192.168.1.2 - 192.168.1.10
    ASAModel: 5505
    ID: 1

ASA2:
  links:
    GigabitEthernet1:
      name: inside
      IP: 192.168.2.1
      SubnetMask: 255.255.255.0
      SecurityLevel: 100
    GigabitEthernet0:
      name: outside
      IP: 192.168.101.62
      SubnetMask: 255.255.255.0
      SecurityLevel: 0
    DefaultGateway: 192.168.101.1
    LocalNetwork: 192.168.2.0 255.255.255.0
    DHCPSubnet: 192.168.2.2 - 192.168.2.10
    ASAModel: 5520
    ID: 2
ASA3:

  links:
  GigabitEthernet1:
    name: inside
    IP: 192.168.3.1
    SubnetMask: 255.255.255.0
    SecurityLevel: 100
  GigabitEthernet0:
    name: outside
    IP: 192.168.101.63
    SubnetMask: 255.255.255.0
    SecurityLevel: 0
  DefaultGateway: 192.168.101.1
  LocalNetwork: 192.168.3.0 255.255.255.0
  DHCPSubnet: 192.168.3.2 -192.168.3.10
  ASAModel: 5520
  ID: 3

ASA4:

  links:
  GigabitEthernet1:
    name: inside
    IP: 192.168.4.1
    SubnetMask: 255.255.255.0
    SecurityLevel: 100
  GigabitEthernet0:
    name: outside
    IP: 192.168.101.64
    SubnetMask: 255.255.255.0
    SecurityLevel: 0
  DefaultGateway: 192.168.101.1
  LocalNetwork: 192.168.4.0 255.255.255.0
  DHCPSubnet: 192.168.4.2 -192.168.4.10
  ASAModel: 5520
  ID: 4

ASA5:

  links:
  GigabitEthernet1:
    name: inside
    IP: 192.168.5.1
    SubnetMask: 255.255.255.0
    SecurityLevel: 100
  GigabitEthernet0:
    name: outside
    IP: 192.168.101.65
    SubnetMask: 255.255.255.0
    SecurityLevel: 0
  DefaultGateway: 192.168.101.1
  LocalNetwork: 192.168.5.0 255.255.255.0
  DHCPSubnet: 192.168.5.2 -192.168.5.10
  ASAModel: 5520
  ID: 5

VPNS2SKey: cisco123

SharedACLs:
  ACLAnsible1:
    DenyObjects:
DenyObjectAnsible_1:
  destination:
    tcp:
      - 811
    udp:
      - 722
  source:
    tcp:
    udp:

ACLLocation:
  links:
    inside:
    out:
      in: ACLAnsible1
    outside:
      out:
      in:

Listing 92: YAML data file for playbook ASAFirewallOpening.yml

A.4 Ansible Active

Ansible scripts for verifying connectivity with Cisco Switches, routers and ASA’s that do not appear in
the Ansible files section [A.1].

The Ansible playbook Active.yml:

---
- hosts: Routers:Switches
  gather_facts: no
  connection: local
  vars_files:
    - ios.yml
    - nodes.yml
  tasks:
    - name: Define connection
      set_fact:
        connection:
        authorize: yes
        username: "{{ ANSIBLE_NET_USERNAME }}"
        password: "{{ ANSIBLE_NET_PASS }}"
        auth_pass: "{{ ANSIBLE_NET_AUTH_PASS }}"

    - name: gathering IOS facts to verify activeness
      ios_facts:
        provider: "{{ connection }}"
      register: gatherInformation
      ignore_errors: True

    - name: Printing information about unreachable nodes
      debug:
        msg: "Failed on node {{inventory_hostname}} with IP address {{ ansible_ssh_host}} might be down, SSH access not working or no route to or from the node"
      when: gatherInformation is failed
hosts: Firewalls
gather_facts: no
collection: local
vars_files:
- ios.yml
- nodes.yml
tasks:
- name: Define connection
  set_fact:
    connection:
      authorize: yes
      username: "{{ ANSIBLE_NET_USERNAME }}"
      password: "{{ ANSIBLE_NET_PASS }}"
      auth_pass: "{{ ANSIBLE_NET_AUTH_PASS }}"

- name: gathering IOS facts to verify activeness
  asa_command:
    commands:
      - show version
        provider: "{{ connection }}"
      register: gatherInformation
      ignore_errors: True

- name: Printing information about unreachable nodes
  debug:
    msg: "Failed on node {{inventory_hostname}} with IP address {{ ansible_ssh_host }} might be down, SSH access not working or no route to or from the node"
when: gatherInformation is failed

Listing 93: Ansible playbook Active.yml

Output for running the Active.yml playbook:

markus@Borgen-kali:~$ ansible-playbook Active.yml

PLAY [Routers:Switches] ******************************************************

TASK [Define connection] ******************************************************
ok: [R1]
ok: [R2]
ok: [R3]
ok: [SW1]
ok: [SW2]

TASK [gathering IOS facts to verify activeness] ***********************
ok: [SW2]
ok: [R1]
ok: [SW1]
ok: [R3]
ok: [R2]

TASK [Printing information about unreachable nodes] *************
skipping: [R1]
skipping: [R2]
skipping: [R3]
skipping: [SW1]
skipping: [SW2]

PLAY [Firewalls] ****************************

TASK [Define connection] ****************************

ok: [ASA2]

TASK [gathering IOS facts to verify activeness] ****************************

fatal: [ASA2]: FAILED! => {
"changed": false, 
"msg": "unable to open shell. 
  Please see: https://docs.ansible.com/ansible/
  network_debug_troubleshooting.html#unable-to-open-shell"
}

...ignoring

TASK [Printing information about unreachable nodes] ****************************

ok: [ASA2] => {
"msg": "Failed on node ASA2 with IP address 192.168.101.62 might be 
  down, SSH access not working or no route to or from the node"
}

PLAY RECAP ****************************

ASA2           : ok=3  changed=0  unreachable=0  failed=0
R1             : ok=2  changed=0  unreachable=0  failed=0
R2             : ok=2  changed=0  unreachable=0  failed=0
R3             : ok=2  changed=0  unreachable=0  failed=0
SW1            : ok=2  changed=0  unreachable=0  failed=0
SW2            : ok=2  changed=0  unreachable=0  failed=0

Listing 94: Ansible playbook Active.yml output

A.5 Ansible audit

Ansible scripts for auditing Cisco Switches, routers and ASA’s that do not appear in the Ansible files section [A.1]

The Ansible playbook Audit.yml:

---
- hosts: Routers:Switches
gather_facts: no
collection: local
vars_files:
- ios.yml
- nodes.yml
tasks:
- name: Define connection
  set_fact:
    connection:
      authorize: yes
      username: "{{ ANSIBLE_NET_USERNAME }}"
      password: "{{ ANSIBLE_NET_PASS }}"
      auth_pass: "{{ ANSIBLE_NET_AUTH_PASS }}"
- name: Generate configs for each device

Page 69
#F or this to work with GNS3, please add more space on the disk0 disk under
withstanding configuration and reload device
#Then start scp server which ip scp server enable on a cisco device. Napalm
→ disk0: archive and lastly write−memory.

− name: Compare configuration with napalm
napalm_install_config:
  provider: "{{ connection }}"
  inline_transfer: True
  secret: "{{ ANSIBLE_NET_AUTH_PASS }}"
  dev_os: ios
  config_file: "/config/{{ inventory_hostname }}.cfg"
  commit_changes: False
  replace_config: True
  get_diffs: True
  diff_file: "/config/Difference_{{ inventory_hostname }}.cfg"

− hosts: Firewalls
  gather_facts: no
  connection: local
  vars_files:
  − ios.yml
  − firewall.yml
  tasks:
  − name: Define connection
    set_fact:
      connection:
        authorize: yes
        username: "{{ ANSIBLE_NET_USERNAME }}"
        password: "{{ ANSIBLE_NET_PASS }}"
        auth_pass: "{{ ANSIBLE_NET_AUTH_PASS }}"

− name: Generate configs for each ASA
  template:
    src: "/ASAJinja.j2"
    dest: "/config/{{ inventory_hostname }}_template.txt"
  register: templateconfig

− name: Gathering asa running config
  asa_command:
    commands: show running
    provider: "{{ connection }}"
  register: showconfig

− name: Save output to configs
  copy:
    content: "{{ showconfig.stdout[0] }}"
    dest: "/config/{{ inventory_hostname }}_showrunning.txt"

− name: Comparing showrunning and the generated template, some lines will
diff: due to passwords hiding in ASA

debug: msg="{{ lookup('pipe', '../compare.bash configs/{{
  inventory_hostname}}_showrunning.txt configs/{{inventory_hostname}}
  }}_template.txt > ../configs/Difference_{{inventory_hostname}}
  _TemplateToShowRun.txt '}}"

- name: Comparing the generated template with showrunning some defaults
diff: msg="{{ lookup('pipe', '../compare.bash configs/{{
  inventory_hostname}}}}_template.txt configs/{{inventory_hostname}}
  _showrunning.txt > ../configs/Difference_{{inventory_hostname}}
  _ShowrunToTemplate.txt '}}"

Listing 95: Ansible playbook Audit.yml

YAML data file for routers and switch configuration:

nodes:
  R1:
    links:
      Serial1/0:
        ip: 10.1.0.1 255.255.255.0
        passiveinterface: False
      FastEthernet0/0:
        ip: 192.168.101.11 255.255.255.0
        passiveinterface: False
      Loopback0:
        ip: 1.1.1.1 255.255.255.255
        passiveinterface: True
    ospf:
      rid: 1.1.1.1
    BGP:
      rid: 1.1.1.1
      AS: 10
      Network: [ '10.1.0.0 mask 255.255.255.0', '192.168.101.0' ]
      Neighbors:
        192.168.101.12: 10
        10.1.0.3: 100

  R2:
    links:
      Serial1/1:
        ip: 10.1.1.2 255.255.255.0
        passiveinterface: no
      FastEthernet0/0:
        ip: 192.168.101.12 255.255.255.0
        passiveinterface: no
      Loopback0:
        ip: 2.2.2.2 255.255.255.255
        passiveinterface: yes
    ospf:
      rid: 2.2.2.2
    BGP:
      rid: 2.2.2.2
      AS: 10
      Network: [ '10.1.1.0 mask 255.255.255.0', '192.168.101.0' ]
      Neighbors:
        192.168.101.11: 10
10.1.1.3: 100

R3:
  links:
   Serial1/1:
     ip: 10.1.1.3 255.255.255.0
     passiveinterface: no
   Serial1/0:
     ip: 10.1.0.3 255.255.255.0
     passiveinterface: no
   Loopback0:
     ip: 3.3.3.3 255.255.255.255
     passiveinterface: yes

ospf:
   rid: 3.3.3.3

BGP:
   rid: 3.3.3.3
   as: 100
   Network: [10.1.1.0 mask 255.255.255.0', '10.1.0.0 mask 255.255.255.0'
   Neighbors:
     10.1.0.1: 10
     10.1.1.2: 10

SW1:
  links:
   FastEthernet1/0/1:
     VLAN: 10
     VoiceVLAN: 11
     Type: access
     Description: Access link to Internet
   FastEthernet1/0/2:
     VLAN: 20
     VoiceVLAN: 11
     Type: access
     Description: Access link to Something else
     PortSec: yes
     PortSecViolation: restrict
     PortSecDevice: mac-address sticky
     PortSecMaximum: 3
   FastEthernet1/0/3:
     VLAN: all
     Type: trunk
     Description: Trunk link to SW2
   FastEthernet1/0/4:
     VLAN: 10
     VoiceVLAN: 11
     Type: access
     Description: Access link to Something
     PortSec: yes
     PortSecViolation: shutdown
     PortSecDevice: mac-address sticky
     PortSecMaximum: 3

Vlan10:
  ip: 192.168.101.91 255.255.255.0

SW2:
links :
  FastEthernet1/0/1:
    VLAN: 10
    VoiceVLAN: 11
    Type: access
    Description: Access link to Internet
  FastEthernet1/0/2:
    VLAN: 20
    VoiceVLAN: 11
    Type: access
    Description: Access link to Something else
    PortSec: yes
    PortSecViolation: restrict
    PortSecDevice: mac-address sticky
    PortSecMaximum: 3
  FastEthernet1/0/3:
    VLAN: all
    Type: trunk
    Description: Trunk link to SW1
  FastEthernet1/0/4:
    VLAN: 10
    VoiceVLAN: 11
    Type: access
    Description: Access link to Something
    PortSec: yes
    PortSecViolation: shutdown
    PortSecDevice: mac-address sticky
    PortSecMaximum: 3

Vlan10:
  ip: 192.168.101.92 255.255.255.0

VLAN:
  VLANNumber10:
    VLANNOT: 10
    VLANTDescription: VLAN_Number_10
  VLANNumber20:
    VLANNOT: 20
    VLANTDescription: VLAN_Number_20
  VLANNumber11:
    VLANNOT: 11
    VLANTDescription: Voice_VLAN
  VLANNumber40:
    VLANNOT: 40
    VLANTDescription: VLAN_Number_40_RSPAN
    RSPAN: True
  VLANNumber100:
    VLANNOT: 100
    VLANTDescription: PVLAN_100_Primary
    PVLAN: True
    PVLANType: primary
  VLANNumber110:
    VLANNOT: 110
    VLANTDescription: PVLAN_110_Isolated
    PVLAN: True
    PVLANType: isolated
  VLANNumber120:
Listing 96: YAML data file for playbook Audit.yml

First Jinja2 file for Cisco routers and Switches:

```yaml
{% if group_names[0] == "Routers" %}
{% include "JinjaRouters.j2" %}
{% elif group_names[0] == "Switches" %}
{% include "JinjaSwitches.j2" %}
{% else %}
{{ group_names[0] }}
{% endif %}
```

Listing 97: Jinja2 template for playbook Audit.yml

Specific Jinja2 file for Cisco routers:

```yaml
{% for node in nodes.keys() %}
{% if inventory_hostname == node %}
hostname {{ inventory_hostname }}
ip domain name AnsibleLab
ip ssh version 2
ip scp server enable
username admin secret 5 $1$wyBg$lQ.m.P7vtS2$X0KSB.16Q/
enable secret 5 $1$ByiX$NBrdOQxg8MuZayC2kIkv.
aaa new-model
aaa authentication login default local enable
aaa authentication login CONSOLE none
line con 0
    login authentication CONSOLE
    stopbits 1
    line aux 0
    stopbits 1
    line vty 0 4
ip cef
no ipv6 cef
{% for link in nodes[node].links.keys() %}
interface {{ link }}
ip address {{ nodes[node].links[link].ip }}
no shutdown
{% endfor %}

service timestamps debug datetime msec
service timestamps log datetime msec
boot-start-marker
boot-end-marker
aaa session-id common
multilink bundle-name authenticated
ip forward-protocol nd
no ip http server
no ip http secure-server
control-plane

router bgp {{ nodes[node].BGP.AS }}
```
bgp log-neighbor-changes
bgp router-id { {nodes[node].BGP.rid} }
{% for networks in nodes[node].BGP.Network %}
    network { {networks} }
{% endfor %}
{% for neighborsIP in nodes[node].BGP.Neighbors.keys() %}
    neighbor { {neighborsIP} } remote-as { {nodes[node].BGP.Neighbors[neighborsIP] - 1} }
{% endfor %}

ip route 192.168.1.0 255.255.255.0 192.168.101.61
{% endif %}
{% endfor %}

Listing 98: Jinja2 template for playbook Audit.yml

Specific Jinja2 file for Cisco switches:

ttp mode transparent
{% for VLANs in VLAN.keys() %}
    vlan { {VLAN[VLANs].VLANID} }
    name { {VLAN[VLANs].VLANDescription} }
    {% if VLAN[VLANs].RSPAN is defined %}
        remote-span
    {% endif %}
    {% endif %}
{% endfor %}

{% for VLANs in VLAN.keys() %}
    {% if VLAN[VLANs].PVLAN is defined %}
        private-vlan { {VLAN[VLANs].PVLANType} }
    {% endif %}
{% endfor %}

{% for VLANs in VLAN.keys() %}
    {% if VLAN[VLANs].PVLAN is defined %}
        {% if VLAN[VLANs].PVLANType == "primary" %}
            vlan { {VLAN[VLANs].VLANID} }
        {% endif %}
    {% endif %}
{% endfor %}

{% set ListOfPVLAN = {"VLAN": ""} %}
{% for VLANs in VLAN.keys() %}
    {% if VLAN[VLANs].PVLAN is defined %}
        {% if VLAN[VLANs].PVLANType == "primary" %}
            private-vlan association { {ListOfPVLAN.VLAN: -1} }
        {% endif %}
    {% endif %}
{% endfor %}
{% for node in nodes.keys() %}
{% if inventory_hostname == node %}
hostname {{inventory_hostname}}
ip domain name AnsibleLab
ip ssh version 2
ip scp server enable
username admin secret 5 $1$wyBg$LQ.m.P7vtS2SX0KSB.16Q/
enable secret 5 $1$ByiX$NBrdOQxg8MuZayC2kIklv.
aaa new-model
aaa authentication login default local enable
aaa authentication login CONSOLE none
line con 0
  login authentication CONSOLE
  stopbits 1
line vty 0 4
ip classless
ip http server
ip http secure-server
ip sla enable reaction-alerts
line vty 5 15

no service pad
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
boot-start-marker
boot-end-marker
aaa session-id common

archive
  path flash:archive
  write-memory

ip routing
ip route 0.0.0.0 0.0.0.0 192.168.101.1

{% for link in nodes[node].links.keys() %}
interface {{link}}
{% if nodes[node].links[link].Type is defined %}
{% endif %}

{% if nodes[node].links[link].Type == "trunk" %}
  switchport trunk encapsulation dot1q
  switchport mode trunk
  switchport nonegotiate
  switchport trunk native vlan 999
  switchport trunk allowed vlan {{nodes[node].links[link].VLAN}}
  description {{nodes[node].links[link].Description}}
{% endif %}

{% if nodes[node].links[link].Type == "access" %}
  switchport mode access
  switchport nonegotiate
  switchport access vlan {{nodes[node].links[link].VLAN}}
  switchport voice vlan {{nodes[node].links[link].VoiceVLAN}}
{% endif %}
{% endfor %}

{% endfor %}
description {{nodes[node].links[link].Description}}
{% if nodes[node].links[link].PortSec is defined%}
{% if nodes[node].links[link].PortSec != True%}
switch port security violation {{nodes[node].links[link].PortSecViolation}}
switch port security maximum {{nodes[node].links[link].PortSecMaximum}}
switch port security {{nodes[node].links[link].PortSecDevice}}
switch port security
{% endif%}
{% endif%}
{% endif%}
{% endif%}
{% if nodes[node].links[link].ip is defined%}
ip address {{nodes[node].links[link].ip}}
no shutdown
{% endif%}
ip route 192.168.1.0 255.255.255.0 192.168.101.61
{% endif %}
{% endif %}
{% endif %}

Listing 99: Jinja2 template for playbook Audit.yml

ASA YAML data file firewall.yml:

Firewalls:
ASA1:
  links:
    Ethernet0/0:
      vlan: 2
    Ethernet0/1:
      vlan: 1
  Vlan1:
    name: inside
    IP: 192.168.1.1
    SubnetMask: 255.255.255.0
    Securitylevel: 100
  Vlan2:
    name: outside
    IP: 192.168.101.61
    SubnetMask: 255.255.255.0
    Securitylevel: 0
    DefaultGateway: 192.168.101.1
    LocalNetwork: 192.168.1.0 255.255.255.0
    DHCPSubnet: 192.168.1.2 – 192.168.1.10
    ASAModel: 5505
    ID: 1

ASA2:
  links:
    GigabitEthernet1:
      name: inside
      IP: 192.168.2.1
SubnetMask: 255.255.255.0
Security level: 100
GigabitEthernet0:
  name: outside
  IP: 192.168.101.62
  SubnetMask: 255.255.255.0
  Security level: 0
Default Gateway: 192.168.101.1
Local Network: 192.168.2.0 255.255.255.0
DHCP Subnet: 192.168.2.2 − 192.168.2.10
ASAModel: 5520
ID: 2
ASA3:
  links:
    GigabitEthernet1:
      name: inside
      IP: 192.168.3.1
      SubnetMask: 255.255.255.0
      Security level: 100
GigabitEthernet0:
  name: outside
  IP: 192.168.101.63
  SubnetMask: 255.255.255.0
  Security level: 0
Default Gateway: 192.168.101.1
Local Network: 192.168.3.0 255.255.255.0
DHCP Subnet: 192.168.3.2 − 192.168.3.10
ASAModel: 5520
ID: 3
ASA4:
  links:
    GigabitEthernet1:
      name: inside
      IP: 192.168.4.1
      SubnetMask: 255.255.255.0
      Security level: 100
GigabitEthernet0:
  name: outside
  IP: 192.168.101.64
  SubnetMask: 255.255.255.0
  Security level: 0
Default Gateway: 192.168.101.1
Local Network: 192.168.4.0 255.255.255.0
DHCP Subnet: 192.168.4.2 − 192.168.4.10
ASAModel: 5520
ID: 4
ASA5:
  links:
    GigabitEthernet1:
      name: inside
      IP: 192.168.5.1
      SubnetMask: 255.255.255.0
      Security level: 100
GigabitEthernet0:
  name: outside
  IP: 192.168.101.65
  SubnetMask: 255.255.255.0
Security level: 0
Default Gateway: 192.168.101.1
Local Network: 192.168.5.0 255.255.255.0
DHCP Subnet: 192.168.5.2–192.168.5.10
ASA Model: 5520
ID: 5

VPNS2SKey: cisco123

Shared ACLs:
ACL Ansible1:
  Deny Objects:
    Deny Object Ansible_1:
      destination:
        tcp:
          - 81
          - 71
        udp:
          - 82
          - 72
    source:
      tcp:
      udp:

ACL Location:
links:
  inside:
  out:
    in: ACL Ansible1
  outside:
    out:
    in:

Listing 100: YAML data file for playbook Audit.yml

Specific Jinja2 file for Cisco ASA:

{% for node in Firewalls.keys() %}
{% if inventory_hostname==node %}
hostname {{node}}
domain–name AnsibleLab
enable password 9jNFZuG3TC5tCVH0 encrypted
username admin password e1z89R3cZeKt6Ib encrypted

{% for link in Firewalls[node].links.keys() %}
{% if Firewalls[node].links[link].vlan is defined %}
interface {{link}}
{% else %}
nameif {{Firewalls[node].links[link].name}}
security–level {{Firewalls[node].links[link].SecurityLevel}}
ip address {{Firewalls[node].links[link].IP}}
{% endif %}

{% endif %}
no shutdown
dns domain−lookup inside
dns domain−lookup outside
dns server−group DefaultDNS
  name−server {{Firewalls[node].DefaultGateway}}
  domain−name AnsibleLab

route outside 0.0.0.0 0.0.0.0 {{Firewalls[node].DefaultGateway}} 1

crypto ipsec security−association lifetime seconds 86400
crypto ipsec security−association pmtu−aging infinite

{% for DestNode in Firewalls.keys() %}
{% if DestNode != node %}
{% if Firewalls[DestNode].links.keys() %}
{% for link in Firewalls[DestNode].links.keys() %}
  {% if Firewalls[DestNode].links[link].name is defined and Firewalls[DestNode].links[link].name=="outside" %}

  access−list SITETOSTEACL{{node}}to{{DestNode}} extended permit ip {{
    Firewalls[node].LocalNetwork} {{Firewalls[DestNode].LocalNetwork}}
  access−list SITETOSTEACL{{node}}to{{DestNode}} extended deny ip any any

route outside {{Firewalls[DestNode].LocalNetwork}} {{Firewalls[DestNode].links[link].IP}}

  crypto ipsec ikev1 transform−set SITETOSTEACL{{node}}to{{DestNode}} esp−aes
  → −256 esp−sha−hmac
  crypto map S2VPN {{Firewalls[node].ID}}{{Firewalls[DestNode].ID}} match
  → address SITETOSTEACL{{node}}to{{DestNode}}
  crypto map S2VPN {{Firewalls[node].ID}}{{Firewalls[DestNode].ID}} set pfs
  → group5
  crypto map S2VPN {{Firewalls[node].ID}}{{Firewalls[DestNode].ID}} set peer
  → {{Firewalls[DestNode].links[link].IP}}
  crypto map S2VPN {{Firewalls[node].ID}}{{Firewalls[DestNode].ID}} set
  → ikev1 transform−set SITETOSTEACL{{node}}to{{DestNode}}
  crypto map S2VPN interface outside

tunnel−group {{Firewalls[DestNode].links[link].IP}} type ipsec−121
  crypto−attributes
    ikev1 pre−shared−key {{VPNS2SKey}}

{% endif %}
{% endfor %}
{% endif %}
{% endfor %}

crypto ca trustpool policy
crypto ikev1 enable outside
crypto ikev1 policy 1
  authentication pre-share
  encryption aes
  hash sha
  group 5
  lifetime 86400

dhcpd address {{Firewalls[node].DHCPSubnet}} inside
 dhcpd enable inside

ntp server 82.102.5.110 source outside
 ssh stricthostkeycheck
 ssh 192.168.1.0 255.255.255.0 inside
 ssh 192.168.1.0 255.255.255.0 outside
 ssh 192.168.101.0 255.255.255.0 outside
 ssh timeout 10
 ssh version 2
 ssh key-exchange group dh-group14-sha1
 console timeout 0
 same-security-traffic permit inter-interface
 same-security-traffic permit intra-interface
 aaa authentication ssh console LOCAL
 http server enable
 http 192.168.1.0 255.255.255.0 inside
 http 192.168.1.0 255.255.255.0 outside

class-map inspection_default
  match default-inspection-traffic
 policy-map type inspect dns preset_dns_map
 parameters
  message-length maximum client auto
  message-length maximum 512
 policy-map global_policy
 class inspection_default
  inspect dns preset_dns_map
  inspect ftp
  inspect h323 h225
  inspect h323 ras
  inspect ip-options
  inspect netbios
  inspect rsh
  inspect rtsp
  inspect skinny
  inspect esmtp
  inspect smtp
  inspect sqlnet
  inspect sunrpc
  inspect tftp
  inspect sip
  inspect xdmcp
  inspect icmp
!
 service-policy global_policy global

{% endif %}
Comparing the template to the configuration on the ASA compare.bash:

```bash
grep -vxFf $1 $2 | sed -r '/^\s*#/d;'  
```

Comparing R1 result:

```
+username admin secret 5 $1$wyBg$LQ.m.P7vtS2SX0KSBD16Q/
+enable secret 5 $1$ByiX$NBrdOQXg8MzayC2kklv.
+no shutdown
+no shutdown
+no shutdown

-username admin secret 5 $1$Dxr.$d2LToi8FoQwrolyImdvEd.
-no ip domain lookup
-username admin secret 5 $1$rczZ$OGZa5GH5cHghOISVcawQ80
-ip tcp synwait-time 5
interface FastEthernet0/0
-duplex full
interface Serial1/0
-serial restart-delay 0
-interface Serial1/1
-no ip address
-shutdown
-serial restart-delay 0
-interface Serial1/2
-no ip address
-shutdown
-serial restart-delay 0
-interface Serial1/3
-no ip address
-shutdown
-serial restart-delay 0

line con 0
-exec-timeout 0 0
-privilege level 15
-logging synchronous
line aux 0
-exec-timeout 0 0
-privilege level 15
-logging synchronous
-end
```

Comparing SW1 result:

```
!+
+vlan 100
+private-vlan association 110,120
interface Vlan1
+no shutdown
interface FastEthernet1/0/3
+switchport trunk allowed vlan all
interface FastEthernet1/0/4
+switchport port-security violation shutdown
```
switch provision ws-c3750-48p
system mtu routing 1500
authentication mac-move permit
ip subnet-zero
crypt pki trustpoint TP-self-signed -2147138432
enrollment selfsigned
subject-name cn=IOS Self-Signed-Certificate -2147138432
revocation check none
rsakeypair TP-self-signed -2147138432
crypt pki certificate chain TP-self-signed -2147138432
certificate self-signed 01
-30820246 308201AF A0030201 02020101 300D0609 2A864886 F70D0101 04050030
-31312F30 2D060355 04031326 49F532D 53656C66 2D536967 6E56642D 43657274
-69666963 6174652D 32313437 31333834 3332301E 170D3933 30333031 30303034
-30305417 0D323030 31303130 30303030 305A3031 312F302D 06035504 03132649
-4F532D53 656C626D 5369676E 65642D43 67727469 66696361 74652D32 31343731
-33383433 3230381F 300D0609 2A864886 F70D0101 01050003 818D0030 81890281
-8100C315 B2C96B31 389DBA0C 7A5CEDDB 43815214 EBB16851 3712DAE6 618EA551
-8691E7EF BB34B668 F2DF9684 9E35CD91 6148B188 OF0F0139 4E28C234 85BSEC55
-13992282 D8DE0CA7 58762319 7C5F7AAA AC8D62A3 B33CC7A9 4C4CD4C9 9C12D830
-FFAF8D85 5BFDD854 09FD027B D8A5EA51 235E5C4C 57D9177E 1346E070 5F82B4E5
-28210203 010001A3 6E306C30 0F060355 1D130101 FF040530 030101FF 30190603
-551D1104 12301082 0E535731 2E416E73 69626C65 4C616230 1F060355 1D230418
-30168014 B919835F 77AE183F E9742E72 4D969048 ED7C78B1 301D0003 551DE0E4
-160414B9 19835F77 AE183F6E 742E724D 969048ED 7C8B1300 000092A 864866F7
-0D010104 05000381 8100342F 7F88A16E D8DA41AD 21B54640 9113A6B0 CDBC3E5
-79335D18 45741789 49E50EAD 43EC2FE6 CCDCE837 A51AAB67 49DC14DE E5BC4A55
-2990BFC7 D6FB72A3 D9965D52E 99F64276 A6F85376 8D3DFFE1B A578628B 5AC9CD4
-977A71CD 7718404C 2F4A83CC 4504C33E E18078E4 E3E4AE4A B59C1E4E 76A10416
-310B3128 AA54215E 3661
quit
spanning-tree mode pvst
spanning-tree etherchannel guard misconfig
spanning-tree extend system-id
vlan internal allocation policy ascending
vlan 100
name PVLAN_100_Primary
private-vlan association 110,120
interface FastEthernet1/0/1
spanning-tree portfast
interface FastEthernet1/0/2
spanning-tree portfast
interface FastEthernet1/0/4
spanning-tree portfast
interface FastEthernet1/0/5
interface FastEthernet1/0/6
interface FastEthernet1/0/7
interface FastEthernet1/0/8
interface FastEthernet1/0/9
interface FastEthernet1/0/10
interface FastEthernet1/0/11
interface FastEthernet1/0/12
interface FastEthernet1/0/13
interface FastEthernet1/0/14
interface FastEthernet1/0/15
interface FastEthernet1/0/16
interface FastEthernet1/0/17
Comparing show running to Template ASA1:

ASA Version 9.1(5)
xlate per-session deny tcp any4 any4
xlate per-session deny tcp any4 any6
xlate per-session deny tcp any6 any4
xlate per-session deny tcp any6 any6
xlate per-session deny udp any4 any4 eq domain
xlate per-session deny udp any4 any6 eq domain
xlate per-session deny udp any6 any4 eq domain
xlate per-session deny udp any6 any6 eq domain
names
interface Ethernet0/2
  shutdown
interface Ethernet0/3
  shutdown
interface Ethernet0/4
shutdown
interface Ethernet0/5
shutdown
interface Ethernet0/6
shutdown
interface Ethernet0/7
shutdown
ftp mode passive
pager lines 24
mtu outside 1500
mtu inside 1500
icmtp unreachable rate-limit 1 burst-size 1
no asdm history enable
arp timeout 14400
no arp permit-nonconnected
timeout xlate 3:00:00
timeout pat-xlate 0:00:30
timeout conn 1:00:00 half-closed 0:10:00 udp 0:02:00 icmp 0:00:02
timeout sunrpc 0:10:00 h323 0:05:00 h225 1:00:00 mgcp 0:05:00 mgcp-pat
  ↪  0:05:00
timeout sip 0:30:00 sip_media 0:02:00 sip-invite 0:03:00 sip-disconnect
  ↪  0:02:00
timeout sip-provisional-media 0:02:00 uauth 0:05:00 absolute
timeout tcp-proxy-reassembly 0:01:00
timeout floating-conn 0:00:00
dynamic-access-policy-record DfltAccessPolicy
user-identity default-domain LOCAL
no snmp-server location
no snmp-server contact
snmp-server enable traps snmp authentication linkup linkdown coldstart
  ↪ warmstart
crypto ikev1 policy 65535
  encryption 3des
group 2
telnet timeout 5
threat-detection statistics access-list
no threat-detection statistics tcp-intercept
ikev1 pre-shared-key *****
ikev1 pre-shared-key *****
ikev1 pre-shared-key *****
ikev1 pre-shared-key *****
prompt hostname context
no call-home reporting anonymous
call-home
profile CiscoTAC-1
  no active
destination address http https://tools.cisco.com/its/service/oddce/
  ↪ services/DDCEService
destination address email callhome@cisco.com
destination transport-method http
subscribe-to-alert-group diagnostic
subscribe-to-alert-group environment
subscribe-to-alert-group inventory periodic monthly
subscribe-to-alert-group configuration periodic monthly
subscribe-to-alert-group telemetry periodic daily
Cryptochecksum:95df0b7cfde214e030c47b7fa4b2836
: end
Comparing Template to show running ASA1:

switchport access vlan 1
no shutdown
no shutdown
no shutdown
no shutdown
ikev1 pre-shared-key cisco123
ikev1 pre-shared-key cisco123
ikev1 pre-shared-key cisco123
ikev1 pre-shared-key cisco123

A.6 Ansible with Python Layer2 diagram

Ansible and Python scripts for creating a layer 2 diagram using CDP that do not appear in the Ansible files section A.1.

The Ansible playbook CDPDiagram.yml:

Listing 105: Output Ansible Audit.yml ASA comparing show running to template

Listing 106: Output Ansible Audit.yml ASA comparing template to show running

Listing 107: Ansible playbook CDPDiagram.yml

Python script for using the data saved from Ansible to create a layer 2 diagram, PythonDiagram.py:

```python
#!/usr/bin/python
```
# Written by Markus Borgenstrand, MABO1602.

# Importing the os and graphviz libraries. OS is for reading the CDP information files and later removing them. Graphviz is for the visual topology.

import os
import graphviz

# Exception to check if there even are any files in the directory.
try:
    ListOfFileNames=os.listdir('./CDP')
except:
    ListOfFileNames=""

NodesConnected={}

i=0
for FileName in ListOfFileNames:
    # Split() can split up a string onto different words, if an argument is used it can split on that instead of space which is standard.
    # Because Ansible creates the files as Node_show_CDP.txt it will split up the Node show CDP.txt into different elements in a list. Only the first holds the node name.
    CurrentNode=FileName.split('_')[0]+'/'+AnsibleLab"

    # Opens the file and reads all the lines in the file.
    CurrentFile=open("CDP/"+FileName,"rt")
    CurrentFileLines=CurrentFile.readlines()

    # The reason why it starts at j=1 is that the first line is not interesting as it holds this information:
    # Device ID  Local Interface  Holdtime  Capability  Platform  Port ID
    # It is good to have for trouble shooting purposes but not needed in the topology.
    j=1

    # This loop loops through all the neighbours in each Node_show_CDP.txt file and adds them to the list, incase there are multiple by the same interface it will add another entry in the dictionary, this is due to the fact that a device might be connected to multiple devices through one link.
    while j<len(CurrentFileLines):
        CurrentLineTemp=CurrentFileLines[j].replace('\n', '').split(" ")
        for element in CurrentLineTemp:
            if element != '':
                CurrentLine.append(element)

        j+=1

    try:
        NodesConnected[CurrentNode]=CurrentNode,CurrentLine[1],CurrentLine[0],CurrentLine[-1]]
    except:
        NodesConnected[CurrentNode]=CurrentNode,CurrentLine[1],CurrentLine[0],CurrentLine[-1]]

    CurrentFile.close()
Nodes Connected now holds all the data about the connections. The structure
is Local device, Local interface, Remote device, remote interface.

Now to remove the loaded files (I like to be modular so each section does
something and then it is easier to test then just throw everything in
at once)

One reason for this to be done is so that old configurations files will
not be added in the graph incase of the node being down.

Scenario: If R1 is connected to R2 who is connected to R3 who is connected
to R1, if R3 goes down the link is only R1 to R2, but if the files
are not removed Ansible will not remove them (only update R1 and R2
file) and then R3 will magically be shown in the graph anyway. This
can of course be commented out for troubleshooting purposes.

for FileName in ListOfFileNames:
    os.system("rm CDP/"+FileName)

Nodes Keys = sorted (Nodes Connected . keys () )
edges ={}

# print Nodes Keys
for Nodes in Nodes Keys:
    i=0
    while i<len (Nodes Connected [ Nodes ]):
        # Need a bit of magic to show the link names correctly as sometimes it takes
        # a space in the name from the show cdp neighbour file.
        # First splitting up the string by seperating everythign with a space
        # between.
        # Then emptying the empty elements and only keeping the elements with
        # something in it. Doing this for the Local and Remote link.
        # The reason for this is that the tuples ('R1.AnsibleLab', 'Ser 1/2') and ('
        # R1.AnsibleLab', 'Ser 1/2') are different due to the "Ser 1/2" and "
        # Ser 1/2" being different.
        Local Link = Nodes Connected [ Nodes ][i+1] . split (" ")
        # print ("Local Link Before: "+str (Local Link))
        Local Link [:] = [ item for item in Local Link if item != '\n']
        Local Link = str (Local Link [0] + Local Link [1])
        Remote Link = Nodes Connected [ Nodes ][i+3] . split (" ")
        # print ("Remote Link Before: "+str (Remote Link))
        Remote Link [:] = [ item for item in Remote Link if item != '\n' and item != "WS-"
        # C3750-"]# This is to avoid issues with C3750 taking up a lot of
        # space in show cdp neighbor output
        Remote Link = str (Remote Link [0] + Remote Link [1])
        # print ("Local Link: "+Local Link)
        # print ("Remote Link: "+Remote Link)

        # The Local variable holds the current Local node and its interface in a
        # tuple, the Remote holds the remote node and its interface. This will
        # then be compared against the edges dictionary so duplicate nodes and
        # links will not show up in the graph. Edges will otherwise show both
        # link from R1 to R3 and the same link from R3 to R1 even though it is
        # one and the same.
        Local = (Nodes Connected [ Nodes ][i] . Local Link)
        Remote = (Nodes Connected [ Nodes ][i+2] . Remote Link)
        print ("Local: "+str (Local))
        print ("Remote: "+str (Remote))

        if (Local in edges . keys () and Remote in edges . values () ):
pass
#Below print statement is kept for troubleshooting purposes
#print("Duplicated local in keys remote in values")
elif(Remote in edges.keys() and Local in edges.values()):
pass
#Below print statement is kept for troubleshooting purposes
#print("Duplicated Remote in keys Local in values")
else:
edges[Local]=Remote
i+=4

#Below is only for troubleshooting if needed or if we need data for
→ manually creating the topology using CDP as the data source.

print(edges)
print(edges.keys())
print(edges.values())

#Below code was copied from https://gist.github.com/floatingstatic/598
↪ f5258cd7fa554af785aa7dec4417d but changed a bit for my project.

def draw_topology(topology, output_filename):
    style = {
        'graph': {
            'label': output_filename,
            'fontsize': '16",
            'fontcolor': 'white',
            'bgcolor': '#333333',
            'rankdir': 'BT',
        },
        'nodes': {
            'fontname': 'Helvetica',
            'shape': 'box',
            'fontcolor': 'white',
            'color': '#006699',
            'style': 'filled',
            'fillcolor': '#006699',
            'margin': '0.4',
        },
        'edges': {
            'style': 'dashed',
            'color': 'green',
            'arrowhead': 'open',
            'fontname': 'Courier',
            'fontsize': '10',
            'fontcolor': 'white',
        }
    }

    nodes = set([key[0] for key in topology.keys() + topology.values()])

g = graphviz.Graph(format='png')

for node in nodes:
g.node(node)

for key, value in topology.items():
    g.edge(key, value)
head, t_label = key
tail, h_label = value
g.edge(head, tail, headlabel=h_label, taillabel=t_label, label="↪")

接纳

↑

∗12)

g.graph_attr.update(
    ('graph' in style and style['graph']) or {})
g.node_attr.update(
    ('nodes' in style and style['nodes']) or {})
g.edge_attr.update(
    ('edges' in style and style['edges']) or {})
g.render(filename=output_filename)

draw_topology(edges, 'Network diagram')

Listing 108: Python script for creating network diagrams

A.7 Ansible with Python Layer2 diagram with STP

Ansible and Python scripts for creating a layer 2 diagram using CDP and STP that do not appear in the Ansible files section A.1.

The Ansible playbook CDPDiagramWithSTP.yml:

```yaml
---
- hosts: Routers
  gather_facts: no
  connection: local
  vars_files:
  - ios.yml
  - nodes.yml
  tasks:
  - name: Define connection
    set_fact:
      connection:
      authorize: yes
      username: "{{ ANSIBLE_NET_USERNAME }}"
      password: "{{ ANSIBLE_NET_PASS }}"
      auth_pass: "{{ ANSIBLE_NET_AUTH_PASS }}"

  - name: Setting up CDP data file for Diagram usage
    ios_command:
      commands: show cdp neighbors | begin Device ID
      provider: "{{ connection }}"
      register: showinformation
  - name: Save output to local folder for python script
    copy:
```
content: "{{showinformation.stdout[0]}}"
dest: "CDP/{{inventory_hostname}}_show_CDP.txt"

- hosts: Switches
gather_facts: no
collection: local
vars_files:
- ios.yml
- nodes.yml
tasks:
- name: Define connection
  set_fact:
    connection:
      authorize: yes
      username: "{{ ANSIBLE_NET_USERNAME }}"
      password: "{{ ANSIBLE_NET_PASS }}"
      auth_pass: "{{ ANSIBLE_NET_AUTH_PASS }}"

- name: Setting up CDP data file for Diagram usage
  ios_command:
    commands: show cdp neighbors | begin Device ID
    provider: "{{ connection }}"
    register: showinformation
- name: Save output to local folder for pythonscript
  copy:
    content: "{{showinformation.stdout[0]}}"
    dest: "CDP/{{inventory_hostname}}_show_CDP.txt"

- name: Setting up STP data file for Diagram usage
  ios_command:
    commands: show spanning-tree blockedports | include VLAN
    provider: "{{ connection }}"
    register: showSTP
    ignore_errors: true
- name: Save output to local folder for pythonscript
  copy:
    content: "{{showSTP.stdout[0]}}"
    dest: "STP/{{inventory_hostname}}_show_STP.txt"

- name: Setting up STP root data file for Diagram usage
  ios_command:
    commands: show spanning-tree | include VLAN|This bridge is the root
    provider: "{{ connection }}"
    register: showSTPRoot
    ignore_errors: true
- name: Save output to local folder for pythonscript
  copy:
    content: "{{showSTPRoot.stdout[0]}}"
    dest: "STPRoot/{{inventory_hostname}}_show_STP_Root.txt"

- name: Running script to create a network diagram
  debug: msg="{{ lookup('pipe', 'l./PythonDiagramWithSpanningTree.py') }}"
Python script for using the data saved from Ansible to create a layer 2 diagram, PythonDiagramWithSpanningTree.py:

```python
#!/usr/bin/python
#Written by Markus Borgenstrand, MAB01602.
#Importing the os and graphviz libraries. OS is for reading the CDP
#information files and later removing them. Graphviz is for the visual
topology.
#This script creates topologies as CDP sees it with links blocked by STP
#removed. If no ports are blocked for a VLAN the topology is the same
#as the Physical topology, this is to avoid creating a topology for
#each VLAN when all are the same (Like for VLAN 1002-1005 that is not
#really used anymore). The STP root for the VLAN is shown for the VLAN
topologies.
import os
import graphviz

#Exception to check if there even are any files in the directory.
try:
    ListOfFileNames=os.listdir('./CDP')
except:
    ListOfFileNames=""

NodesConnected={}

i=0
for FileName in ListOfFileNames:
    #Split() can split up a string onto different words, if an argument is
    #used it can split on that instead of space which is standard.
    #Because Ansible creates the files as Node_show_CDP.txt it will split up
    #the Node show CDP.txt into different elements in a list. Only the
    #first holds the node name.
    CurrentNode=FileName.split('_') [0] +".AnsibleLab"
    #Opens the file and reads all the lines in the file.
    CurrentFile=open("CDP/"+FileName,"rt")
    CurrentFileLines=CurrentFile.readlines()
    #The reason why it starts at j=1 is that the first line is not interesting
    #as it holds this information:
    #Device ID Local InTrfce Holdtme Capability Platform Port
    #ID
    #It is good to have for trouble shooting purposes but not needed in the
    #topology.
    j=1
    #This loop loops through all the neighbours in each Node_show_CDP.txt file
    #and adds them to the list, incase there are multiple by the same
    #interface it will add another entry in the dictionary, this is due to
    #the fact that a device might be connected to multiple devices
    #through one link.
    while j<len(CurrentFileLines):
        #Below temporary variable removes the \n from the files if it exists, as
        #different devices handle rows in different ways it was added here.
        CurrentLineTemp=CurrentFileLines [j].replace(\"\n\",\"\n\n\")
        CurrentLine="]
```

Listing 109: Ansible playbook CDPDiagramWithSTP.yml
for element in CurrentLineTemp:
    if element != ' ':
        CurrentLine.append(element)

j+=1
try:
    NodesConnected[CurrentNode] += [CurrentNode, CurrentLine[1], CurrentLine ← [0], CurrentLine[−1]]
except:
    NodesConnected[CurrentNode] = [CurrentNode, CurrentLine[1], CurrentLine ← [0], CurrentLine[−1]]

CurrentFile.close()

# NodesConnected now holds all the data about the connections. The structure is Local device, Local interface, Remote device, remote interface.

# Now to remove the loaded files. (I like to be modular so each section does something and then it is easier to test then just throw everything in at once)
# One reason for this to be done is so that old configurations files will not be added in the graph incase of the node being down.
# Scenario: If R1 is connected to R2 who is connected to R3 who is connected to R1, if R3 goes down the link is only R1 to R2, but if the files are not removed Ansible will not remove them (only update R1 and R2 file) and then R3 will magically be shown in the graph anyway. This can of course be commented out for troubleshooting purposes.

for FileName in ListOfFileNames:
    os.system("rm CDP/"+FileName)

Nodetokeys=sorted(NodesConnected.keys())
edges={}

# print Nodetokeys
for Nodes in NodetoKeys:
    i=0
    while i<len(NodesConnected[Nodes]):
        # Need a bit of magic to show the linknames correctly as sometimes it takes a space in the name from the show cdp neighbour file.
        # First splitting up the string by seperating everything with a space between.
        # Then emptying the empty elements and only keeping the elements with something in it. Doing this for the Local and Remote link.
        # The reason for this is that the tuples ('R1.AnsibleLab', 'Ser 1/2') and ('R1.AnsibleLab', 'Ser 1/2') are different due to the "Ser 1/2" and "Ser 1/2" being different.
        LocalLink=NodesConnected[Nodes][i+1].split(" ")
        # print ("LocalLink Before: "+str(LocalLink))
        LocalLink[:] = [item for item in LocalLink if item != '']
        LocalLink=str(LocalLink[0]+LocalLink[1])
        RemoteLink=NodesConnected[Nodes][i+3].split(" ")
        # print ("RemoteLink Before: "+str(RemoteLink))
        RemoteLink[:] = [item for item in RemoteLink if item != '' and item != "WS-C3750-"]
        # This is to avoid issues with C3750 taking up a lot of space in show cdp neighbor output and then the parser of the files do not understand as there is only one space between C3750- and the Interface name.
RemoteLink = str(RemoteLink[0] + RemoteLink[1])

# The Local variable holds the current Local node and its interface in a tuple, the Remote holds the remote node and its interface. This will then be compared against the edges dictionary so duplicate nodes and links will not show up in the graph. Edges will otherwise show both link from R1 to R3 and the same link from R3 to R1 even though it is one and the same.
Local = (NodesConnected[Nodes][i], LocalLink)
Remote = (NodesConnected[Nodes][i + 2], RemoteLink)

if (Local in edges.keys() and Remote in edges.values()):
    pass
    # Below print statement is kept for troubleshooting purposes
    # print("Duplicated local in keys remote in values")
elif (Remote in edges.keys() and Local in edges.values()):
    pass
    # Below print statement is kept for troubleshooting purposes
    # print("Duplicated Remote in keys Local in values")
else:
    edges[Local] = Remote
    i += 4

# Below is only for troubleshooting if needed or if we need data for manually creating the topology using CDP as the data source.
# print(edges)
# print(edges.keys())
# print(edges.values())

# Exception to check if there even are any files in the directory.
try:
   ListOfFileNamesSTP = os.listdir('./STP')
except:
    ListOfFileNamesSTP = ""

ListOfTopologies = ['Physical network', edges]

# Next section of the code is to go through the blocking ports from STP which will pop(remove) them from the dictionary for that particular VLAN.

for FileName in ListOfFileNamesSTP:
    # Split() can split up a string onto different words, if an argument is used it can split on that instead of space which is standard.
    # Because Ansible creates the files as Node_show_STP.txt it will split up the Node show STP.txt into different elements in a list. Only the first holds the node name.
    CurrentNode = FileName.split('_')[0] + "._AnsibleLab"
    # Opens the file and reads all the lines in the file.
    CurrentFile = open("STP/" + FileName, "rt")
    CurrentFileLines = CurrentFile.readlines()

    # Loops through all the lines in the file. The interface names in the file holds blocking interfaces meaning they should be removed from the topology.
for Lines in CurrentFileLines:
    CurrentEdges=edges.copy() #Copies the edges dictionary as without it ,
    → the two variables will point to the same data so changing
    → CurrentEdges will change the edges.
    CurrentLineTemp=Lines.replace(‘\n’,‘ ’).split(“ ”) #This is to remove
    → all new line signs in the list.
    CurrentLineTemp=Lines.replace(‘,’ ‘ ’).split(“ ”) #This is to remove
    → the , between multiple blocked ports if there are any.
    CurrentLineTemp[::]=[item for item in CurrentLineTemp if item!='']
#Below if–statements are to fix the output as CDP sees Gig/Fas but STP only
    → sees Gi/Fa as the start of the interface name. CurrentLineTemp[0] is
    → the VLAN number and CurrentLineTemp[1] is the interface that is
    → blocked in STP. It loops through as it can be multiple ports blocked
    → in one VLAN.
    y=0
    while y<len(CurrentLineTemp):
        if CurrentLineTemp[y][0:3]==' Fa':
            → ][3:]
        elif CurrentLineTemp[y][0:3]==' Gi':
            CurrentLineTemp[y]=CurrentLineTemp[y][1:3]+'g'+CurrentLineTemp[y
            → ][3:]
        # print CurrentLineTemp[y]
    #Below is the variable that holds the hostname and interface link same way
    → as in edges dictionary.
    comparing=(CurrentNode,CurrentLineTemp[y])
    #Below if–statements checks if the blocked port is in the CurrentEdges
    → dictionary which holds all interfaces according to CDP. If found the
    → dictionary entry is pop‘ed which means the key and value is removed
    → from the dictionary.
    if comparing in CurrentEdges.keys():
        CurrentEdges.pop(comparing)
    #Below if statement checks if the current link is in the values of the
    → dictionary and will then get the corresponding key and will pop that
    → key.
    if comparing in CurrentEdges.values():
        key=CurrentEdges.keys()[CurrentEdges.values().index(comparing)] #
        → Returns key when the value of the dictionary is comparing,
        → from stack overflow.
        CurrentEdges.pop(key)
        y+=1
        ListOfTopologies.append(CurrentLineTemp[0])
        ListOfTopologies.append(CurrentEdges)

for FileName in ListOfFileNamesSTP:
    os.system("rm STP/"+FileName)

#Next section is to figure out the root of the Spanning tree and remember
that label so it shows in the VLAN STP diagram.

# Exception to check if there even are any files in the directory.
try:
    ListOfFileNames = os.listdir(' ./STPRoot ')
except:
    ListOfFileNames = ""

STPRoot\VL\AN = {}

for FileName in ListOfFileNames:
    # Split() can split up a string onto different words, if an argument is
    # used it can split on that instead of space which is standard.
    # Because Ansible creates the files as Node_show_STP.txt it will split up
    # the Node show STP.txt into different elements in a list. Only the
    # first holds the node name.
    CurrentNode = FileName.split('_')[0] + "\Ansible\n"
    # Opens the file and reads all the lines in the file.
    CurrentFile = open("STPRoot/" + FileName, "rt")
    CurrentFileLines = CurrentFile.readlines()
    i = 0
    # Below loop checks all lines for This bridge is the root and if so the
    # previous element is the VLAN number the CurrentNode is root for.
    for Lines in CurrentFileLines:
        CurrentLineTemp = Lines.replace('\n', '').split("\n")
        CurrentLineTemp = [item for item in CurrentLineTemp if item != '']
        if "This bridge is the root" in str(CurrentLineTemp):
            STPRoot\VL\AN = CurrentFileLines[i - 1].replace('\n', '') = CurrentNode
            i += 1

for FileName in ListOfFileNames:
    os.system("rm STPRoot/" + FileName)

# Below code was copied from https://gist.github.com/floatingstatic/598
# f5258cd7fa554af785aa7dec4417d but changed a bit for my project.
```python
def draw_topology(topology, style, output_filename='topology.png'):
    nodes = set([key[0] for key in topology.keys() + topology.values()])
    g = graphviz.Graph(format='png')
    for node in nodes:
        g.node(node)
    for key, value in topology.items():
        head, t_label = key
        tail, h_label = value
        g.edge(head, tail, headlabel=h_label, taillabel=t_label, label="↪→"*12)
    g.graph_attr.update(
        ('graph' in style and style['graph']) or {})
    g.node_attr.update(
        ('nodes' in style and style['nodes']) or {})
    g.edge_attr.update(
        ('edges' in style and style['edges']) or {})
    g.render(filename=output_filename)

# Below goes through all the different sets of topologies and saves the topology as VLAN0001
# with its topology according to CDP with STPs blocked ports removed as they are not forwarding traffic.
# ListOfTopologies[i+1] is the interconnection of the nodes while
# ListOfTopologies[i] is the VLAN number. VLAN0001.png is the filename for the first VLAN and so on. Complete diagram is called Physical network.

i=0
while i<len(ListOfTopologies):
    # The label is used to show which switch is the STP root for that particular VLAN.
    Label=""
    if ListOfTopologies[i]=="Physical network":
        Label=ListOfTopologies[i]
    else:
        Label=STPRootVLAN|ListOfTopologies[i]|+" is the STP root for "+
    styles = {
        'graph': {
            'label': Label,
            'fontsize': '16',
            'fontcolor': 'white',
            'bgcolor': '#333333',
            'rankdir': 'BT',
```
Listing 110: Python script for creating network diagrams removing STP blocked ports

A.8 Saving configuration

The Ansible playbook for copying running configuration and saving it locally Running.yml:

```yaml
---
- hosts: GNS3
  gather_facts: no
  connection: local

  tasks:
  - name: Get credentials
    include_vars: ios.yml
  - name: Define connection
    set_fact:
      connection:
        authorize: yes
        username: "{{ ANSIBLE_NET_USERNAME }}"
        password: "{{ ANSIBLE_NET_PASS }}"
        auth_pass: "{{ ANSIBLE_NET_AUTH_PASS }}"
  - name: Gathering IOS running config
    ios_command:
      commands: show running
      provider: "{{ connection }}"
      register: config
  - name: Gathering IOS time
    ios_command:
      commands: show clock
      provider: "{{ connection }}"
      register: time

# - name: Gathering time
#   host: localhost
#   debug: msg="{{lookup('pipe','date --iso-8601=minutes')}}"
```
A.9 Ansible time

Raw data for adding VLAN through Ansible:

```
1 Switch try 1 SW:
markus@Borgen-kali:/Desktop/AnsibleProject/AnsibleVLAN$ time ansible --playbook AddingVLAN.yml
PLAY [Switches]

PLAY [Switches]

TASK [Define connection]

TASK [Generate VLAN configuration for each device]
```

Listing 111: Ansible playbook Running.yml
TASK [Pushing the configs created using JinjaAndYAML]
  ↩
  → ******************************************************
changed: [SW1]

PLAY RECAP
  ↩
  → ******************************************************
SW1 : ok=3 changed=1 unreachable=0 failed=0

real 0m7.752s
user 0m4.196s
sys 0m0.510s

1 Switch try 2 SW1:

markus@Borgen-kali:~/Desktop/AnsibleProject/AnsibleVLAN$ time ansible-
  ↩
  → playbook AddingVLAN.yml

PLAY [Switches]
  ↩
  → ******************************************************

TASK [Define connection]
  ↩
  → ******************************************************
ok: [SW1]

TASK [Generate VLAN configuration for each device]
  ↩
  → ******************************************************
changed: [SW1]

TASK [Pushing the configs created using JinjaAndYAML]
  ↩
  → ******************************************************
changed: [SW1]

PLAY RECAP
  ↩
  → ******************************************************
SW1 : ok=3 changed=2 unreachable=0 failed=0

real 0m8.162s
user 0m4.350s
sys 0m0.561s

1 Switch try 3 SW1:

markus@Borgen-kali:~/Desktop/AnsibleProject/AnsibleVLAN$ time ansible-
  ↩
  → playbook AddingVLAN.yml

PLAY [Switches]
  ↩
  → ******************************************************

TASK [Define connection]
  ↩
  → ******************************************************
ok: [SW1]

TASK [Generate VLAN configuration for each device]
  ↩
  → ******************************************************
changed: [SW1]

TASK [Pushing the configs created using JinjaAndYAML]
  ↪  ****************************************
changed: [SW1]

PLAY RECAP
  ↪  ****************************************
SW1: ok=3 changed=2 unreachable=0 failed=0

real 0m7.945s
user 0m4.265s
sys 0m0.543s

1 Switch try 1 SW2:
markus@Borgen-kali:~ /Desktop/AnsibleProject/AnsibleVLAN$ time ansible -
  ↪ playbook AddingVLAN.yml

PLAY [Switches]
  ↪  ****************************************
TASK [Define connection]
  ↪  ****************************************
ok: [SW2]

TASK [Generate VLAN configuration for each device]
  ↪  ****************************************
changed: [SW2]

TASK [Pushing the configs created using JinjaAndYAML]
  ↪  ****************************************
changed: [SW2]

PLAY RECAP
  ↪  ****************************************
SW2: ok=3 changed=2 unreachable=0 failed=0

real 0m7.678s
user 0m4.245s
sys 0m0.516s

1 Switch try 2 SW2:
markus@Borgen-kali:~ /Desktop/AnsibleProject/AnsibleVLAN$ time ansible -
  ↪ playbook AddingVLAN.yml

PLAY [Switches]
  ↪  ****************************************
TASK [Define connection]
  ↪  ****************************************
ok: [SW2]
TASK [Generate VLAN configuration for each device]
  ↦ ****************************
changed: [SW2]

TASK [Pushing the configs created using JinjaAndYAML]
  ↦ ****************************
changed: [SW2]

PLAY RECAP
  ↦ ****************************
SW2:    ok=3  changed=2  unreachable=0  failed=0

real:  0m6.679s
user:  0m4.157s
sys:   0m0.496s

1 Switch try 3 SW2:
markus@Borgen-kali:~ /Desktop/AnsibleProject/AnsibleVLAN$ time ansible-
  ↦ playbook AddingVLAN.yml

PLAY [Switches]
  ↦ ****************************

TASK [Define connection]
  ↦ ****************************
ok: [SW2]

TASK [Generate VLAN configuration for each device]
  ↦ ****************************
changed: [SW2]

TASK [Pushing the configs created using JinjaAndYAML]
  ↦ ****************************
changed: [SW2]

PLAY RECAP
  ↦ ****************************
SW2:    ok=3  changed=2  unreachable=0  failed=0

real:  0m6.496s
user:  0m4.127s
sys:   0m0.509s

2 Switches 1 Try:
markus@Borgen-kali:~ /Desktop/AnsibleProject/AnsibleVLAN$ time ansible-
  ↦ playbook AddingVLAN.yml

PLAY [Switches]
  ↦ ****************************

TASK [Define connection]
  ↦ ****************************
ok: [SW1]
ok: [SW2]

TASK [Generate VLAN configuration for each device]
  ↩  ******************
changed: [SW1]
changed: [SW2]

TASK [Pushing the configs created using JinjaAndYAML]
  ↩  ******************
changed: [SW2]
changed: [SW1]

PLAY RECAP
  ↩  ********************
SW1 : ok=3 changed=2 unreachable=0 failed=0
SW2 : ok=3 changed=2 unreachable=0 failed=0

real 0m7.854s
user 0m7.028s
sys  0m0.878s

2 Switches 2 Try:
markus@Borgen-kali:~/Desktop/AnsibleProject/AnsibleVLAN$ time ansible -
  ↩  playbook AddingVLAN.yml

PLAY [Switches]
  ↩  ********************

TASK [Define connection]
  ↩  ********************
ok: [SW1]
ok: [SW2]

TASK [Generate VLAN configuration for each device]
  ↩  ********************
changed: [SW1]
changed: [SW2]

TASK [Pushing the configs created using JinjaAndYAML]
  ↩  ********************
changed: [SW1]
changed: [SW2]

PLAY RECAP
  ↩  ********************
SW1 : ok=3 changed=2 unreachable=0 failed=0
SW2 : ok=3 changed=2 unreachable=0 failed=0

real 0m6.714s
user 0m6.704s
sys 0m0.832s

2 Switches 3 Try:
PLAY [Switches]
  => ***************************************************************

TASK [Define connection]
  => ***************************************************************
    ok: [SW1]
    ok: [SW2]

TASK [Generate VLAN configuration for each device]
  => ***************************************************************
    changed: [SW1]
    changed: [SW2]

TASK [Pushing the configs created using JinjaAndYAML]
  => ***************************************************************
    changed: [SW1]
    changed: [SW2]

PLAY RECAP
  => ***************************************************************

     ok | changed | unreachable | failed |
----------|----------|-------------|--------|
SW1       | 3        | 2           | 0      |
SW2       | 3        | 2           | 0      |

real 0m7.834s
user 0m6.684s
sys 0m0.810s

Listing 112: Switches time testing output

Raw data for opening/closing ASA firewalls through Ansible:

Ansible 1 ASA:
markus@Borgen-kali:~/Desktop/AnsibleProject/AnsibleASAWithInspection
$ time ansible -playbook ASAFirewallOpening.yml

PLAY [Firewalls]
  => ***************************************************************

TASK [Define connection]
  => ***************************************************************
    ok: [ASA1]

TASK [Generate removal of old Object and Deny statement for each device]
  => **********
    ok: [ASA1]

TASK [Generate new of Object and Deny statement for each device]
  => **********
    changed: [ASA1]

TASK [Generate the ACLs]
  => ***************************************************************
    ok: [ASA1]
TASK [get file content FWObject]
  changed: [ASA1]

TASK [get file content FWRemove]
  changed: [ASA1]

TASK [get file content FWACL]
  changed: [ASA1]

TASK [Clearing the old]
  changed: [ASA1] => (item=no access-list ACLAnsible1 line 1 extended deny
  object-group DenyObjectAnsible_1 any any)
changed: [ASA1] => (item=no object-group service DenyObjectAnsible_1)

TASK [Pushing the new object and deny statement created using JinjaAndYAML]
  changed: [ASA1] => (item=service udp destination eq 722)
changed: [ASA1] => (item=service tcp destination eq 811)

TASK [Pushing the new ACL]
  changed: [ASA1] => (item=access-list ACLAnsible1 line 1 extended deny
  object-group DenyObjectAnsible_1 any any)
changed: [ASA1] => (item=access-list ACLAnsible1 line 999 extended permit
  ip any any)
ok: [ASA1] => (item=access-group ACLAnsible1 in interface inside)

PLAY RECAP

ASA1    : ok=10  changed=7  unreachable=0  failed=0

real    0m19.611s
user    0m11.820s
sys     0m1.497s

Ansible 2 ASA:
markus@Borgen-kali:~ /Desktop/AnsibleProject/AnsibleASAWithInspection$ time
  ansible --playbook ASAFirewallOpening.yml

PLAY [Firewalls]

TASK [Define connection]

ok: [ASA1]
ok: [ASA2]

TASK [Generate removal of old Object and Deny statement for each device]

ok: [ASA2]
ok: [ASA1]
TASK [Generate new of Object and Deny statement for each device]
        ↩  ok: [ASA1]
        ok: [ASA2]

TASK [Generate the ACLs]
        ↩  ok: [ASA1]
        ok: [ASA2]

TASK [get file content FWObject]
        ↩  ok: [ASA1]
        changed: [ASA2]
        changed: [ASA1]

TASK [get file content FWRmove]
        ↩  ok: [ASA1]
        changed: [ASA2]
        changed: [ASA2]

TASK [get file content FWACL]
        ↩  ok: [ASA1]
        changed: [ASA2]
        changed: [ASA2]

TASK [Clearing the old]
        ↩  ok: [ASA1]
        changed: [ASA1] => (item=no access-list ACLansible1 line 1 extended deny
        => object-group DenyObjectAnsible_1 any any)
        changed: [ASA1] => (item=no access-list ACLansible1 line 1 extended deny
        => object-group DenyObjectAnsible_1 any any)
        changed: [ASA1] => (item=no object-group service DenyObjectAnsible_1)
        changed: [ASA2] => (item=no access-list ACLansible1 line 1 extended deny
        => object-group DenyObjectAnsible_1 any any)
        changed: [ASA2] => (item=no access-list ACLansible1 line 1 extended deny
        => object-group DenyObjectAnsible_1 any any)
        changed: [ASA2] => (item=no object-group service DenyObjectAnsible_1)

TASK [Pushing the new object and deny statement created using JinjaAndYAML]
        ↩  ok: [ASA1]
        changed: [ASA1] => (item=service udp destination eq 722)
        changed: [ASA1] => (item=service tcp destination eq 811)
        changed: [ASA2] => (item=service udp destination eq 722)
        changed: [ASA2] => (item=service tcp destination eq 811)

TASK [Pushing the new ACL]
        ↩  ok: [ASA1]
        changed: [ASA1] => (item=access-list ACLansible1 line 1 extended deny
        => object-group DenyObjectAnsible_1 any any)
        changed: [ASA1] => (item=access-list ACLansible1 line 1 extended deny
        => object-group DenyObjectAnsible_1 any any)
        ok: [ASA1] => (item=access-group ACLansible1 in interface inside )
        changed: [ASA2] => (item=access-list ACLansible1 line 1 extended deny
        => object-group DenyObjectAnsible_1 any any)
        changed: [ASA2] => (item=access-list ACLansible1 line 1 extended deny
        => object-group DenyObjectAnsible_1 any any)
        changed: [ASA2] => (item=access-list ACLansible1 line 999 extended permit
        => ip any any)
        ok: [ASA2] => (item=access-group ACLansible1 in interface inside )

PLAY RECAP
        ↩  ************
ASA1: ok=10 changed=6 unreachable=0 failed=0
ASA2: ok=10 changed=6 unreachable=0 failed=0

real 0m49.990s
user 0m23.799s
sys 0m3.547s

Ansible 3 ASA:
marcus@Borgen-kali:/Desktop/AnsibleProject/AnsibleASAWithInspection$ time → ansible-playbook ASAFirewallOpening.yml

PLAY [Firewalls]
  ↪ ****************************

TASK [Define connection]
  ↪ ****************************
  ok: [ASA1]
  ok: [ASA2]
  ok: [ASA3]

TASK [Generate removal of old Object and Deny statement for each device]
  ↪ *******
  ok: [ASA2]
  ok: [ASA1]
  ok: [ASA3]

TASK [Generate new of Object and Deny statement for each device]
  ↪ ***********
  ok: [ASA1]
  ok: [ASA2]
  ok: [ASA3]

TASK [Generate the ACLs]
  ↪ ****************************
  ok: [ASA2]
  ok: [ASA3]
  ok: [ASA1]

TASK [get file content FWObject]
  ↪ ****************************
  changed: [ASA3]
  changed: [ASA1]
  changed: [ASA2]

TASK [get file content FWRmove]
  ↪ ****************************
  changed: [ASA2]
  changed: [ASA1]
  changed: [ASA3]

TASK [get file content FWACL]
  ↪ ****************************
  changed: [ASA2]
  changed: [ASA1]
  changed: [ASA3]
TASK [Clearing the old]

changed: [ASA1] => (item=no access - list ACLAnsible1 line 1 extended deny

object - group DenyObjectAnsible_1 any any)

changed: [ASA1] => (item=no object - group service DenyObjectAnsible_1)

changed: [ASA2] => (item=no access - list ACLAnsible1 line 1 extended deny

object - group DenyObjectAnsible_1 any any)

changed: [ASA3] => (item=no access - list ACLAnsible1 line 1 extended deny

object - group DenyObjectAnsible_1 any any)

changed: [ASA2] => (item=no object - group service DenyObjectAnsible_1)

changed: [ASA3] => (item=no object - group service DenyObjectAnsible_1)

TASK [Pushing the new object and deny statement created using JinjaAndYAML]

changed: [ASA1] => (item=service udp destination eq 722)

changed: [ASA3] => (item=service tcp destination eq 811)

changed: [ASA2] => (item=service udp destination eq 722)

changed: [ASA3] => (item=service tcp destination eq 811)

changed: [ASA2] => (item=service tcp destination eq 811)

changed: [ASA3] => (item=service tcp destination eq 811)

TASK [Pushing the new ACL]

changed: [ASA1] => (item=access - list ACLAnsible1 line 1 extended deny

object - group DenyObjectAnsible_1 any any)

changed: [ASA1] => (item=access - list ACLAnsible1 line 999 extended permit

ip any any)

changed: [ASA3] => (item=access - list ACLAnsible1 line 1 extended deny

object - group DenyObjectAnsible_1 any any)

changed: [ASA2] => (item=access - list ACLAnsible1 line 1 extended deny

object - group DenyObjectAnsible_1 any any)

ok: [ASA1] => (item=access - group ACLAnsible1 in interface inside )

changed: [ASA3] => (item=access - list ACLAnsible1 line 999 extended permit

ip any any)

changed: [ASA2] => (item=access - list ACLAnsible1 line 999 extended permit

ip any any)

ok: [ASA3] => (item=access - group ACLAnsible1 in interface inside )

ok: [ASA2] => (item=access - group ACLAnsible1 in interface inside )

PLAY RECAP

ASA1 : ok=10 changed=6 unreachable=0 failed=0
ASA2 : ok=10 changed=6 unreachable=0 failed=0
ASA3 : ok=10 changed=6 unreachable=0 failed=0

real 1m2.935s
user 0m39.251s
sys 0m5.347s

Ansible 4 ASA:
markus@Borgen-ka:~/.Desktop/AnsibleProject/AnsibleASAWithInspection$ time
ansible -playbook ASAFirewallOpening.yml

PLAY [Firewalls]
TASK [Define connection]
  
ok: [ASA1]
ok: [ASA2]
ok: [ASA3]
ok: [ASA4]

TASK [Generate removal of old Object and Deny statement for each device]
  
ok: [ASA2]
ok: [ASA1]
ok: [ASA3]
ok: [ASA4]

TASK [Generate new of Object and Deny statement for each device]
  
ok: [ASA3]
ok: [ASA1]
ok: [ASA4]
ok: [ASA2]

TASK [Generate the ACLs]
  
ok: [ASA2]
ok: [ASA1]
ok: [ASA3]
ok: [ASA4]

TASK [get file content FWObject]
  
changed: [ASA3]
changed: [ASA4]
changed: [ASA1]
changed: [ASA2]

TASK [get file content FWRmove]
  
changed: [ASA1]
changed: [ASA2]
changed: [ASA3]
changed: [ASA4]

TASK [get file content FWACL]
  
changed: [ASA1]
changed: [ASA2]
changed: [ASA3]
changed: [ASA4]

TASK [Clearing the old]
  
changed: [ASA1]  => (item=no access-list ACLAnsible1 line 1 extended deny
                   object-group DenyObjectAnsible_1 any any)
changed: [ASA1]  => (item=no object-group service DenyObjectAnsible_1)
object-group DenyObjectAnsible_1 any any

changed: [ASA3] => (item=no access-list ACLAnsible1 line 1 extended deny
                     object-group DenyObjectAnsible_1 any any)
changed: [ASA4] => (item=no access-list ACLAnsible1 line 1 extended deny
                     object-group DenyObjectAnsible_1 any any)
changed: [ASA2] => (item=no object-group service DenyObjectAnsible_1)
changed: [ASA3] => (item=no object-group service DenyObjectAnsible_1)
changed: [ASA4] => (item=no object-group service DenyObjectAnsible_1)

TASK [Pushing the new object and deny statement created using JinjaAndYAML]

changed: [ASA1] => (item=service udp destination eq 722)
changed: [ASA1] => (item=service tcp destination eq 811)
changed: [ASA2] => (item=service udp destination eq 722)
changed: [ASA3] => (item=service udp destination eq 722)
changed: [ASA4] => (item=service tcp destination eq 811)
changed: [ASA4] => (item=service tcp destination eq 811)

TASK [Pushing the new ACL]

changed: [ASA1] => (item=access-list ACLAnsible1 line 1 extended deny
                     object-group DenyObjectAnsible_1 any any)
changed: [ASA1] => (item=access-list ACLAnsible1 line 999 extended permit
                     ip any any)
changed: [ASA3] => (item=access-list ACLAnsible1 line 1 extended deny
                     object-group DenyObjectAnsible_1 any any)
changed: [ASA4] => (item=access-list ACLAnsible1 line 1 extended deny
                     object-group DenyObjectAnsible_1 any any)
ok: [ASA1] => (item=access-group ACLAnsible1 in interface inside)
changed: [ASA2] => (item=access-list ACLAnsible1 line 1 extended deny
                     object-group DenyObjectAnsible_1 any any)
changed: [ASA4] => (item=access-list ACLAnsible1 line 999 extended permit
                     ip any any)
changed: [ASA3] => (item=access-list ACLAnsible1 line 999 extended permit
                     ip any any)
failed: [ASA2] (item=access-list ACLAnsible1 line 999 extended permit ip
               any any) => {"changed": false, "item": "access-list ACLAnsible1 line
               999 extended permit ip any any", "msg": "timeout waiting for command
to complete"}

ok: [ASA1] => (item=access-group ACLAnsible1 in interface inside)
ok: [ASA3] => (item=access-group ACLAnsible1 in interface inside)
changed: [ASA2] => (item=access-group ACLAnsible1 in interface inside)
to retry. use: --limit @/home/markus/Desktop/ansibleProject/
          AnsibleASAWithInspection/ASAFirewallOpening.retry

PLAY RECAP

ASA1          : ok=10  changed=6  unreachable=0  failed=0
ASA2          : ok=9   changed=5  unreachable=0  failed=1
ASA3          : ok=10  changed=6  unreachable=0  failed=0
ASA4          : ok=10  changed=6  unreachable=0  failed=0

real  1m7.872s
user  0m56.903s
Ansible 5 ASA:
markus@Borgen-kali:~/Desktop/AnsibleProject/AnsibleASAWithInspection$ time
   → ansible -playbook ASAFirewallOpening.yml

PLAY [Firewalls]
   → ****************************************************

TASK [Define connection]
   → ****************************************************
ok: [ASA1]
ok: [ASA2]
ok: [ASA3]
ok: [ASA4]
ok: [ASA5]

TASK [Generate removal of old Object and Deny statement for each device]
   → ******
ok: [ASA2]
ok: [ASA3]
ok: [ASA1]
ok: [ASA4]
ok: [ASA5]

TASK [Generate new of Object and Deny statement for each device]
   → *******
ok: [ASA1]
ok: [ASA2]
ok: [ASA4]
ok: [ASA3]
ok: [ASA5]

TASK [Generate the ACLs]
   → ****************************************************
ok: [ASA1]
ok: [ASA3]
ok: [ASA5]
ok: [ASA4]
ok: [ASA2]

TASK [get file content FWObject]
   → ****************************************************
changed: [ASA3]
changed: [ASA4]
changed: [ASA1]
changed: [ASA2]
changed: [ASA5]

TASK [get file content FWRmove]
   → ****************************************************
changed: [ASA1]
changed: [ASA5]
changed: [ASA3]
changed: [ASA2]
changed: [ASA4]
TASK [get file content FWACL]
  ↦ ..............................................................
  changed: [ASA1]
  changed: [ASA3]
  changed: [ASA4]
  changed: [ASA2]
  changed: [ASA5]

TASK [Clearing the old]
  ↦ ..............................................................
  changed: [ASA1] => (item=no access-list ACLansible1 line 1 extended deny
  ↦ object-group DenyObjectAnsible_1 any any)
  changed: [ASA1] => (item=no object-group service DenyObjectAnsible_1)
  changed: [ASA3] => (item=no access-list ACLansible1 line 1 extended deny
  ↦ object-group DenyObjectAnsible_1 any any)
  changed: [ASA5] => (item=no access-list ACLansible1 line 1 extended deny
  ↦ object-group DenyObjectAnsible_1 any any)
  changed: [ASA2] => (item=no access-list ACLansible1 line 1 extended deny
  ↦ object-group DenyObjectAnsible_1 any any)
  changed: [ASA4] => (item=no access-list ACLansible1 line 1 extended deny
  ↦ object-group DenyObjectAnsible_1 any any)
  changed: [ASA3] => (item=no object-group service DenyObjectAnsible_1)
  changed: [ASA2] => (item=no object-group service DenyObjectAnsible_1)
  changed: [ASA4] => (item=no object-group service DenyObjectAnsible_1)
  changed: [ASA5] => (item=no object-group service DenyObjectAnsible_1)

TASK [Pushing the new object and deny statement created using JinjaAndYAML]
  ↦ ****
  changed: [ASA1] => (item=service udp destination eq 722)
  changed: [ASA1] => (item=service tcp destination eq 811)
  changed: [ASA3] => (item=service udp destination eq 722)
  changed: [ASA2] => (item=service udp destination eq 722)
  changed: [ASA5] => (item=service udp destination eq 722)
  changed: [ASA4] => (item=service udp destination eq 722)
  changed: [ASA3] => (item=service tcp destination eq 811)
  changed: [ASA2] => (item=service tcp destination eq 811)
  changed: [ASA4] => (item=service tcp destination eq 811)
  changed: [ASA5] => (item=service tcp destination eq 811)

TASK [Pushing the new ACL]
  ↦ .................................
  changed: [ASA1] => (item=access-list ACLansible1 line 1 extended deny
  ↦ object-group DenyObjectAnsible_1 any any)
  changed: [ASA1] => (item=access-list ACLansible1 line 999 extended permit
  ↦ ip any any)
  changed: [ASA4] => (item=access-list ACLansible1 line 1 extended deny
  ↦ object-group DenyObjectAnsible_1 any any)
  changed: [ASA5] => (item=access-list ACLansible1 line 1 extended deny
  ↦ object-group DenyObjectAnsible_1 any any)
  ok: [ASA1] => (item=access-group ACLansible1 in interface inside)
  changed: [ASA3] => (item=access-list ACLansible1 line 1 extended deny
  ↦ object-group DenyObjectAnsible_1 any any)
  changed: [ASA4] => (item=access-list ACLansible1 line 999 extended permit
  ↦ ip any any)
changed: [ASA3] => (item=access-list ACLAnsible1 line 999 extended permit
  ip any any)
changed: [ASA2] => (item=access-list ACLAnsible1 line 999 extended permit
  ip any any)
ok: [ASA4] => (item=access-group ACLAnsible1 in interface inside )
changed: [ASA5] => (item=access-list ACLAnsible1 line 999 extended permit
  ip any any)
ok: [ASA2] => (item=access-group ACLAnsible1 in interface inside )
ok: [ASA3] => (item=access-group ACLAnsible1 in interface inside )
ok: [ASA5] => (item=access-group ACLAnsible1 in interface inside )

PLAY RECAP

ASA1: ok=10 changed=6 unreachable=0 failed=0
ASA2: ok=10 changed=6 unreachable=0 failed=0
ASA3: ok=10 changed=6 unreachable=0 failed=0
ASA4: ok=10 changed=6 unreachable=0 failed=0
ASA5: ok=10 changed=6 unreachable=0 failed=0

real 1m8.335s
user 1m11.502s
sys 0m9.182s

1 real ASA:
real 0m19.611s
user 0m11.820s
sys 0m1.497s

1 real ASA second try:
real 0m20.494s
user 0m11.744s
sys 0m1.517s

1 Virtual ASA:
ASA2:
real 0m58.475s
user 0m15.516s
sys 0m2.274s

ASA3:
real 0m57.220s
user 0m15.674s
sys 0m2.187s

ASA4:
real 0m55.729s
user 0m15.432s
sys 0m2.115s

ASA5:
real 0m54.875s
user 0m15.379s
sys 0m2.145s
ASA2 when only ASA2 is running:
real 0m45.606s
user 0m14.196s
sys 0m2.079s

Test with ASA1 and ASA2 but only ASA2 running virtually:
real 0m45.708s
user 0m24.187s
sys 0m3.328s

Shorted time:
1 ASA:
real 0m19.611s
user 0m11.820s
sys 0m1.497s

2 ASA:
real 0m49.990s
user 0m23.799s
sys 0m3.547s

3 ASA:
real 1m2.935s
user 0m39.251s
sys 0m5.347s

4 ASA:
real 1m7.872s
user 0m56.903s
sys 0m7.858s

5 ASA:
real 1m8.335s
user 1m11.502s
sys 0m9.182s

Listing 113: ASA time testing output

A.10 Python graphs

```python
# Example python code from https://pythonspot.com/matplotlib-bar-chart/
import numpy as np
import matplotlib.pyplot as plt

# ASA case first:
# Data to plot
n_groups = 5  # This is how many bars that should be created
```
means_Manual = ('83.48', '176.23', '232.91', '306.54', '398.51') # Manual time
data
means_Ansible = ('19.611', '49.990', '62.935', '67.872', '68.335') # Ansible
time data
fig, ax=plt.subplots()
index = np.arange(n_groups)
bar_width=0.35
opacity= 0.8

# First bar, manual bar.
rects1 = plt.bar(index, means_Manual, bar_width,
    alpha=opacity,
    color='b',
    label='Manual')

# Second bar, Ansible bar
rects2 = plt.bar(index+bar_width, means_Ansible, bar_width,
    alpha=opacity,
    color='g',
    label='Ansible')

# Below is some information about how the graph should be presented.
plt.xlabel("Amount of ASA's")
plt.ylabel("Time in seconds")
plt.title("ASA testing")
plt.xticks(index+bar_width,(1, 2, 3, 4, 5))
plt.legend()
plt.tight_layout()
plt.savefig('ASA.png')
plt.show()

# VLAN case second:

# Data to plot
n_groups = 6 # This is how many bars that should be created
time data
means_Ansible = ('7.752', '8.162', '7.945', '7.854', '6.714', '7.834') # Ansible
time data
fig, ax=plt.subplots()
index = np.arange(n_groups)
bar_width=0.35
opacity= 0.8

# First bar, manual bar.
rects1 = plt.bar(index, means_Manual, bar_width,
    alpha=opacity,
    color='b',
    label='Manual')

# Second bar, Ansible bar
rects2 = plt.bar(index+bar_width, means_Ansible, bar_width,
    alpha=opacity,
    color='g',
    label='Ansible')
label='Ansible')

# Below is some information about how the graph should be presented.
plt.xlabel("Amount of switches")
plt.ylabel("Time in seconds")
plt.title("VLAN testing")
plt.xticks(index+bar_width, ("1", "1", "1", "2", "2", "2"))
plt.legend()
plt.tight_layout()
plt.savefig('VLAN.png')
plt.show()

Listing 114: Python script for creating graphs