Erkki Hyytiäinen

Verkkoautomaation hyödyntäminen yritysverkoissa

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

30.10.2017

Metropolia

Tekijä Otsikko Sivumäärä Aika	Erkki Hyytiäinen Verkkoautomaation hyödyntäminen yritysverkoissa 32 sivua 30.10.2017
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Tietoverkot
Ohjaaja	Lehtori Erik Pätynen

Insinöörityön tarkoituksena oli tutkia, miten avoimen lähdekoodin automaatioalustaa pystyttäisiin hyödyntämään yritysverkoissa käytettävissä verkkolaitteissa. Valitun automaatioalustan toiminnallisuus ja ominaisuudet tarkastettiin syvällisemmin erityisesti verkkolaitteiden suhteen.

Työtä varten rakennettiin virtualisoitu ympäristö, jossa testattiin, kuinka tutkittavan automaatioalustan rooleja ja mallipohjia voidaan käyttää Ciscon verkkolaitteiden kanssa. Tuloksena saatiin uudelleenkäytettävät mallipohjat ja rooli kytkimien.

Työssä tarkasteltiin myös suosittuja verkkoteknologioita, kuten ohjelmisto-ohjattuja verkkoja, ja kuinka ne täydentävät tai lähtökohtaisesti parantavat verkkojen automaatiota.

Työssä ilmeni, että tutkittu alusta soveltuu nykyisten verkkolaitteiden hallintaan, mutta muovautuu hankalammaksi yrityskäytössä. Tuotantoympäristöissä se pitäisi integroida olemassa olevan ympäristön ja täydentävien järjestelmien kanssa tehokkuuden maksimoimiseksi, mikä taas tekee käyttöönotosta monimutkaisemman. Työssä tehtyjen ratkaisujen avulla pystytään automatisoimaan konfiguraatioita, ja niiden hyödyt korostuvat erityisesti laitemääriltään laajoissa ympäristöissä.

Avainsanat	Ansible, SDN, verkkoautomaatio, konfiguraationhallinta

Author Title Number of Pages Date	Erkki Hyytiäinen Utilizing network automation in corporate networks 32 pages 30 October 2017
Degree	Bachelor of Engineering
Degree Programme	Information and Communication Technology
Specialisation option	Telecommunications and Data Networks
Instructor	Erik Pätynen, Senior Lecturer

The purpose of this project was to study how an open source automation and configuration management platform could be leveraged in day-to-day work. The functionality and capabilities for the chosen automation platform, Ansible, were checked in-depth, specifically focusing on networking equipment support.

A virtual environment was also built to test how Ansible roles and templates can be used with the Cisco networking equipment. As a result, a reusable role with several templates was created.

This study also reviews current networking trends such as SDN, and how these trends also either complement, or improve network automation by design.

Ansible turned out to be a viable product for automating current generation networking devices. When applied to production environment, Ansible will need to be integrated with existing components and complementing platforms to maximize platform effectiveness, making the setup more complex. With the solutions created in this project, parts of network configurations can be automated, and the benefits are particularly emphasized in large environments.

Keywords	Ansible, SDN, network automation, configuration management

Sisällys

Lyhenteet

1 Johdanto 1

2 Verkkoautomaatio ja sen tarve 2

2.1 Automaation tarve 2
2.2 Verkkoautomaatiomenetelmät 2

3 Automaatioon vaikuttavat trendit 6

3.1 Ohjelmisto-ohjatut verkot 6
3.2 Ohjelmointirajapinta (API) 11
3.3 NETCONF-protokolla 12

4 Testaukseen valittu automaatioalusta ja sen toiminnallisuudet 13

4.1 Inventory 14
4.2 Moduulit 15
4.3 Playbook 16

5 Testiympäristön asennus 19

5.1 Ympäristön yleiskuva 19
5.2 GNS3-konfigurointi 19
5.3 Kontrollikoneen konfigurointi 21
5.4 Testitopologian määritys 22

6 Automaatiologiikan testaus verkkolaitteilla 23

6.1 Roolin luonti ja konfigurointi 23
6.2 Roolin testaus 27
6.3 Ympäristön jatkokehitys 31

7 Yhteenveto ja pohdintaa 31

Lähteet 33

Lyhenteet

API	Application Programming Interface. Ohjelmointirajapinta.
SDN	Software Defined Networking. Ohjelmisto-ohjatut verkot.
YAML	Yaml Ain't Markup Language. Helppolukuiseksi suunniteltu datan serialisointikieli.
CLI	Command Line Interface. Komentorivipohjainen käyttöliittymä.
GUI	Graphical User Interface. Graafinen käyttöliittymä.
VLAN	Virtuaalilähiverkko. Tekniikka verkon loogiseen osiointiin.
DHCP	Dynamic Host Configuration Protocol. Protokolla IP-osoitteiden jakamiseen.
NTP	Network Time Protocol. Aikatiedon tasausprotokolla.
SNMP	Simple Network Management Protocol. Protokolla laitteiden tilatietojen keräämiseen ja käsittelyyn.
ZTP	Zero Touch Provisioning. Täysautomaattinen provisointi.
LLDP	Link Layer Discovery Protocol. Siirtoyhteyskerroksella toimiva laitteen tilaa ja ominaisuuksia mainostava protokolla.
STP	Spanning-Tree Protocol. Siirtoyhteyskerroksella toimiva liikennesilmukoiden estoon kehitetty protokolla.
SDN	Software Defined Networking. Ohjelmisto-ohjatut verkot. Verkkosuuntaus, jossa ohjauskerros on eriytetty keskitettyyn pisteeseen eri laitteilta.

NFV	Network Functions Virtualization. Verkkolaitteiden virtualisointi. Menetelmä, jossa eri verkkolaitteiden toiminnallisuus on siirretty virtuaalialustoille.
MAC	Media Access Control. Siirtoyhteyskerroksen menetelmä kerroksen osoitteille ja liikennöinnille. Työssä viitataan MAC-tauluun joka sisältää portteja tai verkkokortteja yksilöiviä MAC-osoitteita.
REST	Representational State Transfer. Arkkitehtuurimalli sovellusrajapintojen toteuttamiseen http-protokollan avulla.
SD-WAN	Software Defined Wide Area Networks. Ohjelmisto-ohjatut laajaverkot. Alikäsite ohjelmisto-ohjatuille verkoille. Menetelmä tuoda kattokäsitteen käsitteitä laajaverkkoyhteyksiin (ks. SDN).
SaaS	Software as a Service. Sovellus palveluna. Pilvipalveluihin liittyvä käsite, jossa sovellus suoritetaan pilvessä.
JSON	Javascript Object Notation. Tiedostomuoto tiedon levittämiseen strukturoidussa muodossa avain-arvoparein
XML	Extensible Markup Language. Tiedostomuoto tiedon levittämiseen ja rakenteelliseen esittämiseen.
RFC	Request For Comments. Numeroitu kokoelma dokumentteja ja standardeja koskien Internetin käytännöistä ja teknisistä menetelmistä.
MIB	Management Information Base. Hierarkkisesti järjestelty joukko tietoa, joilla kuvataan laitteita.

1 Johdanto

Automaation suosio kasvaa jatkuvasti globaalilla tasolla yrityksissä lähes toimialasta riippumatta. Se on yhteiskunnallisella tasolla tärkeä keskustelunaihe sen työllisyyttä muokkaavan voiman takia: kokonaiset ammattikunnat ovat katoavaa työtä automaation ja koneoppimisen kehittymisen vuorovaikutuksen seurauksena. [1.] Oxfordin yliopistossa tehdyn tutkimuksen perusteella jopa 47 % vuonna 2010 mitatuista Yhdysvalloissa sijaitsevista työtehtävistä saattaa kadota automaation seurauksena tulevien vuosikymmenten aikana [2].

Automaatioasteita voi informaatioteknologian alalla kehittää monella tavalla. Insinöörityöni tarkoituksena on selvittää avoimen lähdekoodin työkalujen avulla, miten automaatiota voi hyödyntää verkkolaitteissa. Työssä vertaan tuloksiani oman yritykseni nykytilanteeseen ja teen huomioita, miten tuloksia voisi ottaa käyttöön automaatioasteen kasvattamiseksi. Tärkeimpänä kohdealueena on poistaa ylimääräiset toistuvat työvaiheet, mutta lisäksi selvittää ad hoc -tyyppisten muutoksien tehokkuus verkkolaitteilla suhteessa suoriin muutoksiin laitteilta.

Työskentelen itse yrityksessä, joka tuottaa IT-palveluita ja konsultaatiota. Verkkolaitteet ja niihin liittyvät konsultaatio sekä ylläpitopalvelut ovat suuri osa yrityksen liikevaihdosta. Yritys myös myy monien eri verkkolaitetoimittajien laitteita ja niiden ylläpitopalvelua, joten diversiteetti on suuri. Insinöörityöllä ei kuitenkaan ole suoranaista tilaajaa, vaan tarkoituksena on tutkia päivittäisessä työssä vastaan tulleita tilanteita, joissa automaatioalustan käyttöönotto voisi poistaa ylimääräisiä työvaiheita ja kehittää tuottavuutta.

Tarkastelen myös verkkoteknologioiden nykytrendejä ja sitä, miten ne itsessään tuovat automaatiota tai helpottavat automaation käyttöönottoa.

2 Verkkoautomaatio ja sen tarve

2.1 Automaation tarve

Perinteisesti kaikki verkkolaitteille tehtävät muutokset on tehty joko komentorivipohjaisesti (CLI) tai graafisen käyttöliittymän avulla (GUI), laite kerrallaan. Tämä malli on hidas ja altis inhimillisille virheille. Lisäksi kaikki ylimääräinen toisto on pois tuottavuudesta ja tuotannon kehittämisestä. Laitetoimittajat ja ulkopuoliset tahot ovat kuitenkin heränneet tarpeeseen joko kehittää automaatiota tai tuoda vaihtoehtoisia menetelmiä verkkolaitteiden hallittavuuteen.

Yleisesti verkkojen automaatiossa pätee sama tarve kuin muussakin automaatiossa, eli halutaan tehdä asiat nopeammin ja hallitummin, toisin sanoen tuottavammin ja riskittömämmin. Esimerkkinä voidaan pitää vaikkapa uuden virtuaaliverkon (VLAN) lisäämistä kymmenen kytkimen verkkoon. Ilman automaatiota lisäyksestä vastaavan henkilön täytyy kirjautua jokaiselle laitteelle erikseen, lisätä virtuaaliverkko laitteen kantaan sekä määrittää se tarvittavissa reuna- ja runkoporteissa. Portteja määrittäessä voi tapahtua inhimillinen virhe ja esimerkiksi konfiguraatio voi unohtua tarpeellisesta runkoportista, jolloin liikenne siitä pisteestä eteenpäin katkeaa. Tällöin jo muutenkin toistuvan prosessin lisäksi kestää kauemmin selvittää, miksi tehty muutos ei toiminutkaan halutulla tavalla.

Kuitenkin uusien järjestelmien käyttöönotossa on aina otettava huomioon toteutuksen haasteellisuus: ylittävätkö uuden järjestelmän hyödyt lähtöinvestoinnin ja kuinka nopeasti. Uuden automaatioalustan implementoiminen tuotantoon vaatii ainakin seuraavat asiat harkittavaksi: järjestelmän kompleksisuus ja oppimiskäyrä sekä se, kuinka laajasti järjestelmä voidaan sitoa omaan tuotantoympäristöön.

2.2 Verkkoautomaatiomenetelmät

Verkkoautomaatiossa täytyy ottaa huomioon, mitä kaikkea pystytään automatisoimaan. Omien tarpeiden mukaisesti pystytään automatisoimaan esimerkiksi pelkästään provisiointia ja laitteiden toiminnan määrittelyjä, kuten konfiguraatiomuutoksia. Lisäksi verkkolaitteiden automaattinen reagointi poikkeamiin on esimerkki algoritmeihin perustuvasta monimutkaisemmasta automaatiosta, jolloin mukaan voidaan lisätä vielä koneoppimisen

käsite. Seuraavaksi käyn läpi erilaisia automaatiomenetelmiä ja -malleja. Osa tekniikoista tai malleista on avoimiin standardeihin perustuvia ja osa laitetoimittajien omia ratkaisuja.

Provisiointi

Provisiointi on hyvin konkreettinen ja selkeä esimerkki automatisoitavasta tehtävästä. Provisointi itsessään tarkoittaa ensikonfigurointia. Automaation kompleksisuus riippuu siitä, kuinka vähän halutaan manuaalista ihmisen tekemää konfiguraatiota. Uusien laitteiden konfiguraatioista iso osa on luonteeltaan staattista, joten syvällisempää automaatiologiikkaa tai laitteen itsenäisiä päätöksiä ei usein tarvita. Esimerkkinä mainittakoon aikapalvelimet (NTP, Network Time Protocol), tai SNMP (Simple Network Management Protocol) -konfiguraatiot, jotka ovat ympäristössä usein staattisia.

Verkkolaitetoimittajat tarjoavat myös esimerkiksi omia ZTP (Zero Touch Provisioning) -ratkaisuja, joissa parhaissa tapauksissa laitetta ei tarvitse esikonfiguroida paikan päällä ollenkaan. Tarkastelen seuraavaksi laitevalmistaja Aristan ratkaisua. Valmistajan kytkimet, jotka tukevat ZTP:tä, yrittävät automaattisesti käynnistyksessä DHCP-kyselyiden avulla saada IP-osoitteen sekä tiedon käynnistystiedostosta DHCP-optiolla 67. [3.] Alla on listattu esimerkkikäynnistystiedosto [4], jossa kopioidaan haluttu konfiguraatio ja asennetaan päivitetty versio kytkimelle:

```
[root@cozy Arista]$ more Arista1-ZTP
#!/usr/bin/Cli -p2
enable
copy http://www2.gad.net/Arista/Arista1-startup
flash:startup-config
copy http://www2.gad.net/Arista/EOS-4.9.3.swi flash:
config
boot system flash:EOS-4.9.3.swi
```

Laitetietojen keräys

Laitteilta kerättävien tiedon, kuten liikennemäärät hoitaa usein valmis monitorointijärjestelmä. Järjestelmällä saattaa olla kaikki tarvittavat työkalut riittävään tilastotietojen ke-

ruuseen tai mahdollisuus spesifiseen räätälöintiin, jos haluttua toiminnallisuutta ei sellaisenaan löydy. Automaatioalustoilla voidaan kuitenkin täyttää mahdolliset aukot mitä monitorointijärjestelmä ei välttämättä pysty tarjoamaan, sekä myös täydentää muuta automaatiota, kuten vianratkontaa. Valmiit työkalut on toki suoraan valjastettu rooliinsa, mutta tällöin toimitaan myös työkalun tai sen toimittajan ehdoilla. Datan keruu voi olla siis hieman kaksijakoista. Toisaalta saadaan huomattava vapaus mitä laitteilta kerätään, miten kerätään ja kuinka dataa käytetään. Vastapainona kuitenkin vastuu teknisestä toteutuksesta ja logiikan jatkokehityksestä jää itselle.

Käytännön esimerkkinä voidaan käyttää vaikkapa dokumentaation ylläpidon helpottamista. Automaatioalusta voi kerätä halutulta toimipisteeltä LLDP (Link Layer Discovery Protocol) -protokollan avulla tiedon verkosta ja syöttää sen työkalulle, joka koostaa lopulta topologiasta kuvan. Työkalua ei tarvitse ohjelmoida itse, sillä apuna voidaan käyttää avoimen lähdekoodin työkaluja, kuten Graphvizia. [5.] Toiminnallisuus voidaan liittää vaikka käynnistymään automaattisesti lisälaitteen provisioinnin yhteydessä, jolloin verkkokuva päivittyy lisäyksen myötä.

Migraatiot

Migraatiolla tarkoitetaan tässä yhteydessä esimerkiksi laitteen vaihtoa uudempaan malliin, tai kokonaan siirtymistä toiseen alustaan. Yleinen työelämässä käytetty termi on yliheitto. Migraation automatisoinnin houkuttelevuutta kasvattavat muun muassa eri toimittajien omat syntaksit; konfiguraatiota ei voida usein vain kopioida uudelle laitteelle, vaan se joudutaan muuntamaan uuden laitteen ymmärtämään muotoon. Automaation jouhevuuden parantamiseksi on kehitelty erinäisiä yhtenäisiä datamalleja, sekä työkaluja joita voidaan käyttää automaatioalustojen apuna, kuten NAPALM (Network Automation and Programmability Abstraction Layer with Multivendor support). Käytännössä NAPALM:n tarkoitus on toimia yhtenäisenä, laitetoimittajista riippumattomana rajapintana. [6.] NAPALM:n hyödyt eivät toki rajoitu pelkästään migraatioihin, vaan sitä voidaan käyttää esimerkiksi yleisesti konfiguraation hallinnassa. Konfiguraation abstraktion edut nähdään hyvin konkreettisesti kuvasta 1. Kahden eri laitevalmistajan hallintarakenne on huomattavan erilainen, jolloin konfiguraatio pitäisi rakentaa käsin uudelleen migraatiossa. Lisäksi automaatioalustallakin tarvittaisiin omanlainen toimintalogiikka laitevalmistajakohtaisesti.

Kuva 1. NAPALM:lla abstraktoitu konfiguraatio [7].

Konfiguraationhallinta

Konfiguraationhallinta käsitteenä tarkoittaa tässä yhteydessä tilaa johon halutaan päästä, ja jota halutaan ylläpitää. Verkkolaitteiden tapauksessa tämä voi tarkoittaa vaikkapa uuden verkon konfigurointia, tai uuden säännön lisäämistä palomuurin sääntökantaan. Luvussa 2 mainittu esimerkki virtuaaliverkkojen konfiguroinnista kuuluu juuri konfiguraationhallinnan piiriin.

Vikatilanteiden analysointi ja korjaaminen

Vianratkonta manuaalisesti pelkän komentorivin avulla on hyvin aikaa vievää, varsinkin jos ongelmatilanteessa tila täytyy tarkistaa useammilta laitteilta. NetBrain Technologiesin tekemässä kyselyssä 83 % vastaajista ilmoitti yrityksensä verkon kasvaneen viimeisen vuoden aikana. Lisäksi 71 % vastaajista sanoi käyttävänsä laitteiden komentoriviä pääasiallisena työkalunaan vianratkonnassa ja 40 % vastasi tyypillisen verkko-ongelman ratkaisuajaksi yli neljä tuntia. [8.] Verkko-ongelmat voivat usein osoittautua rahallisesti hyvin kalliiksi organisaatiolle. Gartnerin teettämässä kyselyssä keskimääräinen minuuttikustannus oli vuonna 2014 noin 5 600 dollaria [9].

Automaatiomenetelmien avulla voidaan toteuttaa vaikkapa yksinkertaisesti tarkistuslista. Esimeriksi alusta voi tarkistaa kytkinverkosta massa-ajona perusasiat, kuten broadcast-liikenteen määrän tai STP (Spanning-tree Protocol) -protokollan topologiamuutokset tai

sen, onko STP konfiguroitu oikein joka kytkimelle. Esimerkkiä voidaan jatkaa tekemällä vaikkapa yksinkertaiset korjausliikkeet saatujen tulosten perusteella, kuten verkkoa häiritsevän portin automaattinen sulkeminen ja siitä ylläpitäjälle raportoiminen.

3 Automaatioon vaikuttavat trendit

Nykyhetkellä suuri osa verkkolaitteista on komentorivipohjaisia ja niiden pääasiallinen automaatio perustuu skriptivetoisiin alustoihin ja screen scraping -teknologiaan, jossa tulosteista poimitaan toiminta- ja automaatiologiikan kannalta tärkeitä asioita. Isoiksi ongelmakohdiksi muodostuvat epäkäytännöllisyys ja haasteellisuus. Laitteet eivät palauta automaatioalustalle suoraan tilatietoa, vaan se välillisesti poimitaan ensin kirjautumalla laitteelle, suorittamalla tarvittavat komennot valmistajan syntaksin mukaan ja parsimalla tarvittu tieto. Lisäongelmiksi muodostuvat tässä tapauksessa muun muassa datan validointi ja itse parsiminen. Poimittaessa tietoa komentorivin syötteestä on jokainen välilyönti ja sarkain otettava huomioon. [10.] Kaiken tämän lisäksi data on vielä tuloste, joka on tarkoitettu ihmisille eikä se ole strukturoidussa muodossa, jotta tärkeitä palasia voisi käyttää esimerkiksi muuttujina suoraan palautetun tekstin perusteella.

Tämän työn luvussa 6 tarkastellaan, miten automaatio toimii juuri tälle mittavalle laitemassalle. On kuitenkin tärkeää myös katsoa mihin automaatio on kehittymässä ja miten pinnalla olevat verkkoteknologian trendit itsessään jo tuovat automaatiologiikkaa.

3.1 Ohjelmisto-ohjatut verkot

Ohjelmisto-ohjatut verkot (Software Defined Networking) ovat olleet huomattavassa nosteessa viime vuosina, ja siitä onkin muodostunut eräänlainen IT-alan muotisana [11]. Sen tarkka määritys on hieman hankalaa, sillä teknologia elää ja kehittyy jatkuvasti. Vuosien saatossa termin alle onkin lisätty monia muita eri merkityksiä ja termejä, kuten OpenFlow-protokolla tai NFV (Network Functions Virtualization). [12.] Myös eri laitevalmistajat määrittelevät monesti oman toteutuksen mukaisesti ohjelmisto-ohjattujen verkkojen termiä. Ohjelmisto-ohjatut verkot kattoterminä koskee tällä hetkellä lähinnä konesalien verkkoja.

Pilviteknologioiden suosion räjähdysmäinen nousu on pääasiallisena syynä siihen, miksi ohjelmisto-ohjatut verkot saavat niin suurta huomiota, sillä pilvipalveluiden ja niiden mukana tuodun lähes reaaliaikaisen palvelin- ja laskentatehon skaalautuvuuden myötä konesalien verkot jaetulla ohjauskerroksella eivät pysy mukana kehityksessä. Hieman avaamalla ongelmaa saadaan kolme pääsyytä, miksi vanha malli ei riitä pilvipalveluille:

1 tekniset esteet

2 verkkolaitteiden hinta suhteessa muihin laitteisiin

3 innovaatiosuhde muihin konesaliverkon laitteisiin [13].

Perinteisten konesalin verkkolaitteiden kaikki päätöksenteko on jokaisella laitteella erikseen, ja siksi päätöksien käsittely tai uuden toiminnan lisääminen vaatii konfigurointia usealla eri laitteella. Jokainen laite myös muodostaa ympäröivästä verkostaan kuvan omasta perspektiivistään. Usein konesaleissa ylläpidetään monien eri organisaatioiden laitteita, jotka pitää jakaa toisiltaan piilossa oleviin osiin. Tämä tuo teknisen haasteen muun muassa virtuaaliverkkojen riittävyydelle ja MAC-taulujen (Media Access Control) maksimikoon täyttymiselle. Hintataso ja innovaation niukkuus pysyvät taas korkeana muun muassa laitetoimittajien suljetun ympäristön vuoksi, perinteisillä verkkolaitteilla ei ole käytännössä avointa lähdekoodia. Täten laitekustannuksia nostaa kehitystyö, kun kaikki yleiset laitevalmistajat rakentavat identtistä toimintalogiikkaa eri laitteille. Valmistajat myös pitävät hallussaan sekä komponentit että sovellustason toiminnan, mikä tekee innovaatioista huomattavasti vaikeampaa. Operatiiviset kulut kasvavat toistuvan laitekohtaisen hallinnan vuoksi. [13; 14.]

Ohjelmisto-ohjattujen verkkojen eräänä ydinkonseptina onkin erottaa päätöksiä tekevä ohjauskerros (control plane) itse dataa vastaanottavasta välitystasosta (data plane). Ohjelmisto-ohjatuissa verkoissa ohjauskerroksen roolin ottaa kontrolleri, jonka funktiona on nähdä ja hallita koko ympäröivää verkkoa. Tällä arkkitehtuurimuutoksella saadaan automaation ja yleisen hallittavuuden kannalta etuja, kun pystytään kontrolloimaan monia laitteita keskitetysti. Traditionaalisesti jokaisella verkkolaitteella on kontrolli vain itsestään, joten ohjelmisto-ohjatut verkot muuttavat arkkitehtuurillista paradigmaa tämän osalta. [14.]

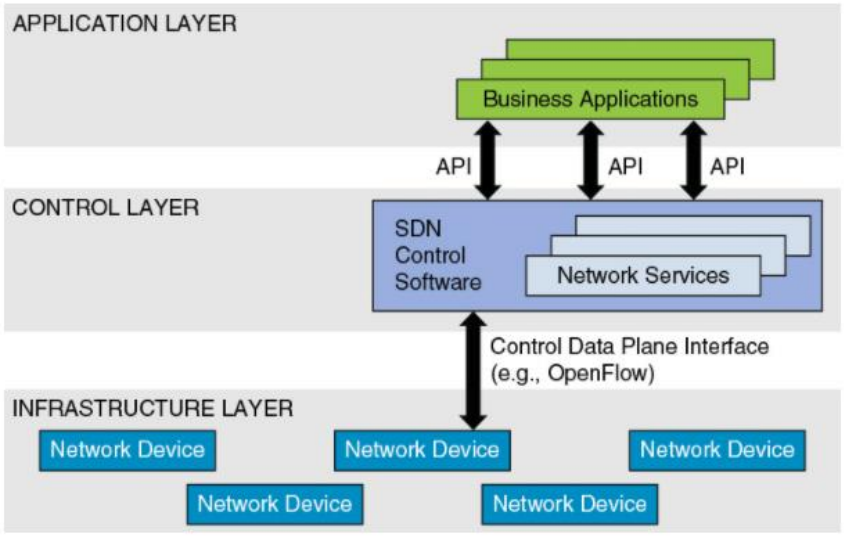
Ohjelmisto-ohjattujen verkkojen arkkitehtuuri jakaantuu perusmallissaan kolmeen kerrokseen:

- sovelluskerros
- ohjauskerros
- infrastruktuurikerros [15].

Sovelluskerros toimii ylimpänä pinossa kuvan 2 mukaisesti, ja sen funktiona on esimerkiksi toimittaa ulkopuoliselle järjestelmälle tietoa verkosta tai vastavuoroisesti sen välityksellä voidaan kontrolloida alempia kerroksia esimerkiksi ulkopuolisen järjestelmän, tai sovelluksen avulla [14].

Ohjauskerroksessa sijaitsee jo aiemmin mainittu kontrolleri, joka liikenteen päätöksenteon lisäksi määrittää, mitä protokollia ja teknologioita verkko käyttää. [17.] Alin kerros taas pitää sisällään juuri lopulliset liikennettä ohjaavat verkkolaitteet, joiden päätöksenteko ja suurin osa älystä on eroteltu kontrollikerrokseen.

Kokonaisuudessaan käyttäjä tai järjestelmä kommunikoi esimerkiksi REST-rajapinnalla (Representational State Transfer) kontrollerin kanssa, ja kontrolleri kommunikoi verkkolaitteiden kanssa esimerkiksi OpenFlow'lla. Yleisesti terminologiassa alempaa arkkitehtuuritasoa kohtaan kommunikoivia rajapintoja kutsutaan nimellä southbound interface ja vastavuoroisesti ylempiä kerroksia kohden nimellä northbound interface. [18.] Hahmottamisen helpottamiseksi kuvasta kaksi nähdään arkkitehtuurikerrokset ja niiden vuorovaikutukset.



Kuva 2. Ohjelmisto-ohjattujen verkkojen arkkitehtuurikerrokset [16].

Kun tarkastellaan SDN-arkkitehtuuria puhtaasti automaation näkökulmasta, voidaan todeta sen tuovan monia mahdollisuuksia eri verkkofunktioiden automatisointiin. Koska kontrolleri näkee myös koko verkkonsa, se pystyy myös automaattisesti havaitsemaan ruuhkaantuneen linkin ja siirtämään liikennettä toista polkua pitkin. Ohjelmisto-ohjattujen verkkojen perustavanlaatuinen ohjauskerroksen erottaminen onkin automaation kannalta juurikin keskeisin etu, kun laitteita ei tarvitse yksitellen konfiguroida ja hallita, ja verkkoliikenteen käsittely tarpeiden perusteella muodostuu selkeämmäksi keskitetyn kokonaiskuvan vuoksi.

Ohjelmisto-ohjatut laajaverkot (SD-WAN)

Koska pääasialliset ohjelmisto-ohjattujen verkkojen käyttötarkoitukset keskittyvät konesaliverkkoihin, on hyvä tarkastella myös kehittyvää teknologiaa, joka keskittyy organisaatioiden haaratoimistoihin. Kuten kattoterminsä, SD-WAN on todella suosittu kehittyvä teknologia. Esimerkiksi operaattorit ovat ilmaisseet kiinnostusta teknologiaan siirtyessään pelkästä infrastruktuurin tarjoajasta älykkäisiin lisäpalveluihin [19].

Ohjelmisto-ohjatut laajaverkot pohjautuvat osittain samaan ideaan kuin SDN, eli ohjaus- ja välitystason erottamiseen. Keskitetty hallinta voidaan toteuttaa esimerkiksi laitetoimittajan pilveen, josta organisaatio voi hallita kaikkia laitteitaan. Gartner määrittelee SD-WAN:lle neljä minimiominaisuutta:

- tuki monelle yhteydelle, kuten MPLS tai 4G
- dynaaminen reitinvalinta ja kuormantasaus
- yksinkertainen käyttöönotto
- tuki virtuaalisille erillisverkoille (VPN) [20].

Vallitsevassa nykytilanteessa yritykset käyttävät usein toimipisteiden laajaverkkoyhteyksinä operaattorien tarjoamia yksityisverkkoja pääasiallisina liittyminään. Varayhteytenä saattaa olla tavallinen laajakaistaliittymä, jonka yhteydessä käytetään VPN-tunnelia. Varaliittymä on mahdollisesti käytössä vain, jos pääliittymä lakkaa toimimasta virhetilanteen vuoksi, tai mahdollisesti varaliittymän kautta ohjataan vain vierasverkon liikennettä. Mitä hienojakoisemmin liikennettä aletaan ohjata liittymien välillä, sitä vaikeaselkoisemmaksi

sen hallinta muodostuu. [12; 13.] Lisäksi yksityiset yritysverkkoliittymät ovat huomattavasti kalliimpia, vaikkakin operaattorien palvelutasot ovat minimissäänkin huomattavasti parempia kuin tavallisissa laajakaistaliittymissä. Liikenteen optimointi näiden liittymien välillä onkin yksi seikka, jonka avulla laitetoimittajat markkinoivat SD-WAN toteutuksien tuomia etuja. Liikenteen käsittely on helpompaa ja kalliista yritysverkkoliittymästä voidaan karsia kapasiteetista hienojakoisemmalla liikenteen jaolla. [18.]

SD-WAN:n tapauksessa haaratoimistojen laitteistot muodostavat toistensa välille loogisen overlay-tunnelin. Tämä tunneli on täysin riippumaton sen alla olevasta fyysisestä verkosta, eli sille ei ole merkitystä onko verkko toteutettu esimerkiksi 4G- tai MPLS-yhteydellä. Laitteet keskustelevat keskenään tunnelien välityksellä ja toimittajan ratkaistusta riippuen mittaavat verkon toimintaa ja tilaa. Keskitetty näkymä pystyy seuraamaan kaikkien toimipisteiden tilaa sekä näkee kokonaiskuvan verkosta. Tämän kokonaiskuvan takia pystytään tekemään hienojakoisempia päätöksiä liikenteen ohjaamiseen. [12; 21.]

SD-WAN ei ole mitenkään standardoitu teknologia, vaan jokainen toimittaja tarjoaa omia ratkaisujaan, ja täten toiminnallisuudessa voi olla eroja. Esimerkin vuoksi tarkastelen SD-WAN-toimittaja Silver Peakin ratkaisua. Silver Peakin tarjoamat SD-WAN-laitteet kulkevat EdgeConnect-nimellä, ja niitä on mahdollista käyttää joko virtualisoituina tai fyysisinä laitteina. Laitteet muodostavat full-mesh-topologian loogisten tunnelien avulla. Liikennettä voidaan kontrolloida liiketoiminnan tarpeiden (Business Intent Policy) perusteella tunnelien välillä. Käytännössä liikennettä voidaan haluttujen linjausten pohjalta ohjata esimerkiksi kriittisyysperustaisesti, ja liikennekohtainen topologia voi olla esimerkiksi Hub-Spoke-mallinen. Liikenteen käyttäytymistä käsitellään myös automaation avulla: jos jokin linkki tunnistetaan ylikäytetyksi tai siinä ilmenee virheitä, pystytään overlay-tunnelin liikenne siirtämään toiselle liittymälle. Manuaalinen muokkaus taas tehdään linjauksen perusteella. Laitteet pystyvät myös nykyisten palomuurien tapaisesti tekemään sovellustunnistusta, joten politiikkaa voidaan tehdä myös sovelluskohtaisesti. [22.]

Ympäristöä kontrolloidaan orkestraattorin kautta, joka sijaitsee joko organisaation omassa ympäristössä virtuaalikoneena, tai pilvipalveluna. Käyttöliittymä on täysin HTML5-pohjainen. Lisenssien perusteella ympäristössä voidaan toteuttaa myös laajaverkkojen kiihdytystä sekä SaaS (Software as a Service) -optimointia suosituimpiin palveluihin (Esimerkiksi Salesforce tai Office 365). [22.]

Automaation näkökulmasta SD-WAN tarjoaa edistystä esimerkiksi provisiointiin: laitteet voidaan lähettää suoraan toimipisteelle, ne saavat konfiguraationsa keskitetystä hallinnasta, eivätkä tarvitse esikonfiguraatiota. Muutoksenhallinta toteutetaan myös keskitetysti, joten laitekohtaisesti muutoksia ei tarvitse tehdä ilman erityistä syytä. Lisäksi yksinkertaistetumpi liikenteen käsittely linkkien välillä, sekä liikenteen automaattinen siirto esimerkiksi viallisen linkin tapauksessa tuovat lisäarvoa hallintaan ja vikatilanteisiin.

3.2 Ohjelmointirajapinnat (API)

Ohjelmointirajapinnat, eli API:t ovat yleistymässä verkkolaitteissa pitkäaikaisen tarpeen vuoksi. Kuten luvun 3 alussa on pohjustettu, komentorivipohjainen laitevalmistajan syntaksilla toteutettava hallinta ja datan keruu ei skaalaudu ja tuottaa vastaukset vain ihmiselle helposti ymmärrettävässä muodossa. Ohjelmointirajapinnat sen sijaan palauttavat datan strukturoidussa muodossa, josta se voidaan parsia helposti ja käyttää hyväksi helpommin, esimerkiksi automaatioalustassa tai monitorointijärjestelmässä.

Tällä hetkellä yleisimmät ohjelmointirajapinnat ovat http-pohjaisia. Suurin osa http-pohjaisista rajapinnoista on REST-pohjaisia, eli asiakas lähettää http-pyynnön ja saa siihen verkkolaitteelta strukturoidun vastauksen. Vastauksen rakenne on usein JSON- tai XML-muodossa. Ohjelmisto-ohjatuissa verkoissa vastaus ei tule suoraan verkkolaitteelta, vaan kontrollerilta.

Laitetoimittajat tarjoavat osalle verkkolaitteistaan http-pohjaisia rajapintoja. Se mitä rajapinnalla pystytään käsittelemään, vaihtelee myös laitekohtaisesti. Kuvasta 3 nähdään Aruba ProCurve -kytkimelle REST-rajapinnan esimerkki, jossa työasema lähettää standardin http POST -pyynnön kytkimelle sisältäen strukturoidussa JSON-muodossa tarvittavat parametrit uuden virtuaalilähiverkon luomiseksi. Kytkin taas palauttaa tilatiedon tehdystä konfiguraatiomuutoksesta vastaavanlaisesti JSON-muodossa.

Creating VLAN using REST:

```
WorkStation# curl --noproxy10.100.167.104 --cookie
"sessionId=09CG1bRuT5hkCPzI97mmDjpn4uLtsmgkBsAaWUr9h7GxlkbsiASak1PEyj7Ov3n" -X POST
http://10.100.167.104:80/rest/v1/vlans -d '{"vlan_id":5, "name":"VLAN5"}'
```

Response from the Switch:

```
{
    "uri": "/rest/v1/vlans/5",
    "vlan_id": 5,
    "name": "VLAN5",
    "status": "VS_PORT_BASED",
    "type": "VT_STATIC",
    "is_voice_enabled": false,
    "is_jumbo_enabled": false,
    "is_dsnoop_enabled": false
}
```

Kuva 3. Virtuaaliverkon luonti HP Aruba REST API:lla [23].

Http-pohjaisten rajapintojen avulla kuvan 3 esimerkin toimintaa voitaisiin laajentaa esimerkiksi ohjelmoijan tekemällä verkkoportaalilla, jonka läpi pystyttäisiin tekemään muutoksia puhtaasti selainpohjaisesti. Hyödyllisyyden aste riippuu tosin ympäristön laitekannasta ja tuesta rajapinnoille.

3.3 NETCONF-protokolla

NETCONF on verkonhallintaprotokolla, joka tarjoaa menetelmät konfiguraatioiden käsittelyyn ja on määritelty RFC 6241:ssä standardiprotokollaksi. Sen eräänä pohjustavana ideana on tuoda parempi vastine huomattavasti vanhemmalle verkkostandardi SNMP:lle, ja NETCONF:n perustavanlaatuinen toimintakin on huomattavasti SNMP:n kaltaista. [12; 24.] SNMP esimerkiksi käyttää hyödykseen MIB:ejä (Management Information Base) eri laitteiden ominaisuuksien havaitsemiseen, kun taas NETCONF:n mallissa käytetään sille suunniteltua datamallinnuskieltä YANG:a vastaavan toiminnallisuuden saamiseksi [12; 25].

Pinnan alla siis yhtenäinen datamalli eri laitetoimittajien ja niiden mallien kanssa toteutetaan YANG:n avulla, kun taas itse kommunikoinnin hoitaa NETCONF. Protokollan yhtenä ydinominaisuutena on mallintaa konfiguraatioiden syöttötavat eri datavarastoihin, eli toisin sanoen konfiguraatiotiedostoihin. Käytettävinä datavarastoina on suoraan suoritukseen syötettävä konfiguraatio, käynnistyksessä ajettava konfiguraatio ja ehdokaskonfiguraatio. [26; 27.] Tätä mallia voidaan verrata suoraan vaikkapa Ciscon ja Juniperin

eriäviin tapoihin käsitellä konfiguraatiota. Suurimmassa osassa Ciscon laitekantaa tehdyt konfiguraatiomuutokset menevät suoraan ajoon, ja ne kopioidaan haluttaessa käynnistyksessä suoritettavaan konfiguraatioon. Juniperin tapauksessa ajonaikainen muutos menee ensin ehdokkaaksi erilliseen tiedostoon ja astuu voimaan vasta, kun se erillisellä komennolla liitetään ajoon ja pysyvään konfiguraatioon. Toisena ydintoimintana on transaktiopohjaisuus. Jos muutoksien teossa tulee esimerkiksi virhe kesken syötön, keskeytetään koko suoritus ja jo läpi menneet muutokset vedetään pois. [12; 26.]

Kaiken kaikkiaan NETCONF tarjoaa verkkoautomaatiolle SNMP:tä kehittyneemmän ja turvallisemman protokollatasolla toimivan tavan hallita verkkolaitteita laiteriippumattomasti. Vaikka se pohjimmiltaan ei ole tuore protokolla, se elää jatkuvassa kehityksessä. Sen pohjalta on esimerkiksi vuonna 2017 valmistunut RFC-8040-standardi RESTCONF, jolla NETCONF-protokollan toiminnallisuus on laajennettu toimimaan http-pohjaisesti. [27.] Verrattaessa kuvan 3 valmistajan omaan rajapintaesimerkkiin saadaan RESTCONF:n avulla yksi kerros abstraktiota lisää, kun ei tarvitse hallita laitetoimittajien rajapintoja erikseen.

4 Testaukseen valittu automaatioalusta ja sen toiminnallisuudet

Halusin tehdä käytännön testejä alustalla, joka kykenisi käsittelemään erityisesti Ciscon tai HP Aruban yleisimpiä tuoteperheitä, koska yrityksessä, jossa työskentelen, nämä alustat dominoivat laitemääriltään asiakasympäristöjä. Lisäksi riittävä dokumentaatio ja kypsyysaste olivat tärkeitä perusteita. Viimeistään keväällä 2017 yrityksen ottaessa Ansiblen pienimuotoisesti käyttöön tuotantoympäristössä oli valinta selkeä.

Ansible on yksi tuoreista automaation ja konfiguraationhallinnan alustoista. Alun perin se on julkaistu vuonna 2012 avoimena lähdekoodina. Samassa segmentissä varteenotettavimmat vastineet ovat Puppet, Chef ja Salt Stack. Ansible on lyhyessä ajassa kerännyt suosiota yksinkertaisuutensa vuoksi. Pohjimmiltaan suorituksessaan Ansible tarvitsee vain sopivan protokollan kommunikaatioon kontrollikoneen ja hallittavan laitteen kanssa, Python-kirjaston vähintään versiolla 2.6 sekä riittävät suoritusoikeudet. [28.] Kuitenkaan asia ei ole niin yksiselitteinen verkkolaitteiden suhteen, sillä näissä laitteissa ei ole välttämättä mahdollisuutta suorittaa Python-skriptejä, eivätkä ne tarjoa mitään valmista rajapintaa.

Kesäkuussa 2017 yli 2 700 eri ihmistä on osallistunut Ansiblen kehitykseen [29]. Pääasiallisesti Ansiblea on käytetty palvelinten käsittelyyn, mutta myöhemmissä versioissa on tullut mukaan suoraan tuki myös verkkolaitteille. Ansiblen pääasialliset ominaisuudet alustana jakaantuvat seuraaviin kategorioihin:

- konfiguraationhallinta
- provisiointi
- automaatio
- orkestrointi.

Rakenteeltaan Ansible on modulaarinen. Se koostuu toisiinsa sidoksissa olevista komponenteista, joista ydinkomponentteja on viisi. Pinnan alla toteutus on ohjelmoitu Pythonilla ja avoimen lähdekoodin vuoksi kuka tahansa voi kehittää itse laajennuksia oman tarpeensa mukaan. Ansible on kuitenkin suunniteltu helppokäyttöisyys mielessä niin, ettei esimerkiksi ohjelmointikokemusta välttämättä tarvita lainkaan. Käyn seuraavaksi läpi, miten Ansible toimii kokonaisvaltaisesti, sekä tarkemmin eriteltynä tärkeimpien komponenttien toiminnan.

4.1 Inventory

Inventory on käytännössä luettelo laitteista, joita Ansiblella on tarkoitus kontrolloida. Se on yksinkertaisimmillaan vain tekstitiedosto, johon listataan järjestelmän määräämällä tavalla laitteet, roolit ja ryhmät. Huomionarvoista on, ettei Ansiblessa ole tietokantavaatimusta laiteluettelolle, mikä omalta osaltaan lisää järjestelmän yksinkertaisuutta. Halutessaan myös ulkopuolisen tietokannan tai järjestelmän voi integroida toimimaan dynaamisena laiteluettelona. [30.] Laiteluetteloita voi olla enemmän kuin yksi halutun tarpeen mukaan, esimerkiksi projekti- tai asiakaskohtaisesti.

Laiteluettelossa ohjattavia laitteita voidaan luetteloida DNS-nimipohjaisesti, tai suoraan IP-osoitteella ympäristön ja tarpeen mukaan. Koneet voidaan ryhmitellä rooliensa mukaisesti, koneet voidaan ryhmittää ja ryhmiä voi sitoa toisiin ryhmiin. Alla olevassa tekemässäni esimerkissä käydään hieman läpi yksinkertaisen laiteluettelon rakennetta,

jossa ristikkomerkki kuvastaa kommenttia:

```
[core]
Coreswitch1
Coreswitch2

[access]
AccessSwitch1
AccessSwitch2
AccessSwitch[03:10] #Listataan yhdellä rivillä 7 laitetta

[campus:children] #campus-ryhmä sisältää yo. ryhmät
core
access

[campus:vars] #Esimerkkimuuttujat. Kohdistuu kaikkiin ryhmiin
ansible_ssh_user=testi
ansible_ss_pass=testi
```

Laiteluettelolla voidaan melko helposti ryhmitellä laitteet roolinsa ja siinä voidaan rakenetaa ryhmiä, jotka pitävät sisällään toisia ryhmiä. Esimerkissä `[campus:children]` pitää sisällään muut laiteluettelon ryhmät. Laite- tai ryhmäkohtaisille muuttujille luettelo on vain yksi vaihtoehto muiden joukossa; esimerkissä käytetty muuttuja pystytään esimerkiksi sijoittamaan erilliseen muuttujatiedostoon.

Koska laajassa tuotantoympäristössä staattisen luettelon ylläpitäminen saattaa muodostua hankalaksi, Ansible tarjoaa mahdollisuuksia ulkopuoliseen luetteloon. Esimerkiksi suurimpien pilvipalvelujen tarjoajien, kuten Amazonin EC2 (Elastic Compute Cloud) -pilvestä voidaan generoida luettelo valmiilla skriptillä. [30.] Jos valmis skripti puuttuu, on mahdollista toteuttaa se itse. Tällöin skriptin on palautettava laitelistaus JSON-muodossa. [31.]

4.2 Moduulit

Moduuleja voi kutsua pieniksi lisäohjelmiksi Ansiblen sisällä, ja ne tekevät käytännössä kaiken konkreettisen työn. Yhdellä moduulilla on tietty tehtävä, kuten tietyn paketinhallinnan käyttö Linux-distribuutiossa, tai tietyn laitevalmistajan syntaksilla komentojen ajo. Pohjimmiltaan moduulit ovat skriptejä, jotka suorituksen lopussa palauttavat Ansiblelle

tietoa JSON-muodossa. Moduuleita voi käyttää suoraan komentoriviltä tai Playbookeissa (Ks. luku 4.3).

Ansiblessa moduulit on jaoteltu ydinmoduuleihin ja kolmannen osapuolen moduuleihin. Ydinmoduulit tulevat suoraan perusasennuksen mukana, ja niitä on sadoittain. Kuka tahansa voi kehittää oman moduulin, ja se voidaan lisätä ydinmoduuliksi, jos yhteisö sen hyväksyy.

Moduulit vaativat lähtökohtaisesti vähintään yhden argumentin suorituksen onnistumiseen. Esimerkkinä tarkastellaan Ciscon IOS-käyttöjärjestelmään tarkoitettua ios_interface-moduulia. Tällä moduulilla voidaan tarkastaa ja tehdä muutoksia IOS-laitteiden portteihin. Minimivaatimuksena moduulilla on yksi argumentti, joka tässä tapauksessa on halutun portin, tai porttien määritys. [32.] Tekemässäni esimerkissä moduuli käy laiteluettelon perusteella tarkistamassa portin olemassaolon, lisää siihen kuvauksen ja käy kytkemässä sen päälle tarvittaessa:

```
tasks:
  - name: Verify interface state
    ios_interface:
      name: GigabitEthernet0/3
      description: Active port
      state: up
```

4.3 Playbook

Playbookit ovat Ansiblen toimintalogiikan ydin. Niiden avulla orkestroidaan ja kontrolloidaan haluttu automaatiologiikka muiden komponenttien avulla. Esimerkiksi kun moduuli suorittaa jonkin tietyn tehtävän laitteessa, playbookissa määritellään mille laitteille se tehdään, ja millä ehdoilla. Yksinkertaisimmillaan playbookissa voidaan suorittaa vain yksi tehtävä, tai vastavuoroisesti sillä voidaan suorittaa hyvin komplekseja ajastettuja tehtäviä tietyssä järjestyksessä.

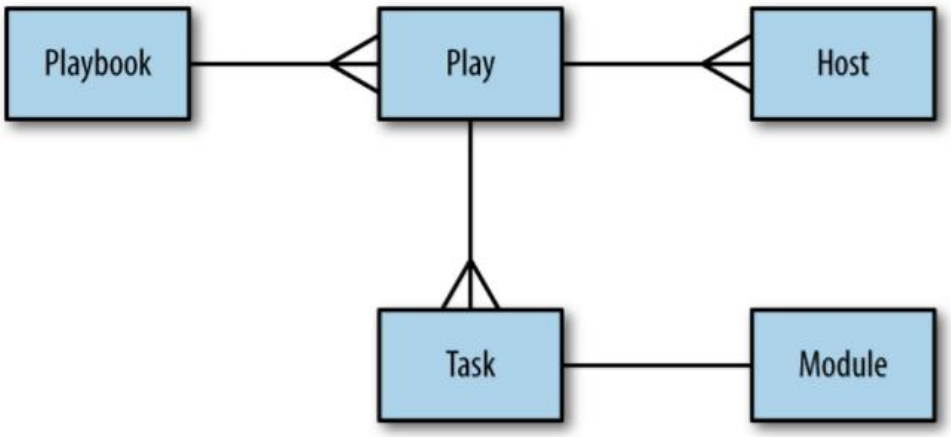
Playbookit noudattavat YAML (Yaml Ain’t Markup Language) -syntaksia, joka on itsessään ihmiselle helppolukuiseksi suunniteltu datan serialisointikieli. Data esitetään avain-arvopareina. Ansible käyttää YAML:a juurikin sen helppolukuisuuden vuoksi. [33.]

Kuvasta 4 nähdään YAML:lle ominaisia syntaksielementtejä. Kolmella viivalla ilmoitetaan YAML-dokumentin alku. Yksittäisellä viivalla määritetään sekvenssi tai lista. Data itsessään pidetään avain-arvoparimuodossa, joka syntaktisesti kuvataan kaksoispisteellä. Välimerkit kuvaavat hierarkiaa syntaksissa: kun katsotaan esimerkiksi kuvan 4 `vars`-arvoa, huomataan sen arvona olevan toinen avain `cli`, jonka arvona taas seuraavat kaksi riviä ovat. Spesifikaation mukaisesti aina kaksi välimerkkiä on oikea sisennystapa. Sarkainmerkkiä ei saa käyttää, sillä se tuottaa eri ympäristöissä eri määrän välejä [34].

```
67
68  ---
69
70  - name: Gather interfaces list
71    hosts: switches
72    connection: local
73    gather_facts: False
74
75    vars:
76      cli:
77        username: inssi
78        password: inssi
79
80
81    tasks:
82      - name: Get interfaces
83        ios_facts:
84          provider: "{{ cli }}"
85          authorize: yes
86          gather_subset:
87            - interfaces
88
89        register: ruututuloste
90      - debug: var=ruututuloste
91
92
```

Kuva 4. YAML-syntaksi playbookin datarakenteella.

Playbookit sitovat yhteen muut ydinkomponentit, moduuleja käytetään tehtävissä playbookin sisällä ja laiteluettelon perusteella valitaan kohteet, joille tehtävät suoritetaan. Kuvasta 5 nähdään Ansiblen ydinkomponenttien suhteet toisiinsa. Esimerkiksi playbookissa on monta erillistä suoritusta, jotka kohdistuvat mahdollisesti useaan laitteeseen.



Kuva 5. Ansiblen ydinkomponenttien ER-kaavio (Entity-Relationship model) [35].

Rakenteellisesti playbook on lista erillisiä tehtäviä, jotka ajetaan järjestyksessä. Minimivaatimuksena on vähintään yksi laite laiteluettelossa, sekä vähintään yksi tehtävä. Kontrollikone itsessään voidaan sisällyttää laiteluetteloon.

Yksinkertaisten ajojen lisäksi Ansible tarjoaa vaihtoehtoja muun muassa paremman abstraktion ja dynaamisuuden saavuttamiseksi. Tärkeimpinä lisätyökaluina ovat roolit ja mallipohjat. Roolien avuilla voidaan tuoda skaalautuvuutta automaatiologiikkaan jakamalla playbookit pienempiin osiin. Esimerkkinä voidaan ajatella vaikkapa ison organisaation toimipistekytkimiä. Näillä kytkimillä on usein paljon identtisiä pohjakonfiguraatioita, mutta myös toimipistekohtaisia eroavaisuuksia. Lisäksi osa kytkimistä voi olla reunakytkimiä ja osa reitittäviä runkokytkimiä. Tällöin voidaan tehdä erilliset roolit pohjakonfiguraatioille, reunakytkimille ja runkokytkimille. Jatkoesimerkkinä uutta toimintaa lisättäessä pystytään helposti kierrättämään sisältöä pienemmistä palasista.

Mallipohjat ovat olleet mahdollisia Ansiblessa versiosta 0.5 lähtien, kun Python-kielelle suunniteltu mallipohjamoottori Jinja2 otettiin siinä käyttöön [36]. Jinja lisättiin tuomaan tukea uudelleenkäytettävälle tekstille ja yleisesti laajentamaan Ansiblen toiminnallisuutta ohjelmoinnissa käytetyillä käsitteillä, kuten silmukoilla ja muuttujilla. Jinjaa voi käyttää hyödyksi niin playbookien sisällä suoraan kuin erillisinä tiedostoina. Suoraan playbookissa Jinja-toiminnallisuudessa on kuitenkin rajoitteita, kuten silmukoiden puuttuminen. Kuvassa 4 on myös mukana Jinjan toiminnallisuutta: rivillä 84 osio `"{{ cli }}"` toimii muuttujana, joka tässä tapauksessa saa arvokseen salasanan ja käyttäjän riveiltä 77—78.

5 Testiympäristön asennus

5.1 Ympäristön yleiskuva

Päätin toteuttaa testiympäristön avoimen lähdekoodin GNS3-verkkosimulaattorilla. Simulaattorissa itsessään käytetään lisäksi GNS3:n omaa Ubuntu Linux -virtuaalikonetta verkkolaitteiden levykuvien suorittamiseen, sillä fyysisen työaseman pohjimmaisena käyttöjärjestelmänä toimii Windows, joka ei pysty suorittamaan natiivisti monia Ciscon verkkolaitteiden levykuvia. Lisäksi se on kehittäjän suosittelema tapa. [37.] Simuloidut verkkolaitteet ja Ansiblen virtualisoitu kontrollikone on kaikki liitetty suoraan GNS3-projektiin. Hypervisorina virtuaalikoneille toimii VMware Workstation. Yksinkertaisimmista välivaiheista on saatettu jättää pois esimerkiksi komentosyötteet, jotta ydinsisältö pysyy selkeämmin hahmoteltavana.

GNS3

GNS eli Graphical Network Simulator 3 toimii testiympäristön kulmakivenä. Sen avulla voidaan emuloida ja simuloida lukuisten eri verkkolaitevalmistajien laitteita samassa ympäristössä. Samaan ympäristöön voi myös lisätä virtuaalikoneita. Käyttötarkoituksena voivat olla esimerkiksi PoC-ympäristöt (Proof of Concept), vianratkonta tai implementointitestaus. Työssä käytetään pääversiota 2.0.0.

VMware Workstation

VMware workstation on virtualisointiohjelma työpöytäkäyttöön. Se on tyypin 2 hypervisori, eli sitä käytetään olemassa olevan käyttöjärjestelmän päällä. VMware Workstation toimii x86-suoritinarkkitehtuurilla eli suurimmalla osalla nykyaikaisista käyttöjärjestelmistä. Työssä käytetään versiota 12 Pro.

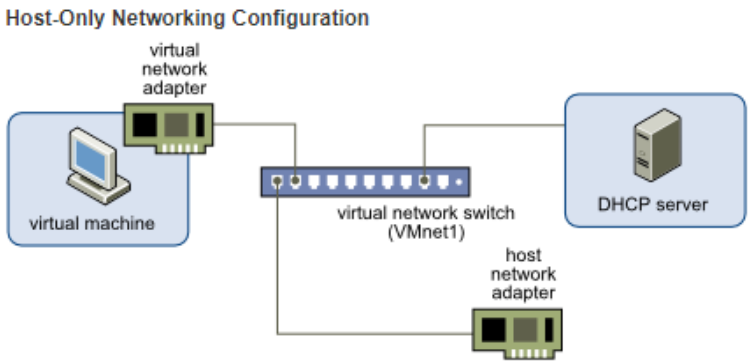
5.2 GNS3-konfigurointi

Itse GNS3 on sovelluksena hyvin suoraviivainen, mutta testiympäristöä varten vaaditaan hieman konfigurointia. GNS3 tarjoaa valmiin levykuvan tarvitsemaansa virtuaalikonetta varten. Levykuva tuodaan VMwareen ja sille allokoidaan riittävät resurssit. Vähimmäis-

vaatimuksena on yksi virtuaalinen prosessoriydin ja kaksi gigatavua allokoitua keskusmuistia. Vaaditut määrät nousevat topologian koon perusteella. Virtuaalikoneelle on asetettu kaksi verkkokorttia, jotka käyttävät VMware Workstationin verkkoja.

VMnet1 / Host Only

VMnet1 on standardi verkko VMwaressa. Tämän verkkorajapinnan kautta työasema ja virtuaalikoneet kommunikoivat keskenään kuvassa 6 havainnollistetun virtuaalikytkimen avulla. Verkko on täysin suljettu, joten kommunikaatiota on pelkästään verkossa olevien virtuaalikoneiden ja fyysisen työaseman välillä. Verkko sisältää perustoiminnallisuudessaan myös DHCP (Dynamic Host Configuration Protocol) -palvelimen, joka jakaa vakiona osoitteita väliltä .128—.254. Työasemalle on allokoitu osoite 192.168.160.1 ja GNS-virtuaalikoneelle 192.168.160.128.

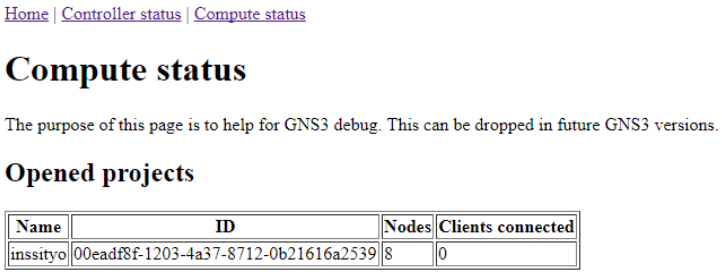


Kuva 6. VMnet1-verkon havainnekuva [38].

VMnet8 / NAT

NAT-verkkoa tarvitaan, jos koneella on esimerkiksi tarve kommunikoida internetiin päin. Verkossa on toteutettuna osoitteenmuunnos NAT (network address translation), joten ulkoa päin pyynnöt näkyvät tulevan fyysisen työaseman osoitteesta. Verkko on hyvin samankaltainen kuin VMnet1. Erotuksena on kuitenkin tästä verkosta löytyvä oletusyhdyskäytävä, joka toimii reittinä ulkoverkkoon ja hoitaa osoitteenmuunnoksen. Verkko on vakiona 192.168.32.0/24 ja osoitteistus identtinen VMnet1:n kanssa.

Virtuaalikone itsessään liitetään GNS3:een yksinkertaisesti asetuksista. Vakioasetuksillaan virtuaalikone käynnistetään automaattisesti GNS3:n käynnistyksen myötä. Virtuaalikone suorittaa vakiona web-palvelinta TCP-portissa 3080, josta voidaan tarkistaa kuvan 7 mukaisesti koneen tila GNS3:n suhteen.



Kuva 7. GNS3-virtuaalikoneen tila. Kuvassa näkyy avoin GNS-projekti ja topologian koko.

Toimivan GNS3-virtuaalikoneen lisäksi pitää ympäristöön liittää topologioissa käytettävät verkkolaitteet. GNS3:lla on sivustollaan valmiit mallipohjat tukemiinsa verkkolaitteisiin. Tässä tapauksessa käytin Ciscon IOSv-levykuvia, joita yleisesti käytetään Ciscon VIRL (Virtual Internet Routing Lab) -ympäristössä. Pohja määrittelee valmiiksi asetukset levykuvalle. Aiemmissa GNS3-versioissa levykuva piti siirtää työasemalta virtuaalikoneelle esimerkiksi FTP:n avulla, mutta nykyään se pystytään siirtämään suoraan pohjan lisäämisessä.

5.3 Kontrollikoneen konfigurointi

Ansiblen hallintakoneeksi asennettiin täysin vakio Ubuntu-distribuutio versiossa 16.04. Koneelle on asetettu kaksi verkkokorttia:

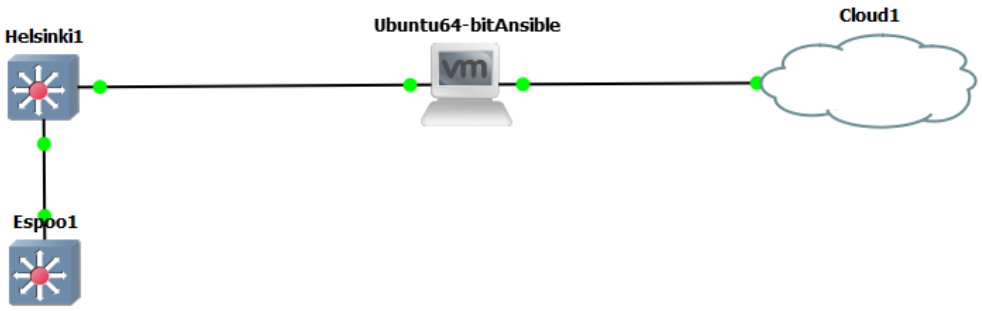
```
auto ens33
iface ens33 inet dhcp
gateway 192.168.32.2
up route add -net 0.0.0.0 netmask 0.0.0.0 gw 192.168.32.2

auto ens38
iface ens38 inet static
address 10.0.0.5
netmask 255.255.255.0
gateway 10.0.0.1
```

Verkkoliitäntä `ens33` on liitettynä VMnet8-virtuaaliverkkoon, jotta kontrollikone pystyy lataamaan tarvittavat työkalut. Liitännälle on asetettu myös staattinen oletusreitti osoittamaan verkon yhdyskäytävää. Toinen verkkoliitäntä `ens38` taas on kytkettynä suoraan GNS3-topologiaan, jota tullaan käyttämään verkkolaitteiden kommunikaatioon.

Koneelle asennettiin tarvittavat sovellukset, kuten SSH-palvelin (Secure Shell) ja Ansible. Asennukset ovat suoraviivaisia ja helppoja, sillä ne pystytään hakemaan suoraan Ubuntun paketinhallinnan kautta.

5.4 Testitopologian määritys

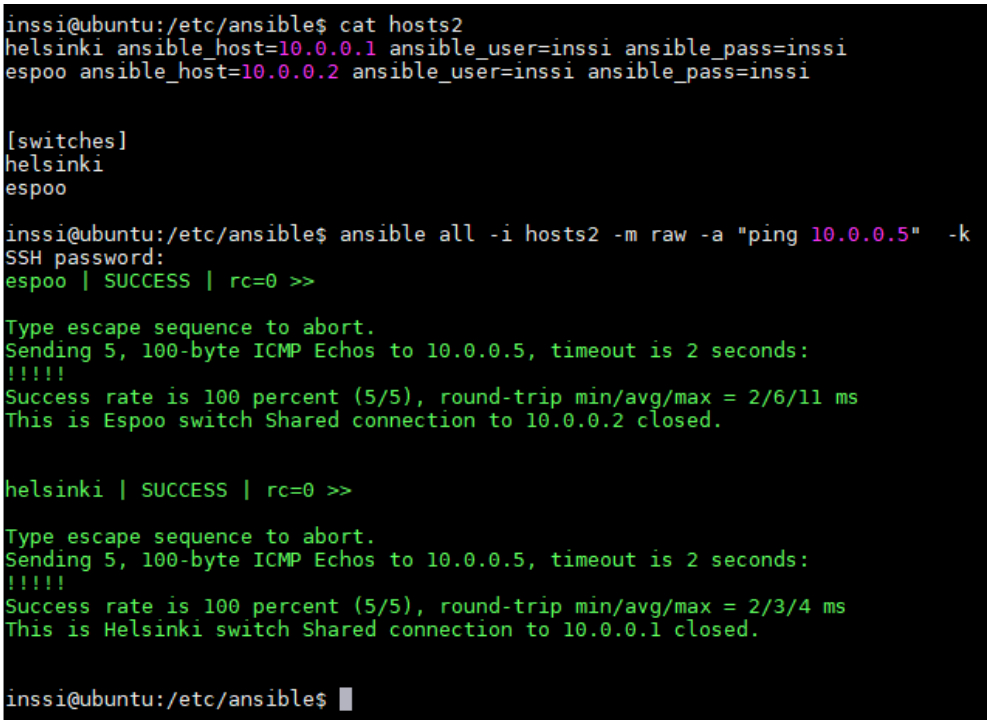
Alustavaan topologiaan on määritelty kuvan 8 mukaisesti kontrollikone ja muutama Cisco kytkin. Lisäksi kontrollikone on kytketty GNS3:n pilvinoodiin, joka on tässä tapauksessa vain abstraktiokerros sidottuna VMware Workstationin VMnet8-verkkoon.



Kuva 8. Testitopologia GNS3-ympäristössä.

Kytkimille tarvitaan seuraavat peruskonfiguraatiot ennen kuin niitä voidaan hallita Ansiblen avulla: SSH-protokollan määritys ja hallintaosoite.

Esikonfiguraation jälkeen varmistetaan yhteyden toimivuus kytkinten ja kontrollikoneen välillä Ansiblen avulla kuvan 9 mukaisesti. Ansible ottaa yhteyden kontrollikoneelta kytkimiin, jotka taas lähettävät suoritetun komennon tulosteen takaisin kontrollikoneelle.

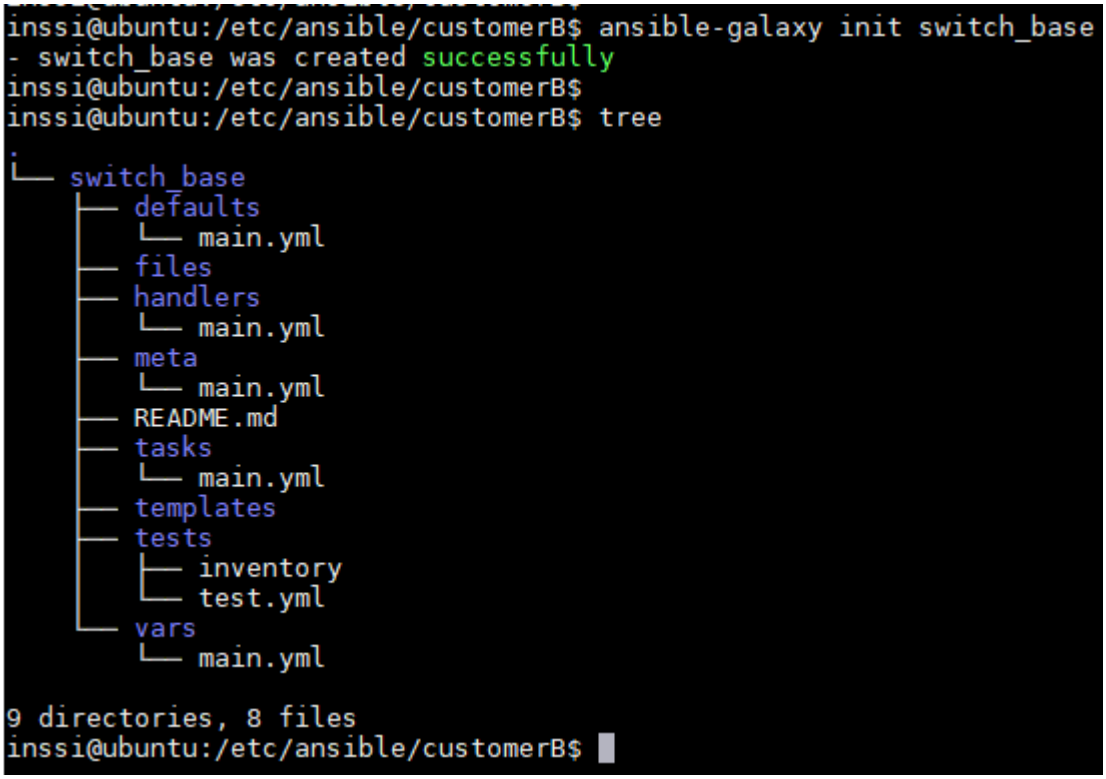


Kuva 9. Kommunikaatiotesti kontrollikoneen ja verkkolaitteiden kesken.

6 Automaatiologiikan testaus verkkolaitteilla

6.1 Roolin luonti ja konfigurointi

Päätin testata, miten Ansiblen roolit ja mallipohjat toimivat käytännössä. Tarkempana toimintasuunnitelmana päätin toteuttaa yhden roolin, jonka avulla pystyisi luomaan koneellisesti konfiguraatiota tavallisille kytkimille, ja sitä voisi jatkojalostaa eri asiakkuuksiin tai ympäristöihin. Aluksi loin roolille valmiin hakemistopohjan. Ansible tarjoaa valmiin komennon `ansible-galaxy init <roolin_nimi>` hakemistorakenteen luomista varten. Kuvassa 10 listatuista kansioista tärkeimmät testin kannalta ovat `tasks`, `templates` ja `vars`.



Kuva 10. Hakemistorakenteen luonti ja rakenne.

`Templates`-hakemistoon luodaan kaikki roolin alaiset mallipohjat, joita tässä tapauksessa tein neljä. Mallipohjassa `switch_baseline.j2` on lähinnä staattista pohjakonfiguraatiota Ciscon syntaksilla, mutta tässäkin tapauksessa on hyvä asettaa ympäristökohtaisia muuttujia uudelleenkäytön helpottamiseksi:

```
hostname {{ item.hostname }}
ip domain-name {{ general.domain_name }}
```

Muissa mallipohjissa taas on enemmän ohjelmointikielien kaltaista toiminnallisuutta, kuten kuvasta 11 nähdään `access_ports_baseline.j2` mallipohjan sisältö ja kuinka se muodostuu jo hyvin kierrätettäväksi toisiin ympäristöihin. Käytännössä tämä mallipohja iteroi eri porttien välillä ja täydentää ne muuttujien arvoilla. Huomionarvoisia ovat ehtolauseet, jotka on tehty sallimaan kytkinporteille tyypillisiä vaihtoehtoja, kuten linkkien yhdisteleminen nippuun.

```
inssi@ubuntu:/etc/ansible/customerB/roles/switch_base/templates$ cat access_ports_baseline.j2
{% for item in access_ports -%}

{% for access in item.ports %}
{% if "-" in access|string %}
interface range {{ access }}
{% else %}
interface {{ access }}
{% endif %}
  switchport mode access
  switchport access vlan {{ item.vlan }}
  description {{ item.desc }}
  {% if item.group is defined -%}
  channel-group {{ item.group }} mode on
  {% endif -%}
  no shutdown
{% endfor %}

{% endfor %}
```

Kuva 11. Mallipohja reunaporttien konfigurointiin.

Mallipohjat pystyisi toki pitämään yhdessä pitkässä tiedostossa, mutta erotteleminen toisaalta tuo hienojakoisuutta. Jakamalla pohjat pienempiin osiin, pystytään pieniä palasia käyttämään paremmin yksitellen ja yksittäisen pohjan kehittäminen ja testaus on helpompaa. Tein kuvaa 11 vastaavat toteutukset myös runkoporteille ja virtuaaliverkoille.

Muuttujat taas määriteltiin tässä tapauksessa sekä roolin alle (`vars`) että projektin juurihakemiston alle `/etc/ansible/customerB/group_vars` (ks. kuva 15). Molemmat ovat hakemistoja, joista Ansible etsii muuttujia automaattisesti. On käyttäjän päätettävissä, kuinka muuttujat sijoitetaan, mutta esimerkiksi suoriksi roolimuuttujiksi ei kannata sijoittaa mitään liian yksilöiviä muuttujia, koska se rampauttaa roolia uudelleenkäytön suhteen.

Kuvan 12 perusteella nähdään, miten mallipohjaa (rivit 14—17) pystyttäisiin uudelleenkäyttämään missä tahansa Cisco-pohjaisessa ympäristössä. Esimerkiksi mallipohja tarvitsee muuttujan `id`, jonka se saa muuttujaksi määritetyltä avain-arvoparilta `vlans.yml`-tiedostosta. Sisältö voi vaihdella, kunhan se pystyy tarjoamaan mallipohjaan määritetyn avaimen. Mainittakoon myös, ettei `subnet`-avaimen arvoja käytetä tällä hetkellä missään, vaan ne on lisätty esimerkin vuoksi. Avainta pystyisi jatkojalostaa vaikkapa staattisen reitityksen muuttujana. Riippuen halutusta lopputilanteesta tulisi muuttujat mahdollisesti sijoittaa `group_vars`-hakemistoon tai laiteluetteloon, jotta muuttujat olisivat ennemminkin playbook-kohtaisia kuin roolikohtaisia.

```
1
2   inssi@ubuntu:/etc/ansible/customerB/roles/switch_base/vars$ cat vlans.yml
3   ---
4
5 ▼ vlans:
6     data: {id: '200', subnet: 10.200.0.0/24, desc: Data Vlan}
7     voice: {id: '250', subnet: 10.250.0.0/24, desc: Voice Vlan}
8     guest: {id: '300', subnet: 10.300.0.0/24, desc: Guest Vlan}
9     poc: {id: '350', subnet: 10.350.0.0/24, desc: POC Vlan}
10
11
12  inssi@ubuntu:/etc/ansible/customerB/roles/switch_base/vars$ cat ../templates/vlans_baseline.j2
13
14  {% for key,value in vlans|dictsort %}
15  vlan {{value.id}}
16    name {{ key }}
17  {% endfor %}
```

Kuva 12. Muuttujien ja mallipohjan relaatiot.

Viimeisenä roolin kannalta tärkeänä asiana tein kuvan 13 mukaisen toteutuksen roolin tehtävähakemistoon `tasks`. Kuvassa näkyvä tiedostonimi `main.yml` on minimivaatimus, joka toistuu muissakin roolin kansioissa kuvan 11 mukaisesti. Tiedoston käyttötarkoitus riippuu hieman siitä, minkä hakemiston alla se on, mutta yleisesti se toimii aloituspisteenä jonka Ansible aina tarkistaa.

```
inssi@ubuntu:/etc/ansible/customerB/roles/switch_base/tasks$ cat main.yml
---

- name: Add variables
  include_vars:
    dir: 'vars'

- name: Generate baseline configurations
  template: src=switch_baseline.j2 dest=/etc/ansible/customerB/configs/{{item.hostname}}.txt
  with_items: "{{ switches }}"

- name: Generate customer specific VLAN's
  template: src=vlans_baseline.j2 dest=/etc/ansible/customerB/configs/vlan_baseline.txt
  with_items: "{{ vlans }}"


- name: Generate customer specific access port layout
  template: src=access_ports_baseline.j2 dest=/etc/ansible/customerB/configs/access_ports.txt

- name: Generate customer specific trunk port layout
  template: src=trunk_ports_baseline.j2 dest=/etc/ansible/customerB/configs/trunk_ports.txt
```

Kuva 13. Roolin suorittamat tehtävät.

Roolin tehtävänä on luoda mallipohjien ja muuttujatiedostojen avulla konfiguraatiot tekstimuodossa, jotta niitä pystytään käyttämään Ciscon verkkolaitteille suunnatussa `ios_config`-moduulissa. Yritin melko pitkään päästä irti välivaiheesta, jossa luodaan mallipohjan kautta ensin erillinen tekstitiedosto. Ilmeisesti kuitenkin tälle moduulille ei voida syöttää suoraan ajonaikaisesti mallipohjaa ja muuttujia.

Koko rooli pystytään ottamaan käyttöön halutuissa playbookeissa, mutta tällöin täytyy ottaa huomioon `main.yml`-tiedoston suoritus, sillä se suoritetaan automaattisesti roolia kutsuttaessa, mikä saattaa olla epätoivottua.

6.2 Roolin testaus

Lisäsin testiympäristöön kolme kytkintä, jolloin niitä oli kaikkiaan viisi kappaletta. Koostin konfiguraation suoritusta varten kuvassa 14 näkyvän playbookin nimeltä `push_all.yml`, joka suoritti roolin kaikkien mallipohjien toimintaa. Lopulta kytkimiin luotiin playbookin pohjalta virtuaalilähiverkkoja, yleinen pohjakonfiguraatio sekä runko- ja reunaporttien konfiguraatiot.

Playbookin toimintaan saattaminen ei ollut kuitenkaan ongelmatonta. Järjestelmän vikojen selvittämisen haasteellisuus tulee myös ottaa huomioon alustan käyttöönoton harkinnassa. Ensimmäiseksi olin unohtanut määrittää uusille kytkimille käyttäjätunnuksen

oikein, jolloin suorituksen aikana sain virheilmoituksena hyvin yleisluonteisen yhteysongelmaviestin:

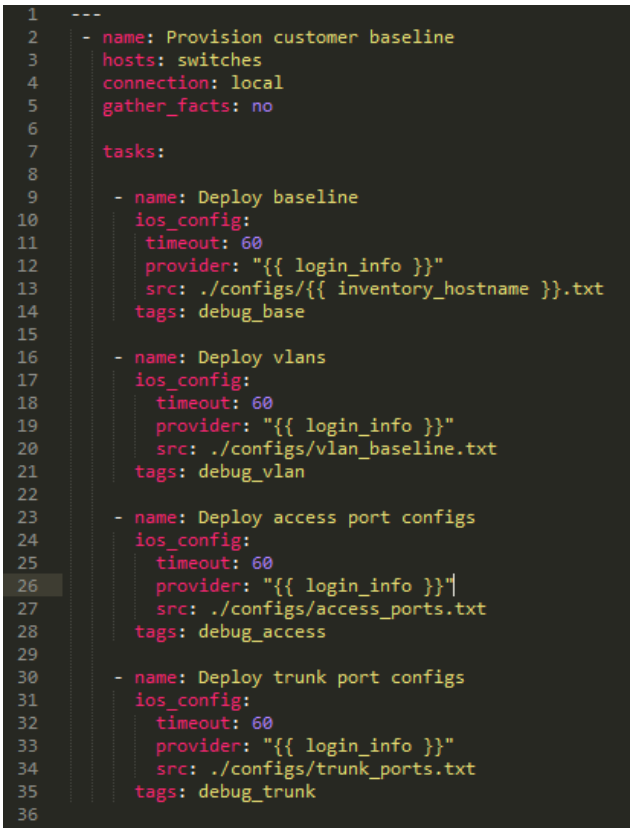
```
fatal: [Vantaa]: FAILED! => {
"changed": false,
"failed": true,
"msg": "unable to open shell.
```

Virheviesti ei tässä tapauksessa yksilöi yhteysongelmaa ollenkaan ja ilmoittaa vain suorituksen epäonnistuneen yhteysongelmiin liittyen. Tässä tapauksessa ongelman selvitys vaatisi erillistä lokitietojen keräämistä sekä sen analysointia ja täten vaikuttaa hidastavasti järjestelmän käyttöön ja operointiin. Seuraava ongelma liittyi suorituksenaikaiseen aikakatkaisuun:

```
fatal: [Espoo]: FAILED! => {"changed": false, "failed": true,
"msg": "unable to retrieve current config", "stderr": "timeout
trying to send command: show running-config", "stderr_lines":
["timeout trying to send command: show running-config"]}
```

Vaikka vian ratkominen edellytti tässä tapauksessa Googlen käyttöä, se oli kuitenkin nyt huomattavasti yksilöivämpi. Tässä tapauksessa sain ongelman ratkottua lisäämällä moduulin aikakatkaisun maksimirajaa tehtäväkohtaisesti.

Kokonaisuudessaan playbook näkyy kuvasta 14. Päätin lisätä tunnisteita vianratkonnan vuoksi. Esimerkiksi rivillä 14 näkyvän `debug_base`-tunnisteen avulla pystyn suorittamaan pelkästään tehtävän `Deploy baseline`. Tässä tapauksessa lisäsin sen lähinnä vianratkonnan vuoksi, mutta isoissa playbookeissa tällä ominaisuudella voidaan tarpeen mukaan suorittaa vain haluttu osanen, ilman tarvetta tehdä kokonaan uutta playbookia yhden tehtävän vuoksi.

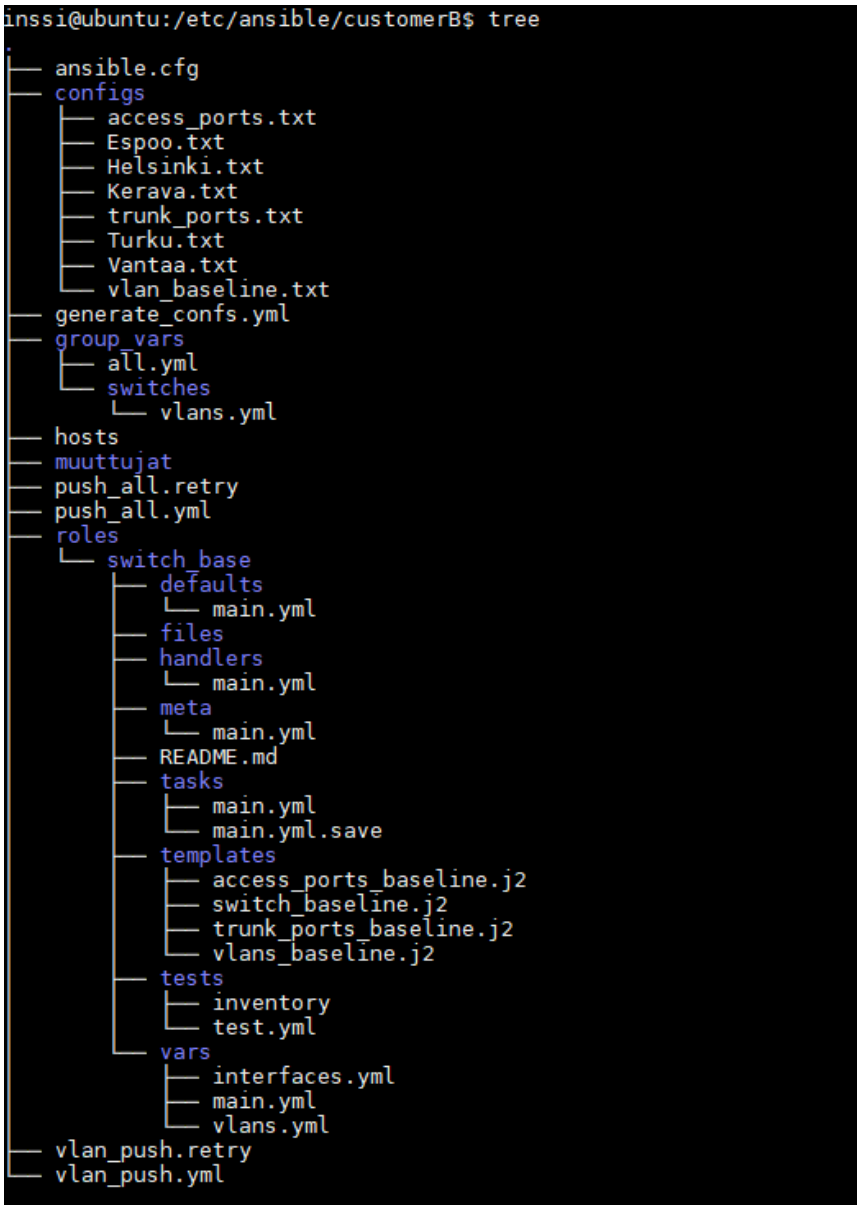


```
---
- name: Provision customer baseline
  hosts: switches
  connection: local
  gather_facts: no

  tasks:

   - name: Deploy baseline
     ios_config:
      timeout: 60
      provider: "{{ login_info }}"
      src: ./configs/{{ inventory_hostname }}.txt
     tags: debug_base

   - name: Deploy vlans
     ios_config:
       timeout: 60
       provider: "{{ login_info }}"
       src: ./configs/vlan_baseline.txt
     tags: debug_vlan

   - name: Deploy access port configs
     ios_config:
       timeout: 60
       provider: "{{ login_info }}"
       src: ./configs/access_ports.txt
     tags: debug_access

   - name: Deploy trunk port configs
     ios_config:
       timeout: 60
       provider: "{{ login_info }}"
       src: ./configs/trunk_ports.txt
     tags: debug_trunk

```

Kuva 14. Lopullinen testaukseen käytetty playbook.

Tekemäni playbookin suorittaminen kesti hieman päälle minuutin. Kestoon vaikutti tässä tapauksessa eniten kontrollikoneen heikko suorituskyky. Kontrollikoneelle on allokoitu yksi virtuaalinen prosessori ja fyysisen työaseman komponentit alkavat olemaan jo kuusi vuotta vanhoja. Ansiblen suhteen yhtaikaisia prosesseja voidaan määrittää suorituskyvyn ehdoilla lähes rajattomasti. Vakioarvona on viisi samanaikaista prosessia, ja se voidaan muuttaa `ansible.cfg`-tiedostosta.

Hakemistorakenne on mielestäni hyvin tärkeä ymmärtää Ansiblessa, varsinkin kun itse tein mahdollisesti epäoptimaalisia päätöksiä esimerkiksi muuttujien sijoittelemisessa. Kuvasta 15 nähdään lopullinen hakemistorakenne, jonka juurena toimii projektin kansio `/etc/Ansible/customerB`.



Kuva 15. Lopullinen hakemistorakenne.

Turhaa päänvaivaa syntyi kokonaisrakenteen hahmottamisessa ja muuttujien käytössä sijoiteltuani niitä eri paikkoihin. Ansiblen oma suosituskin on keskittää muuttujat samaan paikkaan. [39.] Mikään ei kuitenkaan estä samoja muuttujia olemasta eri paikoissa, sillä Ansiblella on tietty arvojärjestys, jonka pohjalta muuttuja valitaan.

6.3 Ympäristön jatkokehitys

Halusin testissä pysyttäytyä pelkästään Ansiblessa, mutta on hyvä mainita lyhyesti, miten automaatioalustan käyttöä voitaisiin parantaa täydentävillä järjestelmillä.

Versionhallinta

Työssä ei käytännössä ollut versionhallintaa, mutta oikeissa ympäristöissä koen sen olevan elinehto. Myöskään suoraan kontrollikoneella editoiminen ei ole kovin kestävää, joten järkevää olisi laittaa projektit esimerkiksi Git-pohjaiseen versionhallintajärjestelmään, kuten Gitlabiin. Tällöin muutoshistoriasta pysyisi täysi kirjanpito ja sivutuotteena varmuuskopiot sekä mahdollisuus jaettuun kehitykseen.

Graafinen käyttöliittymä

Komentorivipohjainen järjestelmän operointi ei myöskään ole kovin nopeaa ja kestävää, jos tähdätään tuotantoympäristöön. Graafisen käyttöliittymän avulla pystyttäisiin yleisesti hallinnoimaan ihmiselle helpommalla tavalla selainpohjaisesti projekteja ja töiden suorituksia, puhumattakaan siitä, kuinka paljon yhdestä näkymästä pystyy näkemään.

Red Hat on tarjonnut aiemmin omasta takaa Ansiblelle vain maksullista käyttöliittymää, joka kulkee nimellä Tower. Kuitenkin juuri työn loppuvaiheessa Red Hat julkaisi avoimen lähdekoodin vastineen AWX:n. [40.]

7 Yhteenveto ja pohdintaa

Testaamani automaatioalusta osoittautui käteväksi järjestelmäksi eritoten rutiininomaisten konfiguraatiomuutosten automatisointiin. Järjestelmä itsessään on suhteellisen helppo käyttää perustasolla, mutta luonnollisesti se muovautuu vaikeaselkoisemmaksi sen mukaan, mitä enemmän sillä halutaan tehdä. Mitä automatisoidumpi lopputulos on, sitä enemmän pohjatyötä vaaditaan järjestelmältä. Tässä tapauksessa kerran kunnolla tehty kuitenkin poistaa lopulta jatkossa ylimääräisiä työvaiheita päivittäisessä työssä. Yleisesti ottaen on käytettävä omaa harkintaa automaatiologiikan kehittämisen suhteen; onko automaatiota järkevä toteuttaa tietylle asialle, vai kannattaako se jatkossakin tehdä manuaalisesti. Esimerkiksi jos automatisoitavia kohteita on vähän ja muutoksia tehdään

hyvin harvoin, on järkevämpää pidättäytyä manuaalisessa työskentelyssä. Abstraktio ja uudelleenkäytettävyys muodostuvat tärkeiksi käsitteiksi, kun automaatiota toteutetaan laajemmassa mittakaavassa, sillä muovautuvalla ja laiteriippumattomalla automaatiologiikalla säästytään turhalta toistolta automaatioalustan hallinnan suhteen.

Yritystasolla tuotantoon valjastamista varten vaadittaisiin myös mahdollisesti haasteellisia lisätoimenpiteitä, kuten integraatio valmiin laitekannan kanssa. On toisaalta turhaa pitää redundanttista tietoa eri järjestelmissä, mutta järjestelmien yhteensovittaminenkin voi muovautua haasteelliseksi. Nykyisessä työssäni Ansible voitaisiin valjastaa ensimmäisenä koskemaan erityisesti tavallisia reunakytkimiä ja niiden työllistävimpiä rutiininomaisia muutoksia. Laitekanta ja rutiinityöskentelyn määrä on tässä tapauksessa mittava, jolloin hyödyllisyyden aste pystyttäisiin mittaamaan nopeasti.

Komentorivipohjainen laitekohtainen hallinta, joka on esimerkiksi omassa organisaatiossani yleinen tapa, ei mielestäni tulevaisuudessa kannata. Toisaalta tästä päästäänkin verkkolaitteiden tulevaisuuteen, jolloin mahdollisesti avointen rajapintojen ja ohjelmisto-ohjattujen verkkojen myötä verkkojen kanssa tekemisissä olevat henkilöt joutuvat kehittämään ohjelmointitaitoja tai vähintäänkin syventämään yhteistyötä ohjelmoijien kanssa. Ansiblen kaltainen alusta ei myöskään menetä rajapintojen ja ohjelmisto-ohjattujen verkkojen myötä tarkoitustaan, vaan sen käyttö luultavasti helpottuu esimerkiksi epäkäytännöllisen datan parsinnan muodostuessa tarpeettomaksi.

Olen kiinnostunut Ansible-alustasta sekä palvelimien että verkkolaitteiden hallinnoimiseen. Seuraavana askeleena on kehittää omaa ohjelmointiosaamista Pythonin avulla, jotta voisin kehittää mahdollisesti itse moduuleita tulevaisuudessa.

Lähteet

1 Mastomäki, Ville-Veikko & Stenhäll, Jaakko. 2014. Teknologinen murros ja politiikka. Verkkodokumentti. <https://teknologinenmurrosjapolitiikka.files.wordpress.com/2014/01/teknologinen-murros-ja-politiikka-lopullinen.pdf> Luettu 15.5.2017.

2 Frey, Carl Benedikt & Osborne, Michael A. 2013. The future of employment: How susceptible are jobs to computerisation? Verkkodokumentti. Oxford University. <http://www.oxfordmartin.ox.ac.uk/downloads/academic/The_Future_of_Employment.pdf>. Luettu 15.5.2017.

3 ZTP basic setup guide. Verkkodokumentti. Arista Networks Inc. <https://eos.arista.com/ztp-set-up-guide/>. Luettu 11.9.2017.

4 Donahue, Gary A. 2012. Arista Warrior. O'Reilly Media.

5 Chou, Eric. 2017. Mastering Python Networking. Packt Publishing.

6 Ogenstad, Patrick. Napalm introduction. Verkkodokumentti. Network Lore. <https://networklore.com/napalm-introduction/>. Luettu 12.9.2017.

7 Ulinic, Mircea. Network Automation with Salt and NAPALM. Verkkodokumentti. Cloudflare Inc. <https://www.nanog.org/sites/default/files/NANOG68%20Network%20Automation%20with%20Salt%20and%20NAPALM%20Mircea%20Ulinic%20Cloudflare%20(1).pdf>. Luettu 12.9.2017.

8 Ho, Grant. NetBrain Technologies Survey Reveals Efficiency and Collaboration Challenges Within Enterprise Networking Teams. Verkkodokumentti. NetBrains Technologies. < https://www.netbraintech.com/news/netbrain-technologies-survey-reveals-efficiency-and-collaboration-challenges-within-enterprise-networking-teams/>. Luettu 13.9.2017.

9 Lerner, Andrew. 2014. The Cost of Downtime. Verkkodokumentti. Gartner. <http://blogs.gartner.com/andrew-lerner/2014/07/16/the-cost-of-downtime/>. Luettu 13.9.2017.

10 Ferro, Greg. Scripting Does Not Scale For Network automation. EtherealMind. Verkkodokumentti. <http://etherealmind.com/scripting-scale-network-automation/>. Luettu 18.9.2017.

11 Kerner, Sean Michael. IDC: SDN is a Buzzword. Verkkodokumentti. Enterprise Networking Planet. <http://www.enterprisenetworkingplanet.com/datacenter/datacenter-blog/idc-sdn-is-a-buzzword.html>. Luettu 18.9.2017.

12 Edelman, Jason; S.Lowe, Scott & Oswalt, Matt. 2015. Network Programmability and Automation. O'Reilly Media Inc.

13 Culver, Timothy; Black, Chuch & Goransson, Paul. 2016. Software Defined Networks. 2nd Edition. Morgan Kaufmann.

14 Bombal, David. What Is SDN? Verkkovideo. Infinite Skills <https://www.safaribooksonline.com/library/view/software-defined-networking/9781491976609/video296132.html>. Katsottu 18.9.2017.

15 RFC-7426. Software-Defined Networking (SDN): Layers and Architecture Terminology. Verkkodokumentti. Internet Research Task Force (IRTF). <https://tools.ietf.org/html/rfc7426#section-3.7>. Luettu 18.9.2017.

16 Doherty, Jim. 2016. SDN and NFV Simplified. Addison-Wesley Professional.

17 Sridhar, Rao. SDN Series Part Eight: Comparison Of Open Source SDN Controllers. Verkkodokumentti. The New Stack. <https://thenewstack.io/sdn-series-part-eight-comparison-of-open-source-sdn-controllers/>. Luettu 18.9.2017.

18 Rouse, Margaret. Definition: northbound interface / southbound interface. Verkkodokumentti. TechTarget. <http://whatis.techtarget.com/definition/northbound-interface-southbound-interface>. Luettu 18.9.2017.

19 Bloomberg, Jason. SD-WAN: Entry Point For Software-Defined Everything. Verkkodokumentti. Forbes. <https://www.forbes.com/sites/jasonbloomberg/2017/03/20/sd-wan-entry-point-for-software-defined-everything/#3731e4b546ee>. Luettu 23.9.2017.

20 Butler, Brandon. SD-WAN: What is it and why you'll use it one day. Verkkodokumentti. Network World. <https://www.networkworld.com/article/3031279/sd-wan/sd-wan-what-it-is-and-why-you-ll-use-it-one-day.html>. Luettu 23.9.2017.

21 Hassan, Syed; Chayapathi, Rajendra & Shah, Paresh. 2016. Network Functions Virtualization (NFV) with a Touch of SDN. Addison-Wesley Professional.

22 Deploying SDWAN Technologies. Verkkoaineisto. Silver Peak. <https://training.silver-peak.com/#/courses/course/219810>. Luettu 5.10.2017.

23 REST API and JSON Schema. Verkkodokumentti. Hewlett Packard Enterprise. <https://h20566.www2.hpe.com/hpsc/doc/public/display?sp4ts.oid=1827663&docLocale=en_US&docId=emr_na-c05068466>. Luettu 23.9.2017.

24 Bombal, David. NETCONF. Verkkodokumentti. Infinite Skills. < https://www.safaribooksonline.com/library/view/software-defined-networking/9781491976609/video296179.html>. Katsottu 16.10.2017.

25 Bombal, David. YANG. Verkkodokumentti. Infinite Skills. <https://www.safaribooksonline.com/library/view/software-defined-networking/9781491976609/video296180.html>. Katsottu 16.10.2017.

26 Oswalt, Matt. SDN Protocols Part 5 – NETCONF. Verkkodokumentti. Keeping It Classless. <https://keepingitclassless.net/2015/03/sdn-protocols-part-5-netconf/>. Luettu 16.10.2017.

27 RFC-8040. RESTCONF Protocol. Verkkodokumentti. Internet Engineering Task Force. <https://tools.ietf.org/html/rfc8040>. Luettu 16.10.2017.

28 Ansible Installation. Verkkodokumentti. Red Hat. <http://docs.ansible.com/ansible/intro_installation.html>. Luettu 1.7.2017.

29 Github, Ansible Contributors. 2017. Verkkodokumentti. Github. https://github.com/ansible/ansible/graphs/contributors

30 Ansible documentation, Dynamic inventory. Verkkodokumentti. Red Hat. <http://docs.ansible.com/ansible/latest/intro_dynamic_inventory.html>. luettu 24.8.2017.

31 Ansible documentation, Developing Dynamic Inventory Sources. Verkkodokumentti. Red Hat. <http://docs.ansible.com/ansible/latest/dev_guide/developing_inventory.html>. Luettu 24.9.2017.

32 Ansible documentation, ios_interface. Verkkodokumentti. Red Hat. <http://docs.ansible.com/ansible/latest/ios_interface_module.html>. Luettu 25.9.2017.

33 Ansible documentation, YAML syntax. Verkkodokumentti. Red Hat. <http://docs.ansible.com/ansible/latest/YAMLSyntax.html>. Luettu 2.10.2017.

34 YAML Specification. Verkkodokumentti. Ben-Kiki, Oren; Evans, Clark & Ingy döt Net. <http://yaml.org/spec/1.2/spec.html>. Luettu 3.10.2017.

35 Hochstein, Lorin & Moser, Rene. 2017. Ansible Up and Running. 2nd Edition. O'Reilly Media.

36 McAllister, Jonathan. 2017. Implementing DevOps with Ansible 2. Packt Publishing.

37 Bombal, David & Duponchelle, Julien. GNS3 documentation, Getting Started with GNS3. Verkkodokumentti. <http://docs.gns3.com/1PvtRW5eAb8RJZ11maEYD9_aLY8kkdhgaMB0wPCz8a38/index.html>. Luettu 6.9.2017.

38 VMware Workstation documentation center, Configuring Host-Only Networking. Verkkodokumentti. VMware. <https://pubs.vmware.com/workstation-9/index.jsp#com.vmware.ws.using.doc/GUID-93BDF7F1-D2E4-42CE-80EA-4E305337D2FC.html>. Luettu 7.9.2017.

39 Ansible documentation, Variables. Verkkodokumentti. Red Hat. <http://docs.ansible.com/ansible/latest/playbooks_variables.html>. Luettu 17.10.2017.

40 Geerling, Jeff. 2017. Ansible open sources Ansible Tower with Aix. Verkkodokumentti. Jeff Geerling. <https://www.jeffgeerling.com/blog/2017/ansible-open-sources-ansible-tower-awx>. Luettu 17.10.2017.