Tampereen yliopisto

Ville Nupponen

HAVAITTUJA VIRHEITÄ ANSIBLEN KÄYTÖSSÄ

Ansiblen perusteet ja mahdollisia virheitä ratkaisuineen

Kandidaatintyö
Informaatioteknologian ja viestinnän tiedekunta
Tarkastajat: Petri Kannisto
Toukokuu 2022

TIIVISTELMÄ

Ville Nupponen: Havaittuja virheitä Ansiblen käytössä
Kandidaatintyö
Tampereen yliopisto
Tietotekniikka
Toukokuu 2022

Tässä työssä perehdytään eri tutkimuksissa ilmenneisiin virheisiin, joita esiintyy, kun käytetään Ansiblea keinona toteuttaa infrastruktuuri. Ilmenneistä virheistä esitellään muutama eniten esille tulleista, jotta saadaan käsitys minkälaisia virheet voivat olla. Virheisiin esitetään myös ratkaisuja, joita käyttämällä vältyttäisiin kyseisiltä virheiltä.

Tarkastellut virheet voidaan jakaa kahteen eri kategoriaan; tietoturvahajuihin ja huonoihin käytäntöihin. Tietoturvahajut ovat mahdollisia virheitä, jotka mahdollisesti heikentävät toteutuksen tietoturvaa. Huonot käytännöt puolestaan on yleisesti määritelty huonoksi tavaksi toteuttaa kyseinen asia. Näistä selvästi kriittisempiä ovatkin juuri tietoturvahajut, jotka voivat aiheuttaa merkittäviä ongelmia, kun huonot käytänteet puolestaan ovat enemmän kehittämisen ja ylläpidettävyyden kannalta häiritseviä.

Tietoturvahajuissa selvästi eniten esille on tullut salaisuuksien kovakoodaaminen. Dokumentaatiossa lähes poikkeuksetta näytetään esimerkit kovakoodatuilla salaisuuksilla, jotta dokumentaatio saadaan pidettyä mahdollisimman yksinkertaisena ja selvänä. Toinen useasti esille noussut tietoturvahaju oli eheystarkistuksen puuttuminen.

Huonoissa käytänteissä esille nousivat riittämätön modulaarisuus ja idempotenssin rikkominen. Riittämättömän modulaarisuuden ongelma tulee vastaan sovelluksen laajentuessa, eikä sitä ole aluksi välttämättä helppo havaita. Projektin lähtiessä huonosti liikkeelle sitä voi olla myöhemmin hankala lähteä korjaamaan. Idempotenssin rikkomisella voidaan aiheuttaa ympäristöihin yllättäviä ongelmia, joita ei välttämättä pystytä toistamaan ja sen myötä korjaaminen voi olla haastavaa.

Esitellyt virheet ovat vain pieni osa pienestä joukosta tutkimuksia eikä kaikkien löydettyjen ongelmien käsittely ole mitenkään mielekästä. Ei ole myöskään oletettavaa, että kukaan pystyisi tiedostamaan ja hallitsemaan kaikkia mahdollisia ongelmia. Automaattiset työkalut virheiden tunnistamiseen eivät ole vielä kovin kattavia, mutta joitakin ongelmia on mahdollista tunnistaa. Yhtenä vaihtoehtona työn loppupuolella on esitelty Ansible Lint -ohjelma, joka kykenee huomauttamaan joistakin huonoista käytänteistä.

Avainsanat: IaC, Infrastructure as Code, Ansible, Tietoturvahajut, Huonot käytännöt, Ansible-lint

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

CI/CD	Continuous Integration and Continuous Delivery/Deployment
IaC	Infrastructure as Code

1. JOHDANTO

Teknologian kehityksen myötä palvelinympäristöjen vaatimukset ovat muuttuneet merkittävästi. Nykyisin suosittuja palveluita on aikoinaan voitu ylläpitää henkilökohtaisilla koneilla. Kuitenkaan nykyään käyttäjämäärien kasvun, ylläpidollisten vaatimusten ja monien muiden syiden vuoksi ei vain jotenkuten toimimaan saatu ratkaisu ole kelvollinen. Eri vaihtoehtoja on nykyään lukemattomia. Tässä työssä on tarkoituksena tutustua yhteen näistä vaihtoehdoista ja tarkastella joitakin käyttäjien aiheuttamia yleisimpiä ongelmia sen käytössä, joita muissa tutkimuksissa on havaittu.

Yhtä palvelinympäristöjen hallintaan tarkoitettua keinoa kutsutaan nimellä "infrastruktuuri koodina"(IaC, Infrastructure as Code). IaC:lla nimensä mukaisesti infrastukruuri rakennetaan koodin avulla. Jokainen yksittäinen toiminta on määritelty koodiin ja tämän jälkeen koodi suoritetaan, jotta halutut muutokset saadaan tehtyä. Yksi tällaista keinoa käyttävistä ratkaisusta on Ansible. Vaihtoehtoisia ratkaisuja ovat muun muassa Puppet, Salt ja Chef. Tässä työssä kuitenkin keskitytään nimenomaan Ansiblella toteutettuun ratkaisuun.

Ensimmäisessä luvussa käydään läpi Ansiblen perusteet, jotka ovat merkittäviä Ansiblen käyttämisen kannalta. Luvussa käydään läpi perusteita, kuten muuttujia ja käsittelijöitä, sekä käydään läpi, miten Ansible rakentuu aina pienistä yksittäisistä tehtävistä kokonaisiksi playbookeiksi. Työn taustalta löytyy useampi ympäristö, joka on toteutettu Ansiblen avulla, joten työssä on yksittäisiä esimerkkejä näistä ympäristöistä. Työssä ei kuitenkaan työn luonteen vuoksi käsitellä konkreettista toteutusta vaan esitellään vain yleisellä tasolla, miten Ansible toimii.

Seuraavassa luvussa tuodaan esille joitakin tutkijoiden havaitsemia virheitä, joita käyttäjät voivat mahdollisesti tehdä Ansiblea käyttäessä. Esitellyt virheet ovat tulleet esille useita kertoja tutkimuksissa. Virheiden esittelyn lisäksi näytetään joitakin mahdollisia ratkaisuja, joilla voidaan välttää kyseisiä virheitä. Viimeisessä luvussa käydään läpi yksi mahdollinen ratkaisu virheiden automaattiseen tunnistamiseen.

2. ANSIBLEN PERUSTEET

Ansible on yksi monista eri vaihtoehdoista toteuttaa infrastruktuuri koodina. Tässä luvussa esitellään lyhyesti, mitä infrastruktuuri koodina on ja mitä eri vaihtoehtoja on. Tämän lisäksi käydään läpi Ansiblen perusteita, joiden tietäminen on välttämätöntä myöhempien lukujen kannalta.

2.1 Infrastruktuuri koodina

Infratruktuuri koodina on nimensä mukaisesti sitä, että infrastruktuuri muodostetaan koodin pohjalta. Lähtökohtaisesti ympäristö on tällöin luotu automaattisesti ja määritellyllä tavalla, ja se on mahdollista toteuttaa uudelleen samanlaisena. Infrastruktuurin toteuttaminen koodin avulla mahdollistaa koko infrastruktuurin dokumennoin toteutuksen ohessa eikä jätä ympäristöön liittyviä asioita tekijän muistin varaan. (Morris, 2020)

Tunnettuja IaC-ratkaisuja ovat muun muassa Ansible, Puppet, Chef, SaltStack ja Terraform. Idea eri ratkaisuissa on pääpiirteittäin sama. Suurimpia eroja tuovat kieli, jolla ratkaisu on toteutettu sekä ajamisessa käytettävä metodi. Osa ratkaisuista, kuten Chef ja Puppet, suosivat veto-metodia, kun puolestaan Ansible ja Terraform työntö-metodia. Veto-metodissa käyttäjä suorittaa ohjelman palvelimella, jolle haluaa tehdä muutoksia. Tällöin ohjelma hakee tarvittavat määritykset keskitetyltä hallintapalvelimelta. Työntö-metodissa puolestaan ohjelma ottaa yhteyden halutuille palvelimille, toteuttaakseen käyttäjän määrittämät toiminnot. (Riti & Flynn, 2021)

2.2 Ansiblen asennus ja käyttö

Versiosta 2.10 alkaen Ansible on jakautunut kahteen eri pakettiin: ansibleen ja ansible-coreen. Varsinainen Ansiblen ydin löytyy ansible-core-paketista. Ansible-paketti puolestaan sisältää yhteisön tarjoaman valmiin kattauksen moduuleja, joita voidaan käyttää eri tehtävissä. (”Ansible Documentation”, 2022)

Ansible tarvitsee toimiakseen Pythonin, joten käyttöjärjestelmästä löytyy todennäköisesti myös Pythonin pakettimanageri eli Pip, jolla Ansiblen asennus onnistuu vaivattomasti:

```
$ pip3 install ansible
```

Tämän jälkeen voidaan toiminnallisuus testata esimerkiksi:

```
$ ansible localhost -m ping
```

2.2.1 Inventaario

Ansiblella hallittavat palvelimet luetteloidaan inventaariossa. Yksinkertaisimmillaan inventaario sisältää listan palvelimien nimistä esimerkin 2.2.1 mukaisesti. (Freeman ym., 2020)

```
mail.example.com

[webservers]
foo.example.com
bar.example.com

[dbservers]
one.example.com
two.example.com
three.example.com
```

Inventaariot voivat myös olla dynaamisia (Freeman ym., 2020). Tällöin lista palvelimista voidaan hakea esimerkiksi palveluntarjoajan ohjelmointirajapinnasta. (”Ansible Documentation”, 2022).

2.2.2 Playbookit ja tehtävät

Ansiblessa suoritetaan yksittäisiä pieniä tehtäviä, joita peräkkäin ajamalla saadaan toteutettua kokonaisuuksia. Tehtävät käyttävät jotain asennettua moduulia. (Freeman ym., 2020) Esimerkissä 2.1 käytetään apt-moduulia, jolla saadaan asennettua määritelty paketti. Esimerkin mukaisesti tehtäville annetaan kuvaava nimi, joka tulee näkymään myöhemmin ulostulossa, kun tehtäviä ajetaan.

Listing 2.1. *Figlet-ohjelman asentaminen Ansiblen apt-moduulilla*

```
- name: Install figlet
  apt:
    name: figlet
```

Playbookit puolestaan määrittelevät rooleista ja tehtävistä muodostuvia kokonaisuuksia. (Freeman ym., 2020)

Listing 2.2. *Esimerkki playbookista*

```
- hosts: hcloud
  pre_tasks:
```

```
  - name: Install figlet
    apt:
      name: figlet
roles:
  - role: timezone
    tags: timezone
  - role: motd
    tags: motd
```

Esimerkissä 2.2 on määritelty hcloud-ryhmälle, joka saadaan haettua inventaariosta, ajettavaksi ensimmäisenä yksittäinen tehtävä ja tämän jälkeen kaksi roolia.

Playbook saadaan ajettua ansible-playbook -komennolla:

```
$ ansible-playbook -i inventory.ini playbook.yml
```

2.2.3 Roolit, sapluunat ja käsittelijät

Yksittäisten tehtävien kirjoittamisen sijaan on mahdollista tehdä rooleja, jotka sisältävät useita tehtäviä ja muita niiden tarvitsemia asioita, kuten sapluunoita (template) tai käsittelöitä (handlers). Samaa roolia voidaan käyttää myös useissa eri playbookeissa, jolloin rooleilla saadaan muodostettua pieniä kokonaisuuksia, joita yhdistelemällä saadaan muodostettua koko kokonaisuus. (Freeman ym., 2020) Roolin kansiorakenne on esimerkin 2.2.3 mukainen (”Ansible Documentation”, 2022).

```
roles/
    example-role/
        tasks/
        handlers/
        library/
        files/
        templates/
        vars/
        defaults/
        meta/
```

Yksinkertaisimmillaan rooli sisältää vain tasks-kansion, jossa on määritelty main.yml-tiedosto, joka sisältää kaikki roolin ajettavat tehtävät. Rooli voi siis käytännössä olla vain yksi tehtävä, kuten aiemmin esitelty esimerkki yksittäisestä tehtävästä 2.1.

Roolit voivat sisältää myös muita hyödyllisiä työkaluja, kuten sapluunoita tai käsittelijöitä. Sapluunoiden avulla voidaan rakentaa esimerkiksi ohjelman asetustiedostolle pohja, johon täytetään tiedot palvelinkohtaisesti. Sapluuna muodostetaan jinja2-kieltä käyttämällä. (Freeman ym., 2020)

Listing 2.3. *Esimerkki käyttäjän ssh-avaimien listaamisesta*

```
{% for key in user.ssh_keys %}
{{ key }}
{% endfor %}
```

Jinja2-kieli tukee datan yksinkertaista käsittelyä, kuten esimerkin 2.3 mukaista listojen läpikäyntiä. Sapluunoja käytetään template-moduulin avulla, josta on esimerkki käsittelijäesimerkin yhteydessä 2.5. (”Ansible Documentation”, 2022)

Ohjelmien asennus voi vaatia useita pieniä tehtäviä, jotta ohjelma toimii oikein. Tällöin ohjelman käynnistämistä ei voida välttämättä toteuttaa ennen kuin kaikki tehtävät on suoritettu. Tällöin on mahdollista merkitä tehtäville käsittelijä. Käsittelijä ajetaan vasta, kun kaikki tehtävät on suoritettu. Yleinen käyttö käsittelijöille on nimenomaan asetustiedostojen yhteydessä. Käsittelijä ajetaan vain jos tehtävä, johon se on liitetty, tekee muutoksia. Tällöin voidaan esimerkiksi käynnistää ohjelma uusiksi mikäli asetustiedostossa on muutoksia ja ohjelma tarvitsee uudelleenkäynnistyksen ladatakseen uudet muutokset. (Freeman ym., 2020)

Käsittelijä laitetaan handlers-kansioon main.yml-tiedostoon esimerkin 2.4 mukaisesti.

Listing 2.4. *Esimerkki käsittelijän toteuttamisesta*

```
- name: restart traefik
  command: docker restart traefik
```

Määriteltyjä käsittelijöitä voidaan käyttää tehtävän yhteydessä notify-avainsanan avulla, kuten esimerkissä 2.5 on näytetty.

Listing 2.5. *Esimerkki käsittelijän toteuttamisesta*

```
- name: Config traefik
  template:
    src: traefik.toml.j2
    dest: /opt/traefik/etc/traefik.toml
    owner: root
    group: root
    mode: 0644
  notify: restart traefik
```

2.2.4 Muuttujat ja holvi

Ansiblessa tietoa säilytetään muuttujissa. Muuttujia on mahdollista määrittää monella eri tavalla. Muuttujat on voitu määritellä esimerkiksi roolissa, inventaariossa tai playbookissa ja ne voivat kohdistua tiettyyn palvelimeen tai palvelimien muodostomaan ryhmään. (Freeman ym., 2020)

Muuttujiin saatetaan kuitenkin haluta määritellä salasanoja tai muita arkaluontoisia tietoja. Jotta arkaluontoiset tiedot saadaan salattua, voidaan käyttää Ansiblen tarjoamaa holvia. (Freeman ym., 2020)

```
$ pwgen -s 77 1 | tr -d '\n' | ansible-vault encrypt_string \
    --vault-password-file ansible-vault.secret \
    --encrypt-vault-id default
Reading plaintext input from stdin. (ctrl-d to end input, twice if
your content does not already have a newline)

!vault |
          $ANSIBLE_VAULT;1.1;AES256
          353864636465653635646131393132383135663864383139613562383636330336
          356330336364356161303237663964653430336663353333353962653539383935
          3938393536360a633761363030373030376139363165326565613664623335363136
          333536313631353361633164613263343334633030386536333263613534643631
          346436313331633664396532620a323439626639376666333634333536
          346131366462303661313036393161623130363138313039626138663 4
          313439343932653838666563343630656266333036653566636563323 2
          303964636133363633333032303531333439613436323263646133303 3
          356637366431336135613561633036383865383261353639663938333 9
          616436646334373662363563303539396230393337366266623763663 3
Encryption successful
```

Esimerkissä 2.2.4 generoidaan satunnainen 77-merkkinen salasana, joka sen jälkeen välitetään ansible-vault -komennolle, joka salaa generoidun salasanan ansible-vault.secret -tiedoston avulla. Tämän jälkeen salattua tietoa voidaan käyttää muuttujien sisältönä esimerkin 2.6 mukaisesti.

Listing 2.6. *Muuttujien määrittely*

```
username: root
password: !vault |
    $ANSIBLE_VAULT;1.1;AES256
    353864636465653635646131393132383135663864383139613562383635633033633
    643561613032376639646534303366633533333539626535393839353636 0a633761
    363030373030376139363165326565613664623335363136313533613633164613263
    343334633030386536333263613534643631333163366439653262 0a323439626639
    376666333634333536346131366462303661313036393161623130363138313039623
    613866343134393439326538386665633436306562663330366535666365633232303
    964636133363633333032303531333439613436323263646133303335663736643 1
    336135613561633036383865383261353639663938333961643664633437366236353
    633035393962303933373662666237636633
```

Kunhan vain Ansible pystyy saamaan tiedoston, jonka avulla salaus toteutettiin, pystyy Ansible purkamaan salaisuuden ajon yhteydessä. Nyt on vain huolehdittava salaamisessa käytettävän tiedoston välittämisestä kaikille sitä tarvitseville ja kaikki muut salaisuudet voivat olla versionhallinnassa.

3. HAVAITTUJA VIRHEITÄ

Tässä luvussa käsitellään joitakin havaittuja virheitä, joita käyttäjä on mahdollisesti voinut tehdä käyttäessään Ansiblea. Luvussa esitetyt virheet ovat esimerkkejä mahdollisista virheistä, joita on tutkimuksissa (Kumara ym., 2021; Rahman ym., 2019; Rahman ym., 2021) havaittu useita kertoja.

3.1 Tietoturvahajut

Tietoturvahajulla tarkoitetaan toistuvia koodausmalleja, jotka heikentävät turvallisuutta (Rahman ym., 2021). IaC:n modulaarisen luonteen vuoksi huono ratkaisu voi toistua varsinaisessa ympäristöissä useita kertoja, vaikka olisikin määritelty vain kerran.

3.1.1 Kovakoodatut salaisuudet

Selvästi yleisin havaituista tietoturvahajuista on kovakoodatut salaisuudet. Kovakoodatuissa salaisuuksissa paljastetaan yleensä selvätekstisenä joitakin arkaluontoisia tietoja, kuten käyttäjätunnuksia ja salasanoja. (Rahman ym., 2019)

Kovakoodatut salaisuudet ovat yksinkertaisimmillaan sitä, että tehtävän vaatima salaisuus kirjoitetaan suoraan tehtävän yhteyteen ja on kenen tahansa luettavissa, joka pääsee käsiksi koodiin. Ongelma on havainnollistettu esimerkissä 3.1.

Listing 3.1. *Esimerkki kovakoodatusta salasanasta*

```
- name: Log into registry
  docker_login:
    username: admin
    password: not_so_secret_password
```

Salasanoja kovakoodataan herkästi siksi, että se on yksinkertaisin tapa. Vaikka salasana tulisikin muuttujan kautta, ei ratkaisu suoraan ole parempi, mikäli muuttujan sisältö on yhä selvätekstisenä luettavissa. Onkin suositeltavaa käyttää holveja apuna salaamaan salaisuuksia (Rahman ym., 2019). Ansible Vaultin avulla pystätään salaamaan muuttujia ja tiedostoja. Kyseinen salaus perustuu siihen, että salauksen purkamiseen tarvitaan erillinen avain, joka jaetaan erikseen sen tarvitseville. Tällöin voidaan varsinainen salaisuus

pitää koodin mukana. Esimerkki Ansible Vaultin käytöstä on esitetty esimerkissä 3.2. Kuten esimerkistä nähdään, ei salaisuutta pysty suorilta käsiltä näkemään. Salaisuus ei ole myöskään selvitettävissä ohjelmallisesti ilman salauksen purkamiseen tarvittavaa avainta.

Listing 3.2. *Esimerkki Ansible Vaultilla suojatusta salaisuudesta*

```
- name: Log into registry
  docker_login:
    username: admin
    password: !vault |
      $ANSIBLE_VAULT;1.2;AES256;dev
      30613233633461343837653833666333643061636561303338373661313838333
      56565363535316232633634346237333435386534626130643336343334646 60a
      66363362393939343931663663386361636237636537333938306331383339353
      26536323964393966663938653062633063333763383366646563343731666 30a
      36373639326266646566343261393261303630396334326362313738623963 30
```

Mahdollisen syyn kovakoodatuille salaisuuksille voivat muodostaa salaisuuksien salaamisen tuoma ylimääräinen vaihe sekä dokumentaatiossa olevien esimerkkien yksinkertaisuus. Aikasempi esimerkki 3.1 vastaa dokumentaatiosta löytyvää esimerkkiä (”Ansible Documentation”, 2022). Dokumentaatio pyrkii toki olemaan tehtävien toiminnallisuuden kannalta yksinkertainen, joten on ymmärrettävää, ettei kyseisen tehtävän esimerkeissä ole lähdetty esittelemään tehtävään suoraan liittymättömän toiminnallisuuden käyttöä.

3.1.2 Eheystarkistuksen puuttuminen

Tiedostojen eheyden tarkistuksessa voidaan käyttää tarkistussummia. Eheyden tarkistamisella pyritään varmistamaan, että tiedostoa ei ole muutettu sen luonnin jälkeen. Tarkistussumma on yleensä 128–512 bittiä pitkä kiinteän pituinen merkkijono riippumatta tiedoston sisällön pituudesta. Saman tarkistussumman luominen muutetulla tiedoston sisällöllä (esimerkiksi mukaan ujutetulla haittaojelmalla) vaatisi raa’an voiman hyökkäyksen, minkä seurauksena se on käytännössä mahdotonta. (Meylan ym., 2020)

Mikäli eheyttä ei tarkisteta, tehdään oletus, että tiedosto on aina turvallinen eikä mahdollisesti korruptoitu hyökkääjän toimesta (Rahman ym., 2021). Esimerkki tällaisesta oletuksesta on mahdollista toteuttaa esimerkin 3.3 mukaisesti. Esimerkistä ei huomaa tarkistussumman puuttumista, mikäli ei ole perehtynyt tarkemmin dokumentaatioon, jossa se mainitaan.

Listing 3.3. *Esimerkki tiedoston lataamisesta ilman tarkistussummaa*

```
- name: Download file
  get_url:
```

```
    url: https://example.com/path/file.txt
    dest: /tmp/file.txt
```

Tarkistussumman lisääminen on tehty monesti hyvinkin yksinkertaisesti. Aiempaan esimerkkiin 3.3 saadaan lisättyä tarkistussumma lisäämällä yksittäinen rivi, kuten esimerkissä 3.4 näytetään.

Listing 3.4. *Esimerkki tiedoston lataamisesta tarkistussummalla*

```
- name: Download file
  get_url:
    url: https://example.com/path/file.txt
    dest: /tmp/file.txt
    checksum: |
      sha256:9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0
      f00a08
```

Mikäli tarkistussummaa ei tarjota valmiiksi, voi sen esimerkiksi generoida siihen tehdyllä komennolla (Ulrich Drepper, 2022).

```
$ sha256sum example.txt
```

Mikäli tarkistussumma ladataan täysin samasta paikasta kuin tiedostokin, on mahdollista, että jos tiedosto saadaan korruptoitua, voidaan myös tarkistussumma korruptoida. Tämän vuoksi voisi olla suositeltavaa määritellä yksittäisen tiedoston latauksessa tarkistussumma suoraan kuten esimerkissä 3.4 ilman että tarkistussummakin haetaan muualta.

3.2 Huonoja käytäntöjä

Asioiden tekemiseen esitellään usein hyviä ja huonoja käytöntöjä. Virheitä tarkastellessa haluamme kiinnittää huomiota erityisesti huonoihin käytäntöihin, joita voi mahdollisesti esiintyä tehdyissä ratkaisussa. Esimerkkejä huonoista käytännöistä Ansiblella on muun muuassa riittämätön modulaarisuus ja idempotenssin rikkominen.

3.2.1 Riittämätön modulaarisuus

Ansiblessa on mahdollista määritellä suoraan yksittäisiä tehtäviä playbookeissa ajettavaksi (”Ansible Documentation”, 2022). On kuitenkin suositeltavaa, että tehtäviä ei käytetä ilman rooleja, jotta koodista saadaan modulaarisempaa. Yleisesti ottaen ei ole hyvä, että yksittäiset muutokset edellyttäisivät koko playbookin ajamista. (Kumara ym., 2021).

Ansiblessa on mahdollista määrittää rooleille ja tehtäville tunnisteita. Tunnisteiden avulla voidaan rajata ajamaan vain merkityt roolit tai tehtävät. (”Ansible Documentation”, 2022)

Esimerkissä 3.5 asennetaan ohjelma pyörimään Dockerissa. Asennuksen yhteydessä määritellään Dockerin datasäiliöt (volume) datan säilyttämistä varten. Kyseinen ohjelma saadaan päivitettyä lataamalla uusi versio Docker imagesta ja käynnistämällä ohjelma uudelleen. Ensimmäinen ja neljäs tehtävä tekevätkin tämän, mikäli annetussa imagessa muutetaan versionumeroa.

Listing 3.5. *Lhcin asentamiseen tarvittavat tehtävät*

```
- name: Pull Docker image
  docker_image:
    name: "{{ lhci_docker_image }}"
    source: pull
  tags: lhci_upgrade

- name: Volume directories
  file:
    path: "{{ item }}"
    state: directory
  with_items:
    - "{{ lhci_path }}"
    - "{{ lhci_data_path }}"

- name: Docker volumes
  docker_volume:
    name: lhci-data
    driver: local
    driver_options:
      type: none
      device: "{{ lhci_data_path }}"
      o: bind

- name: Docker container
  docker_container:
    name: lhci
    image: "{{ lhci_docker_image }}"
    state: started
    restart_policy: unless-stopped

    volumes:
      - lhci-data:/data

    init: true
    tmpfs:
```

```
      - /tmp
    labels:
      traefik.enable: "true"
      traefik.http.services.lhci.loadbalancer.server.port: "9001"
  tags: lhci_upgrade
```

Ei kuitenkaan ole tarvetta määritellä datasäiliöitä uudelleen joka kerta, joten riittää vain ensimmäisen ja neljännen tehtävän ajaminen. Tämän saamme tehtyä määrittämällä esimerkissä näkyvät ‘tags‘-attribuutit ja rajaamalla ajaessa:

```
$ ansible-playbook ... -t lhci_upgrade
```

3.2.2 Idempotenssin rikkominen

IaC:lla hallittavien ympäristöjen halutaan olevan toistettavia eikä hallitsemattomia. Ansiblella on mahdollista ajaa mielivaltaisesti komentoa esimerkiksi ‘command‘- ja ‘shell‘-moduulien avulla. Näitä käyttäessä onkin syytä olla erittäin huolellinen, jotta ympäristön toistettavuus saadaan pidettyä. (Kumara ym., 2021)

Ansiblessa on mahdollista toteuttaa ‘command‘-moduulilla mahdollisesti ajettavia asioita muilla moduuleilla, jolloin vältytään mahdollisesti idempotenssin rikkomiselta. Esimerkissä 3.6 on yksinkertainen esimerkki ‘rm‘-komennon korvaamisesta ‘file‘-moduulilla.

Listing 3.6. *Lhcin asentamiseen tarvittavat tehtävät*

```
- name: Don't do this
  ansible.builtin.command: rm /tmp/example.txt

- name: Do this
  ansible.builtin.file:
    path: /tmp/example.txt
    state: absent
```

4. VIRHEIDEN AUTOMAATTINEN TUNNISTAMINEN

Tutkimuksissa (Rahman ym., 2019) ja (Rahman ym., 2021) käytettiin tutkimuksia varten tehtyjä työkaluja, jotka oli empiirisesti validoitu toimiviksi. Kyseiset työkalut eivät yleisesti ole tiedossa tai käytettävissä. Tässä luvussa on kuitenkin esitelty yksi työkalu, Ansible Lint, joka edesauttaa vastaavien virheiden havaitsemista ja johon on mahdollista rakentaa itse lisätarkistuksia (Sesto, 2020).

4.1 Ansible Lint

Ansible Lint on työkalu, joka on suunniteltu tarkistamaan noudatetaanko parhaita käytäntöjä ja sen myötä edesauttamaan käytäntöjen parantamista (”ansible-community/ansible-lint: Best practices checker for Ansible”, 2022). Työkalua ei ole nimenomaan suunniteltu huomaamaan turvallisuuspuutteita, mutta parhaat käytännöt sisältävät myös asioita, jotka ovat turvallisuuden kannalta oleellisia.

4.1.1 Manuaalinen ajaminen

Ansible vaatii ajamiseen tarkoitetulla palvelimella Python 3.8:n (”Ansible Documentation”, 2022). Tämän myötä koneelle on todennäköisesti asennettu myös pip. Työkalun asentaminen onnistuukin tällöin helposti pipin avulla (”Ansible Lint Documentation”, 2022):

```
$ pip3 install ansible-lint
```

Työkalu ei tarvitse lisämäärityksiä vaan voi tunnistaa automaattisesti tarvittavat tiedostot Ansible-projektista (”Ansible Lint Documentation”, 2022):

```
$ ansible-lint
```

Komennon antama ulostulo voi näyttää esimerkin 4.1.1 mukaiselta.

```
no-handler: Tasks that run when changed should likely be handlers
hcloud_roles/hcloud_server/tasks/main.yml:119 Task/Handler: Discord
    notification


risky-file-permissions: File permissions unset or incorrect
```

```
roles/address_checker/tasks/main.yml:13 Task/Handler: DB volume
    directory
```

Kattavampi ulostulo on esitelty liitteessä A. Liitteestä nähdään, kuinka kehittäjältä jää helposti useita yksinkertaisia ongelmia huomaamatta ja että samat ongelmat toistuvat useasti.

4.1.2 Automaattinen ajaminen Gitlab CI:n avulla

Gitlabia tietovarastona (repository) käyttävät ohjelmistoprojektit voivat käyttää jatkuvan integraation ja kehittämisen suorittamiseen Gitlab CI/CD:ta, joka mahdollistaa erilaisten tehtävien suorittamisen automaattisesti koodivarantoon tehtyjen muutoksien perusteella. Gitlab CI/CD perustuu avoimeen lähdekoodiin ja sen ilmainen versio on ollut jo aikoinaan parhaimpia ja kattavampia. (Karvonen, 2021)

Yleisesti tunnettuja vaihtoehtoisia ratkaisuja ovat muun muassa Github Actions, Travis CI, CircleCI ja Jenkins. Tässä työssä kuitenkin esitellään toteutus vain Gitlab CI/CD:llä, sillä muilla toteuttaminen tapahtuu vastaavasti.

Gitlab CI:ssa tarvittavat määritykset laitetaan '.gitlab-ci.yml'-tiedostoon ("Gitlab CI/CD | Gitlab", 2022). Esimerkissä 4.1 esitellään yksinkertainen vaihtoehto, jolla saadaan ajettua 'ansible-lint' automaattisesti jokaisen muutoksen myötä.

Listing 4.1. *Esimerkki yksinkertaisesta Gitlab CI määrityksestä Ansible Lint:n ajamiseen automaattisesti*

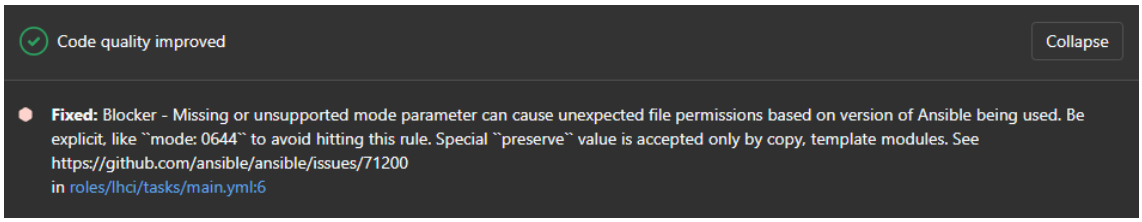
```
stages:
  - lint

lint:ansible:
  stage: lint
  image: python:3-slim
  before_script:
    - pip3 install -r requirements.txt
  script:
    - ansible-lint -f codeclimate > codeclimate.json
  allow_failure: true
  artifacts:
    reports:
      codequality: codeclimate.json
```

Ylläoleva esimerkki on yksinkertainen eikä sisällä esimerkiksi välimuistin määrittämistä, jottei työkaluja tarvitse jokaisella suorituskerralla asentaa uusiksi. Lisäksi työkalun ajaminen joka kerta ei välttämättä ole suositeltava ratkaisu mikäli muutokset eivät kohdistuneen

mihinkään, joka vaikuttaisi lopputulokseen.

4.1.3 Tulosten näyttäminen Gitlabissa

Gitlab CI vertailee koodinlaaturaportteja eri haarojen välillä ja näyttää tulokset Gitlabin käyttöliittymän kautta (”Gitlab CI/CD | Gitlab”, 2022). Kuvassa 4.1 on näytetty esimerkki siitä, kun koodin laatu paranee muutosten myötä.



Kuva 4.1. *Kuvankaappaus Gitlabin Merge Requestin yleiskatsauksesta*

On hyvä huomioida, että kohdehaarassa tulee olla muodostettu vastaava raportti vähintään kerran, jotta vertailua voidaan suorittaa. Muodostetut raportin on mahdollista ladata Gitlabin käyttöliittymän kautta.(”Gitlab CI/CD | Gitlab”, 2022)

5. YHTEENVETO

Tässä työssä perehdyttiin useissa tutkimuksissa havattuihin mahdollisiin virheisiin silloin kun toteutetaan infrastruktuuri koodina. Työssä tarkasteltiin nimenomaan Ansiblen näkökulmasta havaittuja virheitä ja esiteltiin ratkaisuja niihin. Merkittävimmät virheet tulivat ilmi tietoturvahajuja toteuttaneista tutkimuksista. Muut esitellyt virheet ovat lähtöisin Ansiblelle määritellyistä huonoista käytännöistä.

Tietoturvahajuissa useita kertoja esiintyneitä virheitä olivat kovakoodatut salaisuudet ja eheystarkistuksen puuttuminen. Huonoihin käytäntöihin liittyen esiintyneitä virheitä olivat puolestaan riittämätön modulaarisuus ja idempotenssin rikkominen. Kovakoodattuihin salaisuuksiin ratkaisu löytyy Ansiblen holvi-toiminnallisuudesta. Eheystarkistuksille puolestaan tuki löytyy suoraan moduuleihin sisäänrakennettuna ja vaatii vain sen käyttämistä. Huonot käytännöt puolestaan riippuvat tehdyistä ratkaisuista. Asioita voidaan tehdä usealla tavalla, joista tietyissä on ongelmia, joita tulisi välttää.

Tietoturvahajuja ja joitakin huonoja käytänteitä voidaan havaita automaattisilla työkaluilla. Tietoturvahajuihin keskittyneissä tutkimussa käytettiin tutkielmien yhteydessä tehtyjä työkaluja. Kyseiset työkalut eivät kuitenkaan olleet yksinkertaisesti saatavilla, joten tutkielmassa esiteltiin Will Thamesin luoma työkalu, Ansible Lint, joka on nykyisin Ansiblen yhteisötiimin jatkokehittämä työkalu ja yleisesti tunnettu hyvänä ratkaisuna Ansiblen kanssa käytettäväksi.

Vaikka yleisesti IaC:ta pidetään hyvänä ratkaisuna infrastruktuurien rakentamisessa, on hyvä huomioida, että keinot, joilla pystytään tekemään pienellä vaivalla paljon asioita, mahdollistavat myös pienillä virheillä suuren määrän virheitä kun yksittäinen tehtävä voidaan toteuttaa sadoille palvelimille. On hyvä tietää joistakin yleisimmistä virheistä ja mahdollisista keinoista havaita niitä. Tänä päivänä keinoja on monia erilaisten ratkaisujen kehittyessä ja eri ympäristöjen rakentaessa tukea yhä useammille työkaluille.

LÄHTEET

Ansible documentation. (2022, February 6). Retrieved February 6, 2022, from https://docs.ansible.com/

Ansible lint documentation. (2022, March 20). Retrieved March 20, 2022, from https://ansible-lint.readthedocs.io/en/latest/

Ansible-community/ansible-lint: Best practices checker for ansible. (2022, March 20). Retrieved March 20, 2022, from https://github.com/ansible-community/ansible-lint

Freeman, J., Locati, F. A., & Oh, D. (2020). *Practical Ansible 2*. Packt Publishing.

Gitlab CI/CD | Gitlab. (2022, March 20). Retrieved March 20, 2022, from https://docs.gitlab.com/ee/ci/

Karvonen, V. (2021). Python-kehitysympäristön automatisointi. *Informaatioteknologian ja viestinnän tiedekunta - Faculty of Information Technology Communication Sciences*.

Kumara, I., Garriga, M., Romeu, A. U., Di Nucci, D., Palomba, F., Tamburri, D. A., & van den Heuvel, W.-J. (2021). The do's and don'ts of infrastructure code: A systematic gray literature review. *Information and software technology*, *137*, 106593–.

Meylan, A., Cherubini, M., Chapuis, B., Humbert, M., Bilogrevic, I., & Huguenin, K. (2020). A Study on the Use of Checksums for Integrity Verification of Web Downloads. *ACM transactions on privacy and security*, *24*(1), 1–36.

Morris, K. (2020). *Infrastructure as Code, 2nd Edition*. O'Reilly Media, Inc.

Rahman, A., Parnin, C., & Williams, L. (2019). The Seven Sins: Security Smells in Infrastructure as Code Scripts. *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, *2019-*, 164–175.

Rahman, A., Rahman, M. R., Parnin, C., & Williams, L. (2021). Security Smells in Ansible and Chef Scripts: A Replication Study. *ACM transactions on software engineering and methodology*, *30*(1), 1–31.

Riti, P., & Flynn, D. (2021). Infrastructure as Code. Teoksessa *Beginning HCL Programming* (s. 65–78). Apress.

Sesto, V. (2020). Ansible Tests and Variables. Teoksessa *Practical Ansible* (s. 215–258). Apress.

Ulrich Drepper, D. M., Scott Miller. (2022, February 20). *Sha256sum(1) - linux man page*. Retrieved February 20, 2022, from https://linux.die.net/man/1/sha256sum

LIITE A: ANSIBLE-LINT KOMENNON ULOSTULO

```
WARNING  Listing 20 violation(s) that are fatal
no-handler: Tasks that run when changed should likely be handlers
hcloud_roles/hcloud_server/tasks/main.yml:114 Task/Handler: Gather
    facts

no-handler: Tasks that run when changed should likely be handlers
hcloud_roles/hcloud_server/tasks/main.yml:119 Task/Handler: Discord
    notification

risky-file-permissions: File permissions unset or incorrect
roles/address_checker/tasks/main.yml:13 Task/Handler: DB volume
    directory

risky-file-permissions: File permissions unset or incorrect
roles/<sensuroitu>/tasks/main.yml:14 Task/Handler: Database volume
    directory

risky-file-permissions: File permissions unset or incorrect
roles/<sensuroitu>/tasks/main.yml:13 Task/Handler: Database volume
    directory

risky-file-permissions: File permissions unset or incorrect
roles/<sensuroitu>/tasks/main.yml:15 Task/Handler: Database volume
    directory

risky-file-permissions: File permissions unset or incorrect
roles/config/tasks/main.yml:15 Task/Handler: DB volume directory

risky-file-permissions: File permissions unset or incorrect
roles/gitlab/tasks/main.yml:7 Task/Handler: Volume directories

risky-file-permissions: File permissions unset or incorrect
roles/keycloak/tasks/main.yml:13 Task/Handler: Create directories
```

```
risky-file-permissions: File permissions unset or incorrect
roles/lhci/tasks/main.yml:6 Task/Handler: Volume directories

unnamed-task: All tasks should be named
roles/nextcloud/tasks/groupfolders.yml:7 Task/Handler: set_fact
    groupfolders={{ groupfolder_info.stdout | from_json }}

unnamed-task: All tasks should be named
roles/nextcloud/tasks/groupfolders.yml:21 Task/Handler: set_fact
    groupfolders={{ groupfolder_info.stdout | from_json }}

no-changed-when: Commands should not change things if nothing needs
    doing
roles/nextcloud/tasks/groupfolders.yml:24 Task/Handler: Update group
    folder's group

no-changed-when: Commands should not change things if nothing needs
    doing
roles/nextcloud/tasks/groupfolders.yml:28 Task/Handler: Update group
    folder's quota

risky-file-permissions: File permissions unset or incorrect
roles/nextcloud/tasks/main.yml:39 Task/Handler: Volume directories

empty-string-compare: Don't compare to empty string
roles/nextcloud/tasks/main.yml:146 Task/Handler: Set up config owne

empty-string-compare: Don't compare to empty string
roles/nextcloud/tasks/main.yml:179 Task/Handler: Set up plugin config
    owner

risky-file-permissions: File permissions unset or incorrect
roles/<sensuroitu>/tasks/main.yml:17 Task/Handler: Database volume
    directory

command-instead-of-shell: Use shell only when shell functionality is
    required
roles/uisp/tasks/main.yml:9 Task/Handler: Run install script

no-changed-when: Commands should not change things if nothing needs
    doing
roles/uisp/tasks/main.yml:9 Task/Handler: Run install script
```

```
You can skip specific rules or tags by adding them to your
    configuration file:
# .ansible-lint
warn_list:  # or 'skip_list' to silence them completely
- command-instead-of-shell  # Use shell only when shell functionality
    is required
- empty-string-compare  # Don't compare to empty string
- experimental  # all rules tagged as experimental
- no-changed-when  # Commands should not change things if nothing needs
     doing
- no-handler  # Tasks that run when changed should likely be handlers
- unnamed-task  # All tasks should be named

Finished with 10 failure(s), 10 warning(s) on 253 files.
```