Київський національний університет імені Тараса Шевченка
Факультет радіофізики, електроніки та комп'ютерних систем
Кафедра комп'ютерної інженерії

ДОСЛІДЖЕННЯ ЗАСОБІВ ПОБУДОВИ СИСТЕМ АВТОМАТИЗОВАНОГО КОНФІГУРУВАННЯ ОБЛАДНАННЯ МЕРЕЖІ

Кваліфікаційна робота бакалавра
студента спеціальності
123 «Комп'ютерна інженерія»
Владислава ШЕЛІНА
_____(підпис)

Науковий керівник:
к.ф.-м.н., доцент
Юрій БОЙКО
_____(підпис)

Рецензент
д.ф.-м.н., професор
Євген ІВОХІН
_____(підпис)

До захисту допускаю
Протокол засідання кафедри від
"_"_____2022р. №

Завідувач кафедри
к.ф.-м.н., доцент
Юрій БОЙКО

КИЇВ - 2022

РЕФЕРАТ

Кваліфікаційна робота бакалавра: 60 сторінок, 21 рисунок, 1 таблицю, 10 джерел, 4 додатки.

В даній кваліфікаційній роботі бакалавра було розглянуто основні принципи автоматизації конфігурування мережевого обладнання, розглянуто засоби побудови систем автоматизації конфігурування мережевого устаткування, створено програмне забезпечення, що надає можливість збору фактів про поточні параметри та конфігурацію пристрою для проведення подальшого конфігурування наявного мережевого обладнання, а саме комутаторів Cisco (Cisco IOS), D'Link (серії DES та DGS), на яких побудована IT-інфраструктура університету.

КОНФІГУРАЦІЯ, АВТОМАТИЗАЦІЯ, МЕРЕЖЕВЕ ОБЛАДНЕННЯ, МЕРЕЖЕВА ІНФРАСТРУКТУРА, МЕРЕЖЕВИЙ КОМУТАТОР, АВТОМАТИЗОВАНЕ КОНФІГУРУВАННЯ

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

SSH (Secure SHell) – мережевий протокол рівня застосунів, що дозволяє проводити віддалене управління комп’ютерних систем і тунелювання TCP-з'єднань

Telnet (TELetype NETwork) – мережевий протокол для реалізації текстового інтерфейсу по мережі

ОС – операційна система

ПЗ – програмне забезпечення

CLI – інтерфейс командного рядка (англ. command-line interface, CLI)

VLAN – Virtual Local Area Network

ВСТУП

На сьогодні IT-компанії мають значну проблему, з якою стикаються кожен день. Питання полягає в тому, як за найкоротший термін мережевому адміністратору сконфігурувати десятки а то й сотні мережевих пристроїв. Наприклад, треба створити новий VLAN (Virtual Local Area Network) на одному з комутаторів в розгалуженій мережі, на цільовому комутаторі створюється VLAN, йому надається тег, конфігуруються порти доступу. А далі необхідно на кожному проміжному комутаторі в мережі налаштувати магістральні порти, щоб вони передавали трафік цього VLAN-у далі у мережу до необхідного нам місця (серверної, центру обробки данних, тощо). Для цього мережевому адміністратору треба вручну на кожному проміжному пристрої мережі (як приклад, на комутаторі) вводити необхідні налаштування, що не завжди робиться швидко, враховуючи що проміжних пристроїв може бути багато, і не дає гарантію відстуності помилок через людський фактор. Саме для таких випадків використовують автоматизовані системи управління конфігураціями. В даній роботі будуть розглянуті наявні на сьогодні системи управління конфігураціями та створено власну систему для конфігурування наявного мережевого обладнання на основі таких пакетів як Netmiko та TextFSM для обробки текстової відповіді.

У даній роботі буде розглянуто наявні системи автоматизації мережі та створення власної системи для роботи з наявним мережевим обладнанням, а саме комутаторами D’Link серій DGS і DES, та Cisco серії Catalyst.

МЕТА КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

Метою даної роботи є дослідження наявних засобів та технологій побудови систем автоматизованого конфігурування обладнання мережі, огляд та порівняння наявних на ринку систем автоматизації мережі, створення програмного забезпечення, що виконує автоматичне конфігурування наявного мережевого обладнання IT -інфраструктури університету з врахуванням поточних параметрів кожного окремого пристрою.

Розроблене програмне забезпечення зможе полегшити та пришвидшити процес конфігурування наявного мережевого обладнання IT-інфраструктури університету у випадку глобальних змін в конфігурації, коли необхідно провести певні налаштування на великій кількості пристроїв. Використання технологій для обробки текстової відповіді мережевого обладнання дозволяє отримувати поточні параметри його роботи для конфігурування мережевого обладнання.

Об'єкт дослідження – активне обладнання комп’ютерних мереж, програмне забезпечення автоматизації конфігурування наявного мережевого обладнання.

1 ТЕОРЕТИЧНА ЧАСТИНА

1.1 Означення мережевого устаткування

Мережеве устаткування — пристрої, необхідні для роботи комп'ютерної мережі, такими пристроями можуть бути наприклад: маршрутизатор, комутатор, концентратор, патч-панель та інші. Зазвичай виділяють активне і пасивне мережне устаткування. *Активне мережеве устаткування* — обладнання, яке використовує в роботі певну «інтелектуальну» особливість, аналізуючи та сортуючи дані, що проходять через нього. Тобто Маршрутизатор, комутатор (світч) тощо, є активним мережевим обладнанням. Навпаки — повторювач (репітер) і концентратор (хаб) не є активним мережевим устаткуванням, оскільки просто повторюють електричний сигнал для збільшення відстані з'єднання чи топологічного розгалуження і нічого «інтелектуального» собою не представляють.

Під *пасивним мережевим устаткуванням* розуміється обладнання, що не наділене “інтелектуальними” особливостями. Наприклад, вилка/розетка (RG58, RJ45, RJ11, GG45), кабель (коаксіальний і кручена пара UTP/STP), повторювач (repeater), патч-панель, концентратор (hub), та інші.

1.2 Склад устаткування комп’ютерної мережі

Фрагмент комп’ютерної мережі (Рис. 1.1) включає основні типи комунікаційного устаткування, застосовуваного сьогодні для побудови локальних мереж і їх з'єднання через глобальні мережі одне з одним. Для побудови локальних зв'язків між комп'ютерами використовуються різні види кабельних систем, мережеві адаптери, концентратори-повторювачі, мости, комутатори та маршрутизатори. Для підключення локальних мереж до глобальних використовуються спеціальні виходи (WAN порти) мостів і маршрутизаторів, а також апаратура передачі даних по довгих лініях – модеми (при роботі по аналоговимх лініямх) або пристрої підключення до цифрових каналів (ТА – термінальні адаптери мереж ISDN, пристрої обслуговування цифрових виділених каналів типу CSU/DSU тощо).

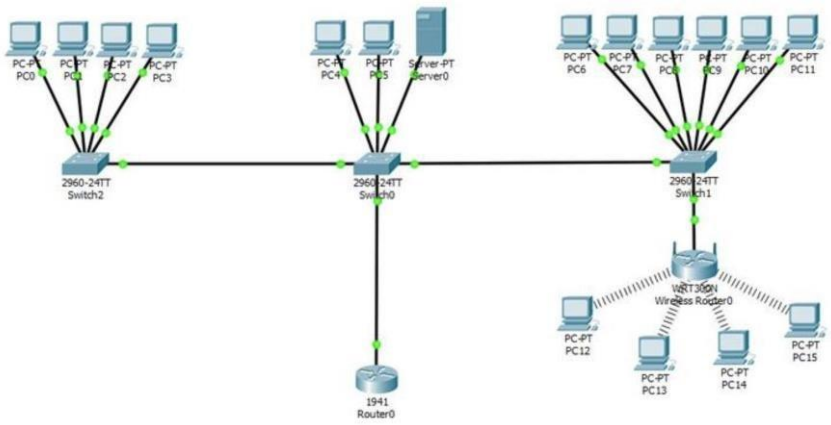


Рис. 1.1 – Топологія мережі з основними типами комунікаційного устаткування

Розглянемо більш детально основне устаткування, яке використовується для побудови комп'ютерних мереж.

Мережевий концентратор (hub).

Концентратор (рис. 1.2) служить центральною точкою, до якої підключаються всі хости в мережі. Концентратор — це пристрій рівня 1 OSI і не має поняття про кадри Ethernet або адресацію. Він просто отримує сигнал з одного порту і надсилає його на всі інші порти.

Рис. 1.2 – Зовнішній вигляд мережевого концентратора

Сьогодні концентратори вважаються застарілими, а замість них зазвичай використовуються комутатори.

Мережевий комутатор (switch):

Як і концентратори, комутатори використовується для підключення кількох хостів разом, але він має багато переваг перед концентратором. Комутатор (Рис.

1.3) є пристроєм рівня 2 OSI, що означає, що він може перевіряти отриманий трафік і приймати рішення про пересилання його певному користувачу.

Рис. 1.3 – Зовнішній вигляд мережевого комутатора

Комутатор зберігає в пам'яті таблицю, у якій вказуються відповідні MAC-адреси пристроїв, що підключені до певного порту комутатора. При ввімкненні комутатора ця таблиця порожня, і комутатор працює в режимі “навчання” (Address learning). У цьому режимі дані, що приходять на будь-який порт передаються на всі інші порти комутатора, при цьому комутатор аналізує кадри й, визначивши MAC-адресу хоста-відправника, заносить його в MAC таблицю у відповідність певному порту. Згодом, якщо на один з портів комутатора надійде кадр, призначений для цільового хоста MAC-адреса якого вже навна в таблиці, то цей кадр буде переданий тільки через порт, вказаний у таблиці. Якщо MAC-адреса хоста-отримувача ще не відома, то кадр буде продубльований на всі інтерфейси. Згодом комутатор вибудовує повну таблицю для всіх своїх портів, і в результаті трафік, призначений для цільового хоста направляється через порт, за яким він знаходиться, таким чином трафік локалізується. Кожен порт комутатора є окремим доменом колізій і може працювати в дуплексному режимі.

Мережевий маршрутизатор (router):

Маршрутизатор (Рис. 1.4) – це пристрій, який направляє пакети з однієї мережі в іншу. Маршрутизатор, як правило, є пристроєм рівня 3

OSI. Маршрутизатори розділяють широкомовні домени і мають можливості фільтрації трафіку.

Рис. 1.4 – Зовнішній вигляд мережевого маршрутизатора

Для того, щоб надіслати пакети в потрібному напрямку маршрутизатор використовує таблицю маршрутизації, яка зберігається пам'яті мершрутизатора. Таблиця маршрутизації може створюватись засобами статичної або динамічної маршрутизації. Крім того, маршрутизатори можуть здійснювати трансляцію адреси відправника й отримувача (використовуючи механізм NAT, Network Address Translation), фільтрацію транзитного потоку даних на основі певних правил з метою обмеження доступу, шифрування/дешифрування даних, що передаються. Маршрутизатори не можуть здійснювати передачу широкомовних повідомлень, таких як ARP-запит.

1.3 Методи налаштування мережевого устаткування (Web UI, CLI)

Операційна система будь-якого пристрою це ні що інше, як інтерфейс між користувальницькими та апаратними компонентами цього пристрою. Операційна система для роботи з нею пропонує певний інтерфейс користувача для взаємодії з електронним пристроєм, як правило це GUI (Graphical user interface) (Рис. 1.5) та CLI (command-line interface) (Рис. 1.6). Деякі операційні системи надають GUI та CLI, тоді як інші пропонують лише CLI.

```
test_switch:5#config snmp system_name test_switch
Command: config snmp system_name test_switch

Success.

test_switch:5#show switch
Command: show switch

Device Type         : DES-3028 Fast Ethernet Switch
MAC Address         : 00-21-91-92-47-B3
IP Address          : 192.168.31.235 (Manual)
VLAN Name           : default
Subnet Mask         : 255.255.255.0
Default Gateway     : 0.0.0.0
Boot PROM Version   : Build 1.00-B04
Firmware Version    : Build 2.94.B22
Hardware Version    : A1
System Name         : test_switch
System Location     :
System Uptime       : 0 days, 0 hours, 2 minutes, 25 seconds
System Contact      :
Spanning Tree       : Disabled
GVRP                : Disabled
IGMP Snooping       : Disabled
VLAN trunk          : Disabled
802.1x              : Disabled
TELNET              : Enabled(TCP  23)
WEB                 : Enabled(TCP  80)
RMON                : Disabled
SSH                 : Enabled(TCP  22)
SSL                 : Disabled
Clipaging           : Enabled
Syslog Global State: Disabled
Dual Image          : Supported
Password Encryption Status : Disabled

test_switch:5#
```

Рис. 1.5 – Текстовий інтерфейс налаштування мережевого обладнання

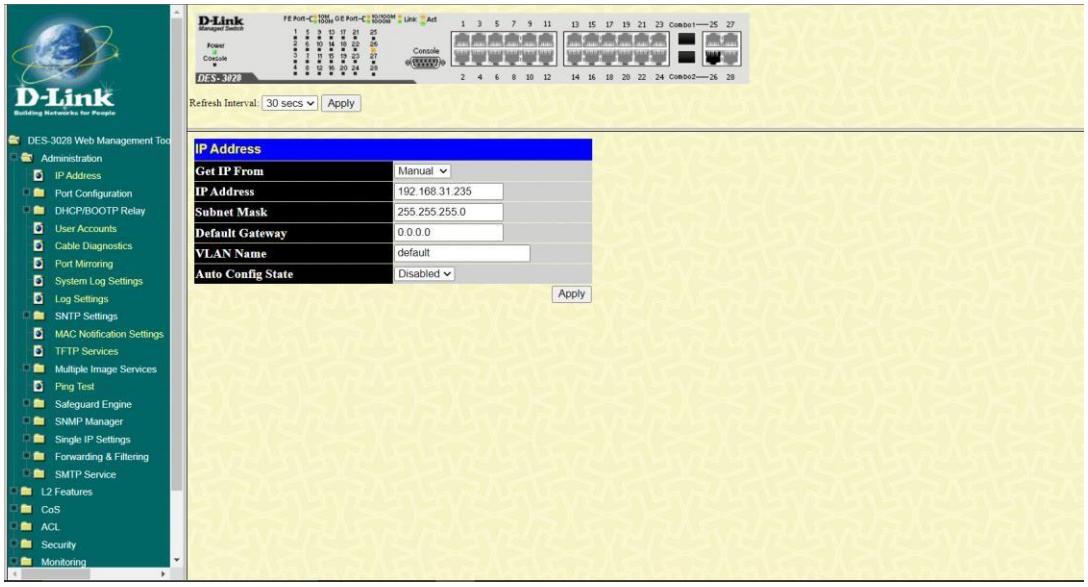


Рис. 1.6 – Графічний інтерфейс налаштування мережевого обладнання

GUI – це тип інтерфейсу, який дає змогу користувачам взаємодіяти з мережевими та іншими пристроями через графічні зображення та візуальні інтерфейси, на відміну від текстових інтерфейсів, заснованих на використанні тексту, текстовому наборі команд та текстовій навігації. CLI означає інтерфейс командного рядка, різновид текстового інтерфейсу користувача й комп'ютера, в якому інструкції комп'ютеру можна дати тільки введенням із клавіатури текстових рядків (команд).

Також відомий під назвою консоль. Як випливає з назви, необхідно написати команди для виконання певного завдання у системі, яка керується через CLI. З іншого боку, графічний інтерфейс користувача пропонує графіку, що складається із значків та зображень, які дозволяють користувачеві виконувати завдання безпосередньо взаємодіючи з ними в GUI-оболонці.

CLI вимагає досвіду в командах для виконання певного завдання, тоді як GUI може керуватися новачком, але GUI має більше споживання пам'яті та обчислювальних ресурсів в порівнянні з текстовим інтерфейсом, з GUI складніше організувати дистанційну роботу, а також неможливість автоматизації, якщо вона не була закладена автором програмного продукту.

1.4 Засоби підключення та роботи з мережевим обладнанням

Керовані мережеві пристрої можна конфігурувати за допомогою наступних інструментів та технологій: *локально* (за допомогою консольного порту, наприклад RS-232); *віддалено* (використовуючи протоколи Telnet, SSH, HTTP, HTTPS та інші).

Для роботи систем автоматичного конфігурування мережевого обладнання необхідно забезпечити керуючому хосту доступ до CLI керованого обладнання та налаштування доступу до нього.

1.4.1 Особливості послідовного порту в мережевому обладненні

За виключенням окремих випадків кожен маршрутизатор або комутатор має консольний порт, який використовується для підключення безпосередньо до комп’ютера для налаштування та керування. Як правило, консольний порт фізично реалізується роз’ємами RS-232, RJ45, miniUSB (Рис. 1.7).

Рис. 1.7 – Послідовний (консольний) порт на комутаторі

Консольний кабель (як правило виконаний у вигляді роз'єму RS232) або крос-кабель (як правило виконаний у вигляді роз'єму RJ45) використовується для підключення до маршрутизатора або консольного порту комутатора і зазвичай використовується під час початкової конфігурації, оскільки на пристрої ще не налаштоване підключення до мережі та віддалений доступ, наприклад по протоколу Telnet, SSH або HTTPS.

1.4.2 Протокол Telnet

Telnet (TELetype NETwork) — це мережевий протокол, який дозволяє користувачу спілкуватися з віддаленим пристроєм. Це протокол віртуального терміналу, який використовується в основному адміністраторами мережі для віддаленого доступу до пристроїв і керування ними. Адміністратор може отримати доступ до пристрою, підключившись по IP-адресі або імені хоста віддаленого пристрою. Telnet за замовчуванням використовує порт TCP 23.

У протоколі не передбачені використання шифрування та перевірки достовірності даних, тому він вразливий для будь-якого виду атак, до яких вразливий його транспорт, тобто протокол TCP. Так що варто враховувати, що сесія Telnet дуже беззахисна, якщо тільки не здійснюється в повністю контрольованій мережі або з застосуванням захисту на мережевому рівні (застосовуючи різні реалізації віртуальних приватних мереж). Через відсутність захисту як такого від Telnet як засобу управління операційними системами давно відмовилися і найчастіше використовується більш захищений протокол під назвою SSH.

1.4.3 Протокол SSH

Secure Shell, SSH - мережевий протокол рівня застосунків, що дозволяє проводити віддалене керування віддаленими пристроями і тунелювання TCP-з'єднань (наприклад, для передачі файлів). Схожий за функціональністю з протоколом Telnet, проте прококол SSH використовує аутентифікацію та шифрування, в тому числі і паролі, що передаються. SSH використовує шифрування. SSH використовує для цих цілей шифрування з відкритим ключем.

Як і Telnet, користувач, який отримує доступ до віддаленого пристрою, повинен мати встановлений клієнт SSH. На віддаленому пристрої повинен бути встановлений і запущений SSH-сервер. SSH за замовчуванням використовує порт TCP 22.

1.5 Системи автоматизації конфігурування мережевого обладнання

Мережеві пристрої, такі як комутатори, маршрутизатори та міжмережеві екрани, вже давно мають інтерфейси управління, особливо у професійних та корпоративних рішеннях. У цих пристроях завжди була популярна взаємодія на основі командного рядка. А отже маючи віддалений доступ до налаштувань мережевого обладнання можна застосувати системи автоматизації конфігурування даного мережевого обладнання.

Нещодавній сплеск інструментів автоматизації мережі ознаменував зміни в тому, як адміністратори будують мережі та керують ними. Хоча інструменти автоматизації серверів і додатків існують вже деякий час, інструменти, орієнтовані на мережу, стали популярними лише в останні кілька років.

Традиційно, ручна робота адміністраторів значною мірою забезпечує налаштування конфігурації та поточні зміни в мережі, але за останні роки з’явилися засоби автоматизації мережі, щоб зменшити кількість цих кроків вручну. Хоча засоби автоматизації мережі можуть використовувати різні підходи до автоматизації , усі вони спрямовані на скорочення часу адміністратора, який витрачається на прості та повторювані процеси налаштування мережевого устаткування.

Майже кожен сучасний інструмент автоматизації мережі може автоматизувати зміни конфігурації в середовищах багатьох виробників. Системи автоматизації досягають цього шляхом обробки команд під певний синтаксис командного рядка, а потім надсилають команди на зміну параметрів на кожен пристрій, який потребує змін в конфігурації. Замість того, щоб адміністратор мережі запускав сеанс Secure Socket Shell (SSH) для кожного маршрутизатора, комутатора та іншого

пристрою в мережі, щоб вручну змінювати текстові конфігурації, інструменти автоматизації створюють сценарії конфігурації, які досягають тієї ж мети за набагато менший час і з меншою кількістю помилок.

Іншим популярним методом для інструментів мережевої автоматизації для доступу та автоматизації конфігурацій мережевих пристроїв є API. Цей більш сучасний та елегантний спосіб взаємодії з мережевим обладнанням може скоротити час адміністратора, який витрачається на часті зміни мережі.

Деякі засоби автоматизації мережі виходять за рамки простої автоматизації змін конфігурації. Приклади інших функцій автоматизації мережі: *резервне копіювання конфігурації* дозволяє створювати заплановані резервні копії що шифруються та безпечно зберігаються на випадок, якщо конфігурацію мережевого устаткування потрібно повернути до попереднього стану або відновити, *контроль доступу до інструментів* керує доступом для тих, хто має повноваження вносити зміни в конфігурацію в певних сегментах мережі, також створюється журнал обліку, щоб показати повну історію змін конфігурації, *моніторинг і перевірка відповідності* автоматизують процес визначення, чи відповідає мережевий пристрій попередньо встановленим стандартам відповідності та регулювання, *оцінка вразливості* автоматично визначає параметри захищеності мікропрограми та конфігурації, які, як відомо, є вразливими, моніторинг продуктивності дозволяє аналізувати дані про продуктивність мережі на обладнанні, щоб надати рекомендації щодо конфігурування для подальшого підвищення продуктивності, *мережева оркестровка* автоматично виявляє мережеві пристрої для централізованого контролю та наскрізної координації доповнень і змін конфігурації локальної мережі.

Наразі існує досить багато систем автоматизації роботи з мережевим обладнанням і кожен інструмент має свої особливості, переваги та недоліки, розглянемо переваги інструментів мережевої автоматизації.

1.5.1 Система керування конфігураціями Puppet

Puppet — це інструмент керування конфігурацією який використовується для розгортання, налаштування та керування серверами. Він виконує такі функції, а саме: визначення окремих конфігурацій для кожного хоста, а також безперервна перевірка та підтвердження того, чи потрібна конфігурація на місці й не змінена (якщо вона змінена Puppet повернеться до необхідної конфігурації) на хості та забезпечує контроль над усіма налаштованими машинами, тому централізовані зміни (основні на головному сервері чи репозиторії) автоматично поширюються на всі пристрої.

Puppet використовує архітектуру клаєнт-серверну архітектуру у якій клієнт і сервер спілкуються через безпечний зашифрований канал за допомогою SSL.

Puppet має велику базу модулів для роботи з різним обладнанням, ось кілька наших найпопулярніших:

- cisco_ios — конфігурує пристрої Cisco Catalyst під керуванням IOS та IOS XE.
- panos — налаштовує брандмауери Palo Alto під керуванням PANOS.
- ciscopuppet — керує мережевими пристроями Cisco Nexus.
- F5 — керує балансувальниками навантаження LTM F5, надаючи типи та постачальники на основі REST.

1.5.2 Система керування конфігураціями Chef

Chef — це програмне забезпечення для автоматизації роботи з мережевим обладнанням, розроблене Opscode . Він написаний Ruby DSL (Domain Specific Language) та має клієнт-серверну архітектуру з встановленим на клієнті агентом. Цей інструмент автоматизації використовується для скорочення ручної роботи та подолання повторюваних рутинних завдань при роботі з мережею.

Програмне забезпечення Chef містить різні компоненти. Три з них є основними компонентами програмного забезпечення Chef:

- Chef-сервер
- Робочі станції (workstations)

- Вузли (nodes)

Термінологія Chef: Рецепт – це набір атрибутів, за допомогою яких ми керуємо інфраструктурою. Ми можемо встановити будь-який мережевий пристрій з атрибутами, збереженими в цих рецептах. Рецепти написані на Ruby. Усе, що нам потрібно запустити, створити або змінити, визначається за допомогою рецептів. Кулінарна книга – це місце, де рецепти групуються та зберігаються. По суті, це збірка рецептів. Існують сценарії, визначені в кулінарній книзі, а необхідні дії для цих сценаріїв також містяться в Кулінарній книзі. Іншими словами, значення та інформація для бажаних станів вузла зберігаються в кулінарній книзі.

Функціонування Chef Software схоже на програмне забезпечення Puppet . Як і в Puppet, в Chef Automation є агенти . Ці агенти отримують так звані “рецепти” та ресурси з Chef Server. Працює ця система наступним чином:

1. Код конфігурації написаний і перевірений зберігається на робочій станції.
2. Код надсилається на сервер Chef.
3. Chef Server розповсюджує так звані “кулінарні книги” керованим вузлам.
4. Вузли заходяться та синхронізуються з Chef Server, витягуючи оновлення з сервера Chef.

1.5.3 Система керування конфігураціями Ansible

Ansible – це система управління конфігураціями. Ansible дозволяє автоматизувати та спростити налаштування, обслуговування та розгортання серверів, мережевого обладнання, служб, ПЗ та ін. На даний момент існує кілька систем керування конфігураціями, однак для роботи з мережевим обладнанням найчастіше використовується Ansible. Пов'язано це з тим, що Ansible не вимагає встановлення агента на керовані хости. Особливо це актуально для пристроїв, які дозволяють працювати з ними тільки через CLI. Крім того, Ansible активно розвивається у бік підтримки мережевого обладнання, і в ньому постійно з'являються нові можливості та модулі для роботи з мережевим обладнанням. Одна з найважливіших переваг Ansible полягає в тому, що з ним легко почати працювати. Ansible в роботі

використовує push-модель: адміністратор сам ініціює процес застосування конфігурації, самі по собі клієнти нічого не змінюють.

Серед переваг Ansible в порівнянні з іншими системами можна виділити основні:

- Працює без встановлення агента на керовані хости
- Використовує SSH для підключення до керованих хостів
- Може виконувати дії локально на керуючому хості
- Використовує YAML для опису сценаріїв
- Містить безліч модулів (їх кількість постійно зростає)
- Виконує зміни за допомогою модулів, написаних мовою Python
- Можна писати свої модулі

Ansible був розроблений для того, щоб зробити код автоматизації переносним і повторно застосовуваним у всіх випадках коли це тільки можливо. У нашому розділі з управління інфраструктурою ми застосовували практично ідентичні плейбуки для налаштування інфраструктури різних постачальників і всі приклади були досить простими; ми можемо їх надалі застосовуючи ролі або видаляючи наявні повтори такого великого коду.

Ansible вирішує ці проблеми, а саме робить можливим написання плейбуків (сценаріїв автоматизації), які працюють у безлічі середовищ для досягнення в точності тих же речей з мінімальними зусиллями. Якщо відвідати індекс мережевих модулів у документації Ansible, можна виявити підтримку понад 50 типів систем, і це число зростає з кожною новою версією Ansible. При такому широкому і зростаючому діапазоні пристроїв, що підтримуються, мережному адміністратору стає простіше керувати всіма його пристроями з одного централізованого місця без необхідності в проприєтарних інструментах.

Приклади завдань, які допоможе вирішити Ansible:

- підключення по SSH до пристроїв (можливе паралельне підключення)
- відправка команд на пристрої
- конфігураційний файл може зберігатись в системі керування версіями на зразок git

- зручний синтаксис опису конфігурації для пристроїв
- можна розбивати пристрої на групи та відправляти певні команди на всю групу пристроїв
- підтримка шаблонів конфігурацій на основі Jinja2

Крім того, Ansible пропонує сотні готових модулів для роботи з мережевим обладнанням, які виконують велику частину важкої роботи для створення процесів автоматизації. Попередньо вбудовані модулі включають шаблони автоматизації для кількох постачальників, таких як Cisco, Dell, Extreme і Juniper. Ansible для мережевої автоматизації є особливо вдалим варіантом, якщо він уже використовується командами серверів, програм і розробників.

1.6 Порівняння існуючих систем автоматизації мережі

Сьогоднішні спеціалісти з системного адміністрування та devops мають вміти в середньому керувати набагато більшою кількістю серверів та мережевого обладнання, на яких розміщено набагато більша кількість додатків. Таким чином, такі інструменти, як Puppet і Ansible швидко стають важливими компонентами для управління таким великим числом серверів і мережевого обладнання.

Ansible, Puppet і Chef являють собою системи управління конфігураціями (SCM), необхідні для побудови та підтримки інфраструктури мережевого та серверного обладнання. Основні відмінності даних систем наведені у Таблиці 1.1.

	Chef	Puppet	Ansible
Архітектура	Клієнт-сервер	Клієнт-сервер	Безагентна, керування по SSH
Синтаксис	Ruby DSL	Puppet DSL	YAML
Мова програмування	Ruby – клієнт Erlang - сервер	Ruby	Python
Підключення до цільових хостів	SSH	Агент на цільовому хості підключається по SSL до Puppet-серверу	Ansible підключається до цільового хосту по SSH

Таблиця 1.1 – Порівняння існуючих систем автоматизації мережі

Ansible краще підходить тим, хто цікавиться конфігурацією в стилі YAML і розділяє філософію Ansible – бути максимально простим, при цьому досить швидко і паралельно керувати великим пулом машин. Крім того, ця філософія спрямована на використання існуючих функцій SSH встановлення агентів на цільові хости. Puppet більше підходить командам, що віддають перевагу DSL, який моделює системні ресурси послідовним, повторюваним чином. Це саме те, що виконує Puppet DSL разом із цілою екосистемою інструментів для того, щоб зробити роботу великих команд передбачуваною та легкою. Chef надає можливості для визначення інфраструктури як коду (IAC), але як і Puppet вимагає встановлення агента на цільові хости. Тому в нашому випадку перевага віддається саме Ansible, так як він не потребує встановлення на керовані хости агенту, здійснює керування за допомогою підключення по SSH та має масштабовану структуру.

1.7 Можливості Python для автоматизації мереж

Впродовж останніх кількох років мережева автоматизація набула великої популярності. В результаті сучасному інженеру доступний постійно зростаючий арсенал інструментів на різних платформах, які допомагають реалізувати автоматичну конфігурацію мережі та керування змінами в мережевій конфігурації.

Python є найбільш популярною мовою програмування, коли ми говоримо про автоматизацію мережі. Навіть мережевий гігант Cisco віддав перевагу мові програмування Python перед будь-якою іншою мовою для цілей автоматизації. Cisco також запровадила навчальну програму під назвою «Програмування для мережевих інженерів (PRNE)» і рекомендує її пройти, перш ніж шукати сертифікаційну підготовку на рівні асоційованих спеціалістів за версією DevNet Ie, DEVASC 200-901.

Якщо говорити про Python, то це мова програмування загального призначення, яка постачається з багатьма бібліотеками. Ці бібліотеки корисні для написання програм, орієнтованих на роботу з мережею. Усі ці бібліотеки є достатньо зрілими, добре перевіреними та мають сильну підтримку спільноти. Можна також

використовувати і інші мови програмування, такі як Java, C# тощо, але в даному випадку Python має ряд суттєвих переваг.

Отже, Python має такі переваги перед іншими мовами програмування в задачах мережевої автоматизації:

- Це легка у використанні та зріла мова.
- Його легко вчити, читати і писати.
- Python має широкий вибір бібліотек, що значно розширює функціонал та швидкість розробки.
- Python ідеально підходить для завдань сценаріїв, які ідеально підходять для автоматизації мережі.
- Будучи найпопулярнішою мовою програмування, Python має величезну спільноту. Отже, якщо виникнуть будь-які запитання щодо певної теми відповідь знаходиться доволі швидко.

1.7.1 Використання бібліотеки Paramiko

Paramiko – це реалізація протоколу SSHv2 на Python. Paramiko надає функціональність клієнта та сервера. Він зручніший у використанні, ніж наприклад, pexpect, але з більш вузькою функціональністю (підтримує тільки SSH). Головною перевагою використання paramiko є те, що модуль не вимагає встановлення агента на керовані хости. Особливо це актуально для пристроїв, які дозволяють працювати з ними тільки через CLI інтерфейс. Даний модуль може ефективно використовуватись для створення систем автоматизації роботи з мережевим обладнанням.

Код скрипта, що здійснює підкючення та виконання команди show switch за допомогою бібліотеки paramiko:

```
import paramiko
import sys
import time
import pdb
HOST = "123.456.789.101"
USER = "username"
PASS = "password"
PORT = 22
ITERATION = 3
```

```
def fn():
    client1=paramiko.SSHClient()
client1.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    client1.connect(HOST,username=USER,password=PASS,port=PORT)
    print("SSH connection to %s established: " + HOST)

    remote_conn = client1.invoke_shell()
    remote_conn.send("show switch")
    remote_conn.send("\n")
    remote_conn.send("n")
    remote_conn.send("\n")
    time.sleep(2)
    output = remote_conn.recv(10000)
    output = output.decode("utf-8")

    print(str(output))
    client1.close()
fn()
```

Текстовий вивід комутатора, що повертає нам результат виконання команди:

```
[iris@MiWiFi-R4AC-srv Desktop]$ python3 paramiko_example.py
SSH connection ti %s established: 192.168.31.235


                DGS-3120-24TC Gigabit Ethernet Switch
                       Command Line Interface

                     Firmware: Build 4.24.B001
          Copyright(C) 2014 D-Link Corporation. All rights reserved.

test:admin#show switch
Command: show switch

Device Type        : DES-3028 Fast Ethernet Switch
MAC Address        : 00-21-91-92-47-B3
IP Address         : 192.168.31.235 (Manual)
VLAN Name          : default
Subnet   Mask    :   255.255.255.0
Default Gateway : 0.0.0.0
Boot PROM Version : Build 1.00-B04
Firmware Version  :  Build 2.94.B22
Hardware Version : A1
System Name        : test
System Location    :
System Uptime      : 0 days, 0 hours, 0 minutes, 56 seconds
```

```
System Contact     :
Spanning Tree      : Disabled
GVRP               : Disabled
IGMP Snooping      : Disabled
VLAN trunk         : Disabled
802.1x             : Disabled
TELNET             : Enabled(TCP  23)
WEB                : Enabled(TCP  80)
RMON               : Disabled
SSH                : Enabled(TCP  22)
SSL                : Disabled
CLI Paging          : Disabled
Syslog Global State: Disabled
Dual Image         : Supported
Password Encryption Status : Disabled

test:admin#
[iris@MiWiFi-R4AC-srv Desktop]$
```

1.7.2 Використання бібліотеки Netmiko

Netmiko – це Python модуль, який спрощує використання paramiko для мережевих пристроїв, netmiko це «обгортка» навколо paramiko, яка орієнтована на роботу з мережевим обладнанням. Модуль Netmiko було створено для врахування особливостей окремих мережевих пристроїв при роботі з ними та бере не себе основну роботу з встановлення підключення з пристроєм та роботі з ним, розробнику залишається лише вказати набір команд та логіку роботи системи автоматизації. Наприклад, Netmiko розуміє, що під час зміни конфігурації для пристрою Cisco всі команди, які надсилаються, повинні спочатку починатися з підвищення до режиму глобальної конфігурації і, таким чином, автоматично видаватимуть термінал налаштування від імені поточного користувача, або у випадку роботи з комутаторами D’Link автоматично перед початком роботи вимикає функцію посторінкового виводу інформації clipaging. Ці, здавалося б, дрібні деталі в кінцевому підсумку призводять до різкого зниження складності написання сценаріїв автоматизації.

Приклад використання модулю Netmiko для виконання команди “show switch” на комутаторі D’Link, в даному випадку ми вказуємо тип пристрою, до

якого здійснюється підключення (dlink_ds) щоб модуль враховував особливості пристрою, з яким відбувається робота:

```
from netmiko import (
    ConnectHandler,
    NetmikoTimeoutException,
    NetmikoAuthenticationException,
)
def send_show_command(device, commands):
    result = {}
    try:
        with ConnectHandler(**device) as ssh:
            ssh.enable()
            for command in commands:
                output = ssh.send_command(command)
                result[command] = output
        return result
    except (NetmikoTimeoutException, NetmikoAuthenticationException) as
error:
        print(error)
if_name_== "_main__":
    device = {
        "device_type": "dlink_ds",
        "host": "10.25.210.225",
        "username": "iris",
        "password": "iris"
    }
    result = send_show_command(device, ["show switch"])
    print(result, width=120)
```

Модуль Netmiko дозволяє виконувати підключення до обладнання та виконання команд за допомогою протоколів SSH та Telnet. Модуль має широкий список підтримуваних платформ від найбільш популярних, таких як cisco_ios та huawei, до нішевих по типу ubiquiti_edgeswitch, кожен модуль містить інформацію про особливості даної серії пристроїв та особливості що враховуються при підключенні та роботі. Список доступних платформ можна дізнатись за посиланням (https://github.com/ktbyers/netmiko/blob/develop/PLATFORMS.md).

1.8 Взаємодія з обладнанням на основі поточних параметрів

На даний час існує багато систем автоматизації процесу конфігурування мережевого обладнання, основні з них це: Puppet, Chef, Salt, Jenkins та Ansible. Але

на думку автора роботи головним недоліком даних систем є те, що процес виконання певних налаштувань виконується без врахування поточних налаштувань та параметрів мережевого обладнання. Можливість автоматизованого конфігурування на основі поточних параметрів та налаштувань дозволить створювати більш гнучкі системи автоматизації роботи з мережевим обладнанням, що не виконують певний прописаний набір команд, а проводять аналіз поточних налаштувань і тільки на їх основі виконують команди на мережевому обладнанні.

Обробка отриманого текстового виводу мережевого обладнання може значно розширити можливості систем автоматизації мережевого обладнання, стає можливим робота з мережевим обладнанням на основі логічного аналізу виводу певних команд діагностики, сервісу, виводу поточних параметрів та налаштувань. В даному випадку є два підходи, перший підхід передбачає отримання автоматизованою системою файлу поточної конфігурації мережевого обладнання з наступною його обробкою команда за командою, це дозволяє отримати поточні налаштування пристрою, стан певних системних параметрів та функцій, але даний метод не передбачає отримання інформації про поточні параметри роботи та стан системи в реальному часі, другим підходом є виконання певних сервісних команд, що виводять поточну інформацію про пристрій, поточні налаштування та стан. Як приклад, нижче наведені команди show switch для D'Link (Рис. 1.8) та show version для Cisco (Рис. 1.9).

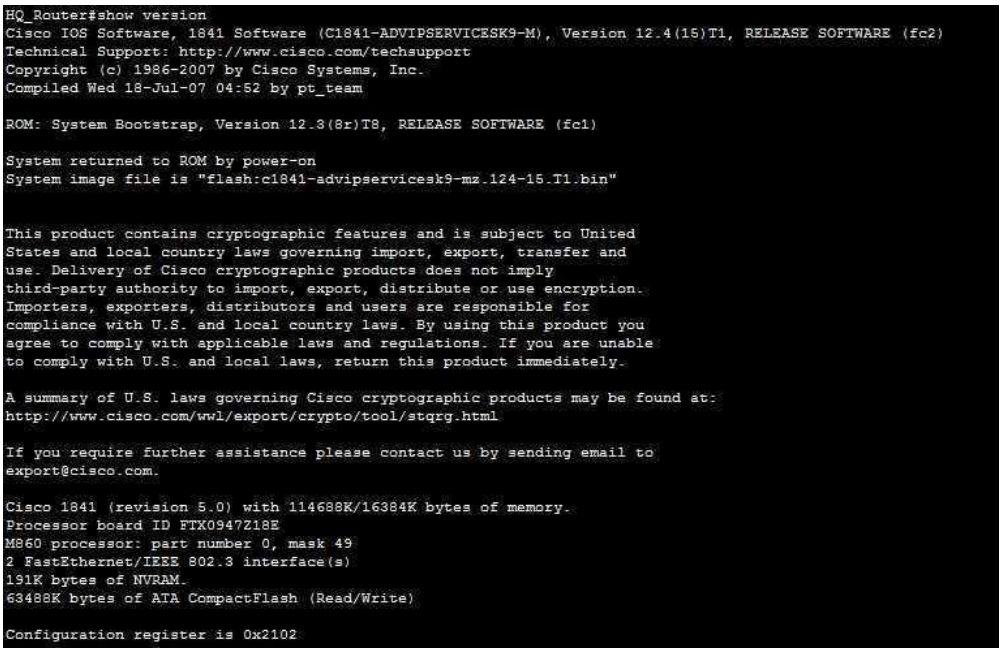


Рис. 1.8 – Вивід команди show version на пристрої Cisco

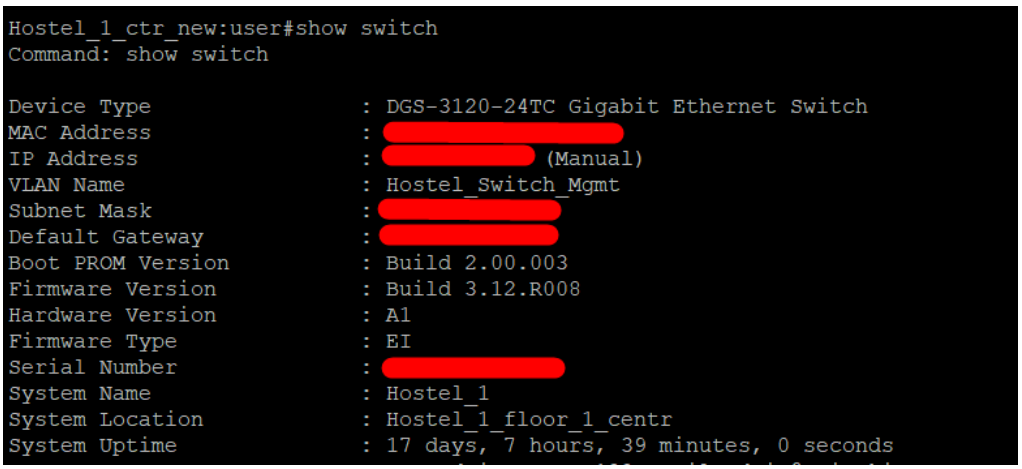


Рис. 1.9 – Вивід команди show switch на пристрої D’Link

Найбільш раціональним є варіант використання другого підходу, так як для систем автоматизації дуже актуальними є поточні параметри мережевого пристрою. Але постає задача логічного розбору виводу мережевого пристрою, так як дані, що ми отримуємо, є просто частково структурованим текстом. Цю задачу можна вирішити використанням регулярних виразів, що будуть шукати в тексті певні поля, що задовольняють вимоги регулярного виразу. Це велика кількість коду, що є не зручним для читання та модифікацій, оптимальним рішенням є розбір виводу за певним шаблоном, що створюється один раз і легко піддається зміні/модифікації.

1.8.1 Застосування текстового аналізатора TextFSM

В компанії Google для роботи з власним мережевим обладнанням був створений інструмент TextFSM. *TextFSM* — це кінцевий автомат на основі шаблонів для розбору напівформатованого тексту в таблиці. Він був розроблений Google і розміщений на Google Code. Він легко налаштовується, оскільки працює з окремими визначеннями шаблонами, які містять змінні та правила з регулярними виразами для обробки даних. Ця бібліотека дуже корисна для аналізу будь-якого текстового виводу CLI з мережевих пристроїв. Google TextFSM — це модуль Python, тому його можна легко використовувати в сценаріях Python або як окремий інструмент. Це зручний спосіб перетворити вихід CLI у структуровані дані.

Розглянемо декілька прикладів розбору виводу команд для комутаторів D’Link за допомогою TextFSM. Для початку виконаємо команду “show switch” на комутаторі та збережемо її вивід. Команда виводить основні дані про мережевий пристрій:

```
Command: show switch

Device Type        : DES-3028 Fast Ethernet Switch
MAC Address        : 00-21-91-92-47-B3
IP Address         : 192.168.31.235 (Manual)
VLAN Name          : default
Subnet   Mask    :   255.255.255.0
Default Gateway : 0.0.0.0
Boot PROM Version : Build 1.00-B04
Firmware Version  :  Build  2.94.B22
Hardware Version : A1
System Name        : test
System Location    :
System Uptime      : 0 days, 0 hours, 0 minutes, 56 seconds
System Contact     :
Spanning Tree      : Disabled
GVRP               : Disabled
IGMP Snooping      : Disabled
VLAN trunk         : Disabled
802.1 x            : Disabled
TELNET             : Enabled(TCP  23)
WEB                : Enabled(TCP  80)
RMON               : Disabled
SSH                : Enabled(TCP  22)
SSL                : Disabled
CLI Paging          : Disabled
Syslog Global State: Disabled
Dual Image         : Supported
Password Encryption Status : Disabled
```

Python скрипт приймає текстовий вивід команди з комутатора (виконання команди та отримання виводу також можна автоматизувати за допомогою інструментів Python).

```
import textfsm
from pprint import pprint
import yaml

with open('output.txt') as f:
    show_switch = f.readlines()

with open('show_switch.template') as template:
    fsm = textfsm.TextFSM(template)
    result = fsm.ParseText(show_switch)
```

```
zip_iterator = zip(fsm.header, result[0])
dictionary = dict(zip_iterator)
pprint(dictionary)

with open('show_switch.yaml', 'w') as file:
    documents = yaml.dump(dictionary, file)
```

TextFSM вимагає визначення, як аналізувати вихідні дані. Це описується в окремому файлі за допомогою визначеної структури та деяких правил із регулярними виразами. Логіку, як TextFSM розвиває шаблон, трохи важко зрозуміти вперше, але я спробую окреслити її на основі шаблону, який я використовую в цьому прикладі.

Розглянемо з шаблону TextFSM, який для даної команди виглядає так:

```
Value  Device  (\S+)
Value Mac (\w+(-\w+){5})
Value IP (\d+(\.\d+){3})
Value VLAN (\S+)
Value Subnet (\d+(\.\d+){3})
Value Gateway (\d+(\.\d+){3})
Value Boot (Build \S+)
Value Firmware (Build \S+)
Value Hardware (\S+)
Value Name (\S+)
Value Location (\S+)
Value Telnet (\S+.+)
Value SSH (\S+.+)
Value Clipaging (\S+)

Start
  ^.*Device Type.*: ${Device}
  ^.*MAC Address.*: ${Mac}
  ^.*IP Address.*: ${IP}
  ^.*VLAN Name.*: ${VLAN}
  ^.*Subnet Mask.*: ${Subnet}
  ^.*Default Gateway.*: ${Gateway}
  ^.*Boot PROM Version.*: ${Boot}
  ^.*Firmware Version.*: ${Firmware}
  ^.*Hardware Version.*: ${Hardware}
  ^.*System Name.*: ${Name}
  ^.*System Location.*: ${Location}
  ^.*TELNET.*: ${Telnet}
  ^.*SSH.*: ${SSH}
```

```
^.*Clipaging.*: ${Clipaging} -> Record
```

Файл шаблону TextFSM структурований у два основні розділи:

1. Визначення значень, які визначають поля та типи, які слід витягти
2. Визначення стану, що визначає правила аналізу вмісту

Синтаксичний аналізатор TextFSM створить записи таблиці на основі заданого визначення шаблону під час виконання. Якщо регулярний вираз із визначенням збігається зі значенням, то значення витягуються на основі пов'язаного визначення , а потім значення додається до буфера. TextFSM продовжить аналіз порядково (за замовчуванням). Якщо значення вже призначено, вміст буде замінено.

Після регулярного виразу можна визначити дію, розмежовану символом ->. За замовчуванням це дія Next, яка вказує парсеру завершити роботу з цим рядком виводу, перейти до наступного та використати наступне правило, якщо поточний рядок відповідає регулярному виразу.

В кінці ми змінимо дію за замовчуванням на Record, щоб доручити парсеру зберегти поточний запис і створити новий, який буде оброблений за цим самим шаблоном. Якщо ми не будемо використовувати цю дію, TextFSM створить лише один запис з усього нашого вхідного файлу.

Після виконання скрипту Python ми отримали файл show_switch.yaml. Файл містить структуровані дані, до яких можна програмно отримати доступ та обробляти їх в процесі виконання скрипта автоматизації. Вміст файлу show_switch.yaml:

```
Boot: Build 1.00-B04
Clipaging: Disabled
Device: DES-3028
Firmware: Build 2.94.B22
Gateway: 0.0.0.0
Hardware: A1
IP: 192.168.31.235
Location: ''
Mac: 00-21-91-92-47-B3
Name: test
SSH: Enabled(TCP  22)
Subnet: 255.255.255.0
Telnet: Enabled(TCP  23)
```

```
 VLAN: default
```

Також наведемо приклад отримання даних про VLAN (команда show vlan), для комутаторів D’Link. Отримана відповідь для даної команди виглядає наступним чином:

```
Command: show vlan


VLAN Trunk State        : Disabled
VLAN Trunk Member Ports :


VID             : 1           VLAN  Name      :  default
VLAN  Type      :  Static     Advertisement    :  Enabled
Member Ports : 1-5
Static Ports    : 1-5
Current Tagged Ports   :
Current Untagged Ports : 1-5
Static Tagged Ports    :
Static Untagged Ports : 1-5
Forbidden Ports        :

VID             : 12          VLAN Name       : ada
VLAN Type       : Static      Advertisement   : Disabled
Member Ports    : 7
Static Ports    : 7
Current Tagged Ports   : 7
Current Untagged Ports :
Static Tagged Ports    : 7
Static Untagged Ports :
Forbidden Ports        :

VID             : 24          VLAN Name       : basic
VLAN Type       : Static      Advertisement   : Disabled
Member Ports    : 9-12,24
Static Ports    : 9-12,24
Current Tagged Ports   : 9-12,24
Current Untagged Ports :
Static Tagged Ports    : 9-12,24
Static Untagged Ports :
Forbidden Ports        :

Total Entries  : 3
```

Шаблон текстового аналізатора TextFSM, що містить набір правил згідно яких відбувається обробка відповіді заданої команди, виглядає наступним чином:

```
Value VID (\S+)
Value Name (\S+)
Value Type (\S+)
Value Advertisement (\S+)
Value Member_ports ( \S+|)
Value Current_Tagged_Ports ( \S+|)
Value Current_Untagged_Ports ( \S+|)
Value Static_Tagged_Ports ( \S+|)
Value Static_Untagged_Ports ( \S+|)

Start
  ^.*VID.*: ${VID} .*VLAN Name.* : ${Name}
  ^.*VLAN Type.*: ${Type} .*Advertisement.* : ${Advertisement}
  ^.*Member Ports.*:${Member_ports}
  ^.*Current Tagged Ports.*:${Current_Tagged_Ports}
  ^.*Current Untagged Ports.*:${Current_Untagged_Ports}
  ^.*Static Tagged Ports.*:${Static_Tagged_Ports}
  ^.*Static Untagged Ports.*:${Static_Untagged_Ports} -> Record
```

Отримані дані після обробки:

```
[{'Advertisement': 'Enabled',
  'VID': '1',
  'VLAN_Current_Tagged_Ports': '',
  'VLAN_Current_Untagged_Ports': ' 1-5',
  'VLAN_Member_ports': ' 1-5',
  'VLAN_Name': 'default',
  'VLAN_Static_Tagged_Ports': '',
  'VLAN_Static_Untagged_Ports': ' 1-5',
  'VLAN_Type': 'Static'},
 {'Advertisement': 'Disabled',
  'VID': '12',
  'VLAN_Current_Tagged_Ports': ' 7',
  'VLAN_Current_Untagged_Ports': '',
  'VLAN_Member_ports': ' 7',
  'VLAN_Name': 'ada',
  'VLAN_Static_Tagged_Ports': ' 7',
  'VLAN_Static_Untagged_Ports': '',
  'VLAN_Type': 'Static'},
 {'Advertisement': 'Disabled',
  'VID': '24',
  'VLAN_Current_Tagged_Ports': ' 9-12,24',
  'VLAN_Current_Untagged_Ports': '',
  'VLAN_Member_ports': ' 9-12,24',
  'VLAN_Name': 'basic',
  'VLAN_Static_Tagged_Ports': ' 9-12,24',
  'VLAN_Static_Untagged_Ports': '',
  'VLAN_Type': 'Static'}]
```

Використання аналізатора TextFSM є кращим, ніж проста рядкова обробка за допомогою власноруч описаних регулярних виразів, що в такому випадку ускладнює розуміння, так як кожен інженер має власний підхід до написання регулярних виразів. Використання стандартизованих шаблонів дає краще уявлення про те, як оброблятиметься отриманий текст у структуровані дані. Сам шаблон ззвоні повторює структуру вхідних даних. Також дані шаблони можуть бути універсальними для різних випадків.

2 ПРАКТИЧНА ЧАСТИНА

2.1 Процедура інсталяції Ansible

Ansible потрібно встановлювати лише на тій машині, з якою виконуватиметься керування пристроями.

Вимоги до керуючого хосту:

- підтримка python 3 (тестувалося на 3.9)
- Windows не може бути керуючим хостом

Підтримувані операційні системи:

- Red Hat Enterprise Linux 7 64-розрядна версія та вище
- CentOS 7 64-розрядна версія та вище
- Ubuntu 14.04 LTS 64-розрядна версія та вище
- MacOS High Sierra та вище

В процесі виконання даної роботи була проведена інсталяція та тестування на операційних системах CentOS 8 та MacOS Big Sur 11.6.1.

Для ОС CentOS:

1. yum install epel-release
2. yum install ansible

Для OS MacOS:

1. pip3 install ansible

Надалі в якості Control machine для Ansible буде використовуватись ОС CentOS 8 64-bit. Команди та процес встановлення програмного пакету Ansible наведені на (рис. 2.1) та (рис.2.2).

```
[root@MiWiFi-R4AC-srv iris]# yum install epel-release
CentOS Linux 8 - AppStream                      12 kB/s | 4.3 kB     00:00
CentOS Linux 8 - AppStream                     6.8 MB/s | 8.2 MB     00:01
CentOS Linux 8 - BaseOS                         29 kB/s | 3.9 kB     00:00
CentOS Linux 8 - BaseOS                        3.5 MB/s | 3.5 MB     00:00
CentOS Linux 8 - Extras                        9.9 kB/s | 1.5 kB     00:00
Extra Packages for Enterprise Linux 8 - x86_64   44 kB/s |  31 kB     00:00
Extra Packages for Enterprise Linux 8 - x86_64  5.7 MB/s |  11 MB     00:01
Extra Packages for Enterprise Linux Modular 8 - x86_64
Extra Packages for Enterprise Linux Modular 8 - x86_64
Package epel-release-8-11.el8.noarch is already installed.
Dependencies resolved.
```

Рис. 2.1 – Встановлення EPEL

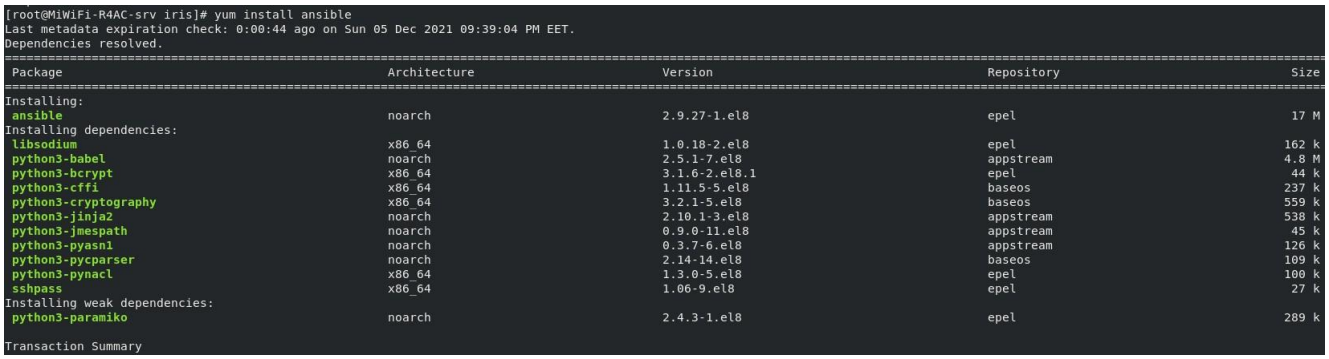


Рис. 2.2 – Встановлення Ansible

Перевіримо наявність встановленого ПЗ Ansible командою: ansible –version

У Ansible наявні певні терміни та засоби, у даному списку перераховані основні з них:

- **Control machine** - керуючий хост. Сервер Ansible, з якого відбувається керування іншими хостами, серед вимог до керуючого хосту: 1. підтримка Python 3; 2. Windows не може бути керуючим хостом;
- **Manage node** - керовані хости, мережеве обладнення, сервери, тощо
- **Inventory** – інвентарний файл (Рис. 2.3). У цьому файлі описуються хости, групи хостів, а також їх змінні;
- **Playbook** - файл сценаріїв (Рис. 2.4)., у якому міститься опис завдань
- **Play** – сценарій (набір завдань), зв'язує задачі з хостами, для яких ці завдання треба виконати;
- **Task** – завдання, викликає модуль із зазначеними параметрами та змінними;
- **Module** – модуль Ansible. Реалізує певні функції.

```
1  [cisco]
2  172.16.0.2
3
4  [cisco:vars]
5  ansible_network_os=ios
6  ansible_ssh_user=root
7  ansible_password=123456
```

Рис. 2.3 – Приклад інвентарного файлу

```
---
- name: Configure VLANs and ports
  hosts: cisco
  connection: ansible.netcommon.network_cli
  gather_facts: no
  tasks:

    # Override device configuration of all VLANs with provided configuration
    - name: Configure VLANs
      cisco.ios.ios_vlans:
        config:
        - name: sales
          vlan_id: 20
        state: overridden

```

Рис. 2.4 – Приклад файлу сценаріїв

Перевірка наявності встановленого пакету Ansible та його поточної версії виконується командою “ansible --version”:

```
> ansible --version
ansible 2.9.27
config file = /etc/ansible/ansible.cfq
configured module search path = ['/root/.ansible/plugins/mod-
ules','/usr/share/ansible/plugins/modules']
ansible python module location = /usr/lib/python3.6/site-packages/ansible
executable location = /bin/ansible
python version = 3.6.8 (default, Sep 10 2021, 09:13:53) [GCC 8.5.0 20210514 (Red
Hat 8.5.0-3)]
```

Налаштування інвентарного файлу. Інвентарний файл містить інформацію про хости, які будуть контролюватись за допомогою Ansible. У файл інвентаризації можливо включити до кількох сотень хостів і розділити їх в групи і підгрупи. У інвентарному файлі також можливо визначити змінні, що діють тільки для певних хостів або груп, щоб їх можна було використовувати з плейбуками та шаблонами. Щоб змінити заданий за замовчуванням вміст інвентарного файлу Ansible необхідно відкрити файл /etc/ansible/hosts у редакторі на контрольній машині Ansible командою: nano /etc/ansible/hosts

Файл інвентаризації, використовуваний Ansible за замовчуванням, містить ряд прикладів, які можна використовувати як зразки при налаштуванні інвентарного файлу.

```
[cisco]
switch1 ansible_host = 10.25.210.226
switch2 ansible_host = 10.25.210.227

[cisco:vars]
```

```
ansible_network_os=ios
ansible_ssh_user=iris
ansible_password=iris
```

Як тестовий приклад у [cisco] визначається група з двох комутаторів, кожен з яких має власний ідентифікатор: switch1 та switch2, та IP-адреси. Підгрупа [cisco:vars] задає параметри хостів групи [cisco], які будуть діяти для всіх хостів, включених у цю групу, а саме тип системи, логін та пароль для підключення по SSH. Нажаль спроби генерації та застосування SSH-ключів на наявних комутаторах не дали позитивного результату, тому надалі буде використовуватись автентифікація по логіну та паролю користувача.

Параметри Ansible можна змінювати у конфігураційному файлі (sudo nano /etc/ansible/ansible.cfg). У конфігураційному файлі можна змінювати безліч параметрів, які впливають на роботу Ansible. Повний список параметрів та їх опис можна знайти у документації, а приклад конфігураційного файлу винесено в Додаток А.

2.1.1 Тестування взаємодії Ansible з наявним обладнанням

Після налаштування інвентарного файлу можливо перевірити здатність Ansible підключатися до цільового мережевого обладнання і запускати команди через SSH. Запустіть на локальний комп'ютер або узле управління наступною командою Ansible: Тестування виконується командою: ansible all -m ping -u iris, де all – всі елементи інвентарного файлу, iris – ім’я користувача, від якого виконується вхід. Ця команда використовує вбудований модуль ping контрольної машини Ansible для тестування підключень до хостів, що знаходяться у інвентарному файлі.

Модуль ping тестує наступне:

- доступність хостів;
- наявність діючих облікових записів SSH;
- здатність виконувати команди на хостах.

Спробуємо перевірити можливість підключення до мережевого комутатора Cisco Catalyst 2960 та можливість виконання на ньому автоматизованих команд керування. Також необхідно встановити відповідний модуль ansible для Cisco ios командою: ansible-galaxy collection install cisco.ios

Тестування функціонування Ansible проводилось на комутаторі Cisco Catalyst 2960, схема організації наведена на (рис. 2.5).

Рис. 2.5 – Схема організації мережі

Вміст інвентарного файлу:

```
[cisco]
192.168.31.234

[cisco:vars]
ansible_connection=ssh
ansible_network_os=ios
ansible_user=iris
ansible_password=iris
```

Вивід списку пристроїв, доступних в інвентарному файлі:

```
ansible all –list-hosts
  hosts (1):
    192.168.31.234
```

Результати виконання команди *ping*, що перевіряє доступність обладнання:

```
ansible -i hosts all -m ping
192.168.31.234 | SUCCESS => {
       “ansible facts”: {
          “discovered_interpreter_python”: “/usr/libexec/platform-python”
       },
       “changed”: false,
       “ping”: pong
}
```

Відповідь "pong" від хосту означає, що хост доступний і виконувати команди та плейбуки Ansible на цьому хості можливо. У Ansible існує поняття facts (факти). Це і є інформація про машини, на яких відбувається запуск. Факти - статична інформація про сервер: операційна система, процесор, пам'ять, налаштування мережі, змінні оточення, інтерфейси тощо. У плейбуках вона визначається у вигляді звичайних змінних, які можна використовувати та підставляти.

Тепер можливо використовувати команди Ansible як для окремих хостів, так і для груп і підгруп хостів (за умови їх доступності). Наприклад, можливо перевірити інтерфейси кожного хосту в групі за допомогою наступної команди:

```
ansible cisco -m raw -a ‘show interface status’ –ask-pass -c ssh

SSH password:
192.168.31.234 | CHANGED | rc=0  >>
Port      Name    Status          Vlan   Duplex    Speed        Type
Fa0/1             Notconnect      1      auto      auto     10/100BaseTX
Fa0/2             Notconnect      1      auto      auto     10/100BaseTX
Fa0/3             Notconnect      1      auto      auto     10/100BaseTX
Fa0/4             Notconnect      1      auto      auto     10/100BaseTX
Fa0/5             notconnect      1      auto      auto     10/100BaseTX
..
```

Також були проведені спроби налаштування роботи Ansible з комутаторами D’Link серії DGS 3120, але нажаль засобами Ansible підключитись по SSH до них неможливо. Модуль paramiko нажаль теж не показав працездатності, хоча окремо він працює, що показано в Додатку Б, тому для таких випадків, коли вбудована оболонка не дозволяє відправити команди через sshpass, запропоновани використовувати вбудований модуль Telnet, приклад наведено у Додатку В.

2.1.2 Конфігурування мережевого обладнання за допомогою Ansible

Основні дії по автоматизації в Ansible описуються у Playbook за допомогою мови YAML, вона досить зручна для вивчення, розуміння та написання сценаріїв автоматизацій. Розглянемо приклад створення vlan з id: 20 та занесення порту 2 у нього.

Playbook з назвою vlan.yml виглядає наступним чином, він містить в собі набір дій, що має виконати пакет Ansible та додаткові параметри:

```
---
- name: Configure VLANs and ports
  hosts: cisco
  connection: ansible. netcommon. network cli
  gather facts: no
  tasks:

  # Override device configuration of all VANs with provided configuration
     -  name: Configure VLANS
        cisco.ios.ios_vlans:
        config:
           - name: sales
              vlan id: 20
        state: overridden

  # Replaces device configuration of listed 12 interfaces with provided
configuration
```

```
-  name: Configure switch ports
   cisco. ios. ios l2 interfaces:
     config:
     - name: "{{ item }}"
        mode: access
        access:
           vlan: 20
      state: replaced
  with items:
    - FastEthernet0/2
```

Запуск виконання playbook-а vlan.yml відбувається наступною командою "*ansible-playbook vlan.yaml*", повідомлення про успішне виконання наведено на (Рис. 2.6)

```
[root@localhost ansible]# ansible-playbook vlan.yml

PLAY [Configure VLANs and ports] ***********************************************

TASK [Configure VLANs] *********************************************************
changed: [192.168.31.234]

TASK [Configure switch ports] **************************************************
changed: [192.168.31.234] => (item=FastEthernet0/2)

PLAY RECAP *********************************************************************
192.168.31.234             : ok=2    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Рис. 2.6 – Повідомлення про успішне виконання playbook-а

Результат виконання можна переглянути на комутаторі (Рис. 2.7), був створений VLAN з ID 20, назвою "sales", також в даний VLAN внесено порт Fa0/2:

```
S1#show vlan brief

VLAN Name                             Status    Ports
---- -------------------------------- --------- -------------------
1    default                          active    Fa0/1, Fa0/3, Fa
                                                Fa0/20, Fa0/21, F
                                                Fa0/37, Fa0/38, F
20   sales                            active    Fa0/2
```

Рис. 2.7 – Наявні VLAN-и на тестовому комутаторі після виконання сценарію автоматизованого конфігурування

Як бачимо задача автоматизації виконана успішно, створено новий vlan та додано порти, дану задачу можна автоматизовано виконувати одночасно на десятках комутаторів, що спростить налаштування та зменшить загальні витрати часу, необхідного для налаштування. Це один невеликий приклад використання сценаріїв автоматизації, але можливо зробити повну автоматизовану конфігурацію мережевого обладнання та автоматичне збереження файлів конфігурації. Головна

ціль цього - показати, що будь-яке завдання з управління мережею, що повторюється, можна автоматизувати, і що Ansible не просто дозволяє керувати конфігураціями, а й допомагає позбутися рутини і заощадити час.

2.2 Розробка шаблонів аналізатора TextFSM для мережевого устаткування DLink

В попередніх розділах було розглянуто переваги, недоліки та особливості наявних систем автоматизації роботи з мережевим обладнанням. Одним з недоліків на думку автора даної роботи є те, що дані системи при роботі з мережевим устаткуванням не враховують так званий feedback, а саме текстову відповідь, яку повертає пристрій після виконання тієї чи іншої команди. Дана відповідь містить багато корисного навантаження, яке варто враховувати в роботі з даним обладнанням.

Для вирішення задачі розбору було обрано найефективніший на думку автора роботи інструмент, а саме текстовий аналізатор TextFSM, розроблений в Google. Але для аналізу текстової відповіді комутатора необхідно мати відповідний шаблон, за допомогою якого TextFSM перетворює напівструктурований текстовий вивід в структурований набір даних (у вигляді словника з парою “ключ”:”значення”) з яким можна легко працювати, отримуючи необхідну інформацію.

Розробляючи систему автоматизації мережі в першу чергу потрібно враховувати наявне обладнання мережі, а саме його виробника та модель. Від цих двох основних параметрів залежать особливості командного рядка та виводу, отриманого в результаті виконання тієї чи іншої команди.

2.2.1 Основні моделі наявних пристроїв, відмінності в їх CLI.

Розробляючи систему автоматизованого конфігурування мережевого обладнання автор роботи брав до уваги пристрої, наявні в мережевій інфраструктурі університету, а саме комутатори від вендора D’Link серій DGS та DES. Тому для створення системи автоматизації конфігурування мережевого обладнання щоб

враховувати корисне навантаження необхідно розробити шаблони текстового аналізатора TextFSM для обробки виводу основних команд інтерфейсу налаштувань D'Link CLI.

На щастя, компанія-розробник D'Link має стандартизований набір команд та інтерфейс командного рядка налаштувань, а також опублікована детальна документація до нього. Опираючись на них можна розробити шаблони аналізатора TextFSM для отримання необхідних даних з текстового виводу. Але є декілька моментів, на які варто звернути увагу, а саме те, що вивід не завжди однаковий на різних моделях і прошивках. Якщо мова йде про мережеве обладнання Cisco, то дана компанія має чіткий стандарт командного рядка, який не змінюється з виходом оновлень програмного забезпечення (за виключенням доданих нових команд та технологій, які теж входять в нові ревізії стандарту), то з власного досвіду і спостережень було помічено певні відмінності в інтерфейсі командного рядка комутаторів D'Link. Як приклад, можна навести вивід команди "show switch" на комутаторах D'Link серій DGS та DES. Так, для серії DES в виводі ми маємо поле "Clipaging", але на комутаторах серії DGS це поле підписане як "CLI Paging". Ці відмінності в інтерфейсі командного рядка необхідно детально враховувати при розробці шаблонів текстового аналізатора TextFSM і ніяк, окрім практичної роботи і "живого" тестування на обладнанні дізантись про ці особливості нема можливості, так як вони не є задокументованими. Надалі при розробці шаблонів TextFSM для комутаторів D'Link ці особливості будуть описані та враховані.

2.2.2 Основні команди, створення шаблонів аналізатора

Докуметація для командного рядка мережевих пристроїв D'Link містить величезну кількість команд, серед них можна виділити декілька груп, а саме: команди налаштування без відповіді, команди налаштування з відповіддю та команди "show", що виводять різноманінту поточну інформацію про пристрій та його актуальні налаштування, та команди діагностики на зразок "cable diag".

Під час розробки шаблонів аналізатора TextFSM біли обрані команди, що в результаті свого виконання на пристрої повертають найбільш корисну та необ-

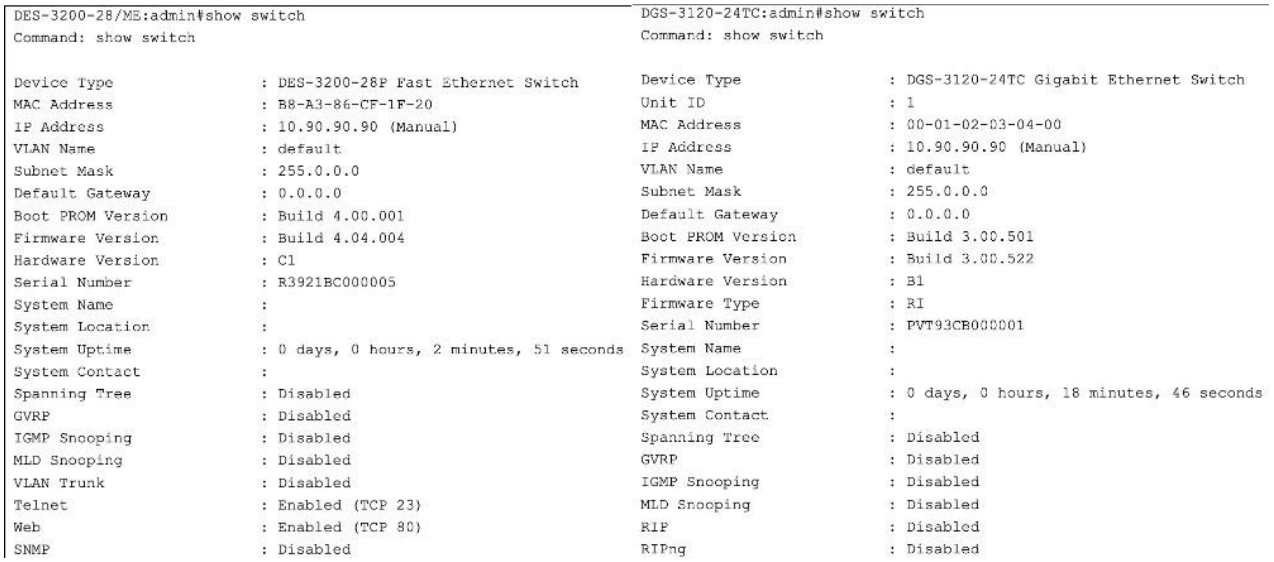
хідну інформацію, яку можна враховувати при автоматичному конфігуруванні мережевого обладнання. Це такі команди, як: show switch, show ports, show ports description, show arpentry, show fdb, show iproute, show ipif, show ip-mac-port binding, cable diag. Також є необхідним шаблон обробки результату виконання команд налаштування та особливих випадків (вказує на успішне виконання команди, помилку виконання, помилку вводу команди, синтаксичні помилки введених команд).

В розділі 1.8.2 детально розписувався процес створення шаблону TextFSM для обробки виводу команди “show switch” інтерфейсу командного рядка D’Link. Для використання даного шаблону на серіях D’Link DES та DGS зі стандартним CLI необхідно врахувати одну особливість, а саме назву поля, яке вказує на статус опції посторінкового виводу (CLI paging). Вирішенням даної проблеми є вказання двох опцій запису даних з шаблоном в змінну, якщо один шаблон не спрацьовує, то спрацьовує інший і відбувається запис даних в зміну Clipaging:

```
Value Clipaging (\S+)

Start
  ^.*Clipaging.*: ${Clipaging}
  ^.*CLI Paging.*: ${Clipaging} -> Record
```

Також серед помічених відмінностей в інтерфейсі командного рядка є те, що порядок виводу не завжди співпадає на різних серіях і версіях пристроїв (Рис. 2.8), але ця особливість не впливає на те, як текстовий аналізатор TextFSM виконує обробку текстового виводу, дані заносяться в відповідні їм поля згідно визначених в шаблоні правил.



```
DES-3200-28/ME:admin#show switch
Command: show switch

Device Type              : DES-3200-28P Fast Ethernet Switch
MAC Address              : B8-A3-86-CF-1F-20
IP Address               : 10.90.90.90 (Manual)
VLAN Name                : default
Subnet Mask              : 255.0.0.0
Default Gateway          : 0.0.0.0
Boot PROM Version        : Build 4.00.001
Firmware Version         : Build 4.04.004
Hardware Version         : C1
Serial Number            : R3921BC000005
System Name              :
System Location          :
System Uptime            : 0 days, 0 hours, 2 minutes, 51 seconds
System Contact           :
Spanning Tree            : Disabled
GVRP                     : Disabled
IGMP Snooping            : Disabled
MLD Snooping             : Disabled
VLAN Trunk               : Disabled
Telnet                   : Enabled (TCP 23)
Web                      : Enabled (TCP 80)
SNMP                     : Disabled
```

```
DGS-3120-24TC:admin#show switch
Command: show switch

Device Type              : DGS-3120-24TC Gigabit Ethernet Switch
Unit ID                  : 1
MAC Address              : 00-01-02-03-04-00
IP Address               : 10.90.90.90 (Manual)
VLAN Name                : default
Subnet Mask              : 255.0.0.0
Default Gateway          : 0.0.0.0
Boot PROM Version        : Build 3.00.501
Firmware Version         : Build 3.00.522
Hardware Version         : B1
Firmware Type            : RI
Serial Number            : PVT93CB000001
System Name              :
System Location          :
System Uptime            : 0 days, 0 hours, 18 minutes, 46 seconds
System Contact           :
Spanning Tree            : Disabled
GVRP                     : Disabled
IGMP Snooping            : Disabled
MLD Snooping             : Disabled
RIP                      : Disabled
RIPng                    : Disabled
```

Рис. 2.8 – відмінності в порядку виводу параметрів для різних серій

Створені шаблони текстового аналізатора TextFSM для обробки виводу команд текстового інтерфейсу D’Link опубліковані автором роботи за посиланням: https://github.com/Vionikk/D-Link_TextFSM_templates та у Додатку Г. Також слід вказати, що існує публічний репозиторій, який містить багато шаблонів TextFSM для основних виробників мережевого обладнання (https://github.com/networktocode/ntc-templates), зокрема в ньому наявні шаблони для обробки текстового виводу команд комутаторів Cisco, Juniper Networks, MikroTik, Extreme Networks, і, так як шаблони обробки виводу для комутаторів D’Link відсутні як в доступних ресурсах і даному репозиторії, проведену роботу можна вважати актуальною. Особливості текстового виводу інтерфейсу командного рядка мережевого обладнання необхідно враховувати при розробці шаблонів текстового аналізатора TextFSM для коректного отримання даних з різних моделей та версій мережевих пристроїв, що мають розбіжності в межах одного стандарту CLI.

2.3 Розробка прототипу системи автоматизованого конфігурування мережевого обладнання

Перевірка функціонування прототипу системи автоматизованого конфігурування мережевого обладнання проводиться на змодельованій мережі, схема організації якої вказана на (рис. 2.9). Switch1 та Switch2 являють собою комутатори

D’Link DGS 3120 та D’Link DGS 3028 відповідно, дані моделі комутаторів є найбільш поширені в IT-інфраструктурі університету. На пристроях налаштовані статичні IP-адреси, а на локальній машині запущений прототип системи автоматизованого конфігурування мережевого обладнання.

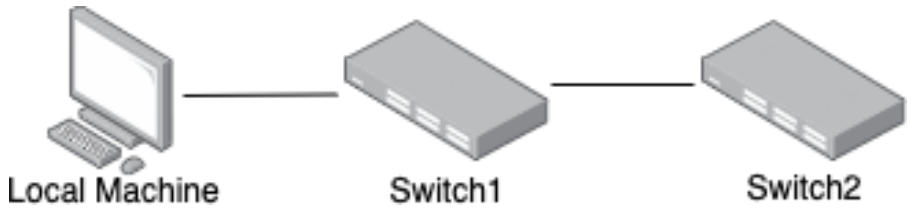


Рис. 2.9 – Схема організації дослідної мережі

У даній топології наявні Ethernet-коммутатори та локальна машина, на якій запущений програмний пакет та яка забезпечує підключення по SSH комутатора. Архітектура створеного програмного комплексу зображена на (рис. 2.10).

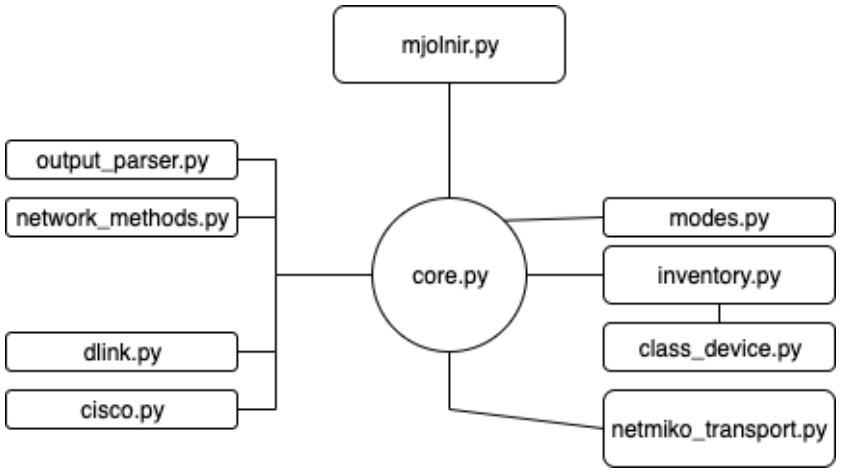


Рис. 2.10 – Архітектура розробленого прототипу

Прототип системи для автоматичного конфігурування мережевого обладнання створений мовою програмування Python, так як у розділах 1.6 та 1.7 було показано, що мова програмування Python найкраще відповідає обраним критеріям для розробки подібних систем та роботи з мережею, та базується на основі таких

бібліотеках як **Netmiko**, яка виконує задачі з підключення до пристрою через SSH або Telnet, посилання команд та отримання відповіді, та **TextFSM**, що виконує задачу обробки отриманої відповіді, а саме перетворення напівструктурованих даних у структуровані для подальшого використання їх в процесі конфігурування даного обладнання.

Розглянемо практичне застосування даного прототипу. Для початку необхідно задати початкові параметри. Дані про пристрої, а саме його ip-адреса, ім'я, тип та дані облікового запису містяться у так званому інвентарному файлі, він дозволяє також систематизувати пристрої в групи, з якими можна окремо працювати. Заданий інвентарний файл в форматі .yaml має таку структуру:

```
---

# Dictionaries are devices
device1:
  device_type: dlink_ds
  ip: 10.10.10.27
  username: admin
  password: password1

device2:
  device_type: dlink_ds
  ip: 10.10.10.28
  username: admin
  password: password1

# Lists are groups of devices
rig1:
  - device1

rig2:
  - device2

module:
  - device1
  - device2
```

Інформація про кожен окремий пристрій задається словником, що має свою унікальну назву. У полі "device_type" вказується тип пристрою, зі списку пристроїв та платформ, що підтримуються бібліотекою Netmiko, поля "ip", "username" та "password" містять відповідно IP-адресу пристрою та дані облікового запису, за допомогою якого здійснюється авторизація на пристрої. Нижче

вказуються групи пристроїв, кожна група містить набір пристроїв і в процесі роботи можна вказати назву окремих груп пристроїв, щоб проводити налаштування лише з ними.

Також необхідно задати функцію роботи програми, робиться це під час запуску програми за допомогою аргументу -m [mode].

Серед основних функцій наявні такі:

1. “ping” – перевіряє доступність пристроїв та можливість виконання команд конфігурування в оболонці пристрою
2. “show_inventory” – виводить вміст інвентарного файлу, пристрої та групи пристроїв
3. “execute_commands” – виконує на обладнанні заданий список команд або набір команд з текстового файлу
4. “dlink_show_vlans” – виводить таблицю наявних VLAN-ві для кожного комутатора, з яким відбувається робота
5. “dlink_create_vlan” – створює VLAN з заданими параметрами, опираючись на поточну конфігурацію
6. “dlink_add_port_to_vlan” – додає порт в вказаний VLAN, перевіряючи наявність такого.

Розглянемо алгоритм роботи програми для функції створення VLAN-у з заданими параметрами (рис. 2.11), команда виглядає наступним чином:

```
mjolnir.py -i inventory.yaml -g rig1 -m dlink_create_vlan -tag 7 -name testVlan
```

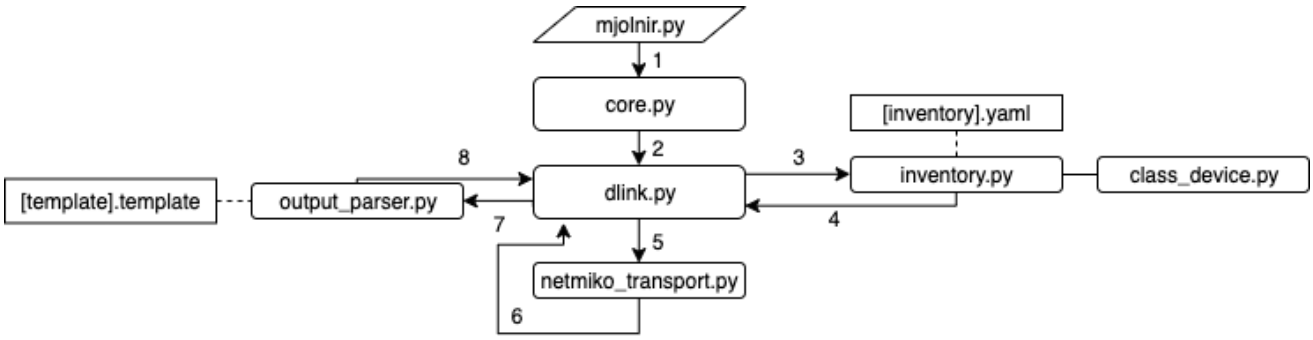


Рис. 2.11 – Алгоритм роботи методу

У команді задаю ться інвентарний файл, група або пристрій, функція та параметри для неї.Файл mjolnir.py приймає аргументи від користувача та передає файлу core.py, в свою чергу core.py перенаправляє отримані данні та параметри відповідному методу, а саме “dlink_create_vlan” який надсилає параметри інвентарного файлу у inventory.py щоб отримати список об’єктів класу даних Device з файлу class_device.py. Далі dlink.py надсилає дані про пристрій та команду “show vlan” в метод send_command(), що знаходиться в файлі netmiko_transport.py, та отримує текстову відповідь виконання надісланої команди на поточному пристрої. Після цього відповідь надсилається у метод parse_output() або parse_output_to_dict() де виконується обробка отриманих даних за допомогою TextFSM та відповідного шаблону для даної команди. Опрацьовані дані у вигляді словнику повертаються у dlink.py де зі словника отримується необхідні нам дані, в даному випадку це поля “VID” та “VLAN_Name”, що містить список список тегів та імен наявних VLAN-ів. Після цього в списку проводиться пошук, чи наявний VLAN з такими параметрами, якщо ні – посилається команда на створення VLAN-у з заданими параметрами та подальша обробка відповіді на наявність помилки. Якщо ж VLAN з хоч одним схожим параметром існує – користувач отримує повідомлення про це. Результат виконання програми наведений на (рис. 2.12).

Рис. 2.12 – Запуск та вивід програми

Даний прототип системи автоматизованого конфігурування мережевого обладнання може бути доповнений необхідними в роботі функціями, що використовують поточні параметри конфігурації пристрою, які можна отримати обробкою текстової відповіді за допомогою засобів бібліотеки TextFSM та відповідного шаблону для команди, яка сформувала даний вивід.

ВИСНОВКИ

Проведений в роботі аналіз встановив, що система автоматизованого конфігурування мережевим обладнаннм Ansible володіє кращими характеристиками згідно обраних критеріїв.

При аналізі застосування системи Ansible встановлено, що у базовій конфігурації Ansible відсутній модуль для управління частиною мережевого обладнання, що використовується в IT-інфраструктурі університету.

Виявлено, що в наявних системах автоматизованого конфігурування мережевого обладнання суттєвим недоліком є відсутність врахування поточних параметрів конфігурації мережевих пристроїв.

Проведений аналіз показав, що стандарт інтерфейсу командного рядка обладнання компанії D'Link має відмінності в формі відповіді на різних моделях обладнання.

Використання бібліотек Netmiko та TextFSM дозволяє створювати системи автоматизованого конфігурування обладнання окремих виробників та серій, що враховує поточні параметри конфігурації мережевих пристроїв.

В результаті розроблено шаблони, що використовуються для обробки відповіді інтерфейсу командного рядка комутаторів D'Link, та прототип системи автоматизованого конфігурування мережевого обладнання, що виконує конфігурування обладнання D'Link, враховуючи поточну конфігурацію мережевого пристрою.

ПЕРЕЛІК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. Документація Python 3.8 [Електронний ресурс] – Режим доступу до ресурсу: https://docs.python.org/3.8/ (дата звернення 05.04.2022)
2. Interface and Hardware Components Configuration Guide, Cisco IOS – [Електронний ресурс] – Режим доступу до ресурсу: https://www.cisco.com/c/en/us/td-/docs/switches/lan/catalyst9400/software/release/-16-9/configuration_guide/int_hw/b-169_int_and_hw_9400_cg.html (дата звернення 11.03.2022)
3. Ansible Documentation – [Електронний ресурс] – Режим доступу до ресурсу: https://docs.ansible.com/ansible/latest/index.html (дата звернення 11.03.2022)
4. Parsing command output with TextFSM [Електронний ресурс] – Режим доступу до ресурсу: https://pyneng.readthedocs.io/en/latest/book/21_textfsm/index.html (дата звернення 11.02.2022)
5. Документація інтерфейсу командного рядку комутатора D'Link DGS 3120 – [Електронний ресурс] – Режим доступу до ресурсу: URL: https://eu.dlink.com/-/media/business_products/dgs/dgs-3120/manual/dgs3120_series_r300_cli-reference-guide.pdf (дата звернення 17.05.2022)
6. Network Programmability and Automation: Skills for the Next-Generation Network Engineer 1st Edition. Jason Edelman.
7. Python for network engineers. Paramiko. – [Електронний ресурс] – Режим доступу до ресурсу: URL: https://pyneng.readthedocs.io/en/latest/book/18_ssh_telnet/-paramiko.html (дата звернення 29.05.2022)
8. Python for network engineers. Netmiko. – [Електронний ресурс] – Режим доступу до ресурсу: URL: https://pyneng.readthedocs.io/en/latest/book/18_ssh_telnet-/netmiko.html (дата звернення 28.05.2022)
9. Netmiko multi-vendor library official repository and documentation. – [Електронний ресурс] – Режим доступу до ресурсу: URL: https://github.com/ktbyers/netmiko (дата звернення 30.04.2022)
10. TextFSM library official repository and documentation. – [Електронний ресурс] – Режим доступу до ресурсу: URL: https://github.com/google/textfsm/wiki/TextFSM (дата звернення 02.05.2022)

ДОДАТКИ

ДОДАТОК А

```
# config file for ansible -- https://ansible.com/
# ===============================================

# nearly all parameters can be overridden in ansible-playbook
# or with command line flags. ansible will read ANSIBLE_CONFIG,
# ansible.cfg in the current working directory, .ansible.cfg in
# the home directory or /etc/ansible/ansible.cfg, whichever it
# finds first

[defaults]
inventory = hosts
host_key_checking = false
deprecation_warnings = False

# some basic default values...

#inventory      = /etc/ansible/hosts
#library        = /usr/share/my_modules/
#module_utils   = /usr/share/my_module_utils/
#remote_tmp     = ~/.ansible/tmp
#local_tmp      = ~/.ansible/tmp
#plugin_filters_cfg = /etc/ansible/plugin_filters.yml
#forks          = 5
#poll_interval  = 15
#sudo_user      = root
#ask_sudo_pass = True
#ask_pass      = True
#transport      = smart
#remote_port    = 22
#module_lang    = C
#module_set_locale = False

# plays will gather facts by default, which contain information about
# the remote system.
#
# smart - gather by default, but don't regather if already gathered
# implicit - gather by default, turn off with gather_facts: False
# explicit - do not gather by default, must say gather_facts: True
#gathering = implicit
```

```
# This only affects the gathering done by a play's gather_facts directive,
# by default gathering retrieves all facts subsets
# all - gather all subsets
# network - gather min and network facts
# hardware - gather hardware facts (longest facts to retrieve)
# virtual - gather min and virtual facts
# facter - import facts from facter
# ohai - import facts from ohai
# You can combine them using comma (ex: network,virtual)
# You can negate them using ! (ex: !hardware,!facter,!ohai)
# A minimal set of facts is always gathered.
#gather_subset = all

# some hardware related facts are collected
# with a maximum timeout of 10 seconds. This
# option lets you increase or decrease that
# timeout to something more suitable for the
# environment.
# gather_timeout = 10

# Ansible facts are available inside the ansible_facts.* dictionary
# namespace. This setting maintains the behaviour which was the default prior
# to 2.5, duplicating these variables into the main namespace, each with a
# prefix of 'ansible_'.
# This variable is set to True by default for backwards compatibility. It
# will be changed to a default of 'False' in a future release.
# ansible_facts.
# inject_facts_as_vars = True

# additional paths to search for roles in, colon separated
#roles_path    = /etc/ansible/roles

# uncomment this to disable SSH key host checking
#host_key_checking = False

# change the default callback, you can only have one 'stdout' type  enabled at a time.
#stdout_callback = skippy
```

```
## this is done to avoid running all of a type by default.
## These setting lists those that you want enabled for your system.
## Custom plugins should not need this unless plugin author specifies it.

# enable callback plugins, they can output to stdout but cannot be 'stdout' type.
#callback_whitelist = timer, mail

# Determine whether includes in tasks and handlers are "static" by
# default. As of 2.0, includes are dynamic by default. Setting these
# values to True will make includes behave more like they did in the
# 1.x versions.
#task_includes_static = False
#handler_includes_static = False

# Controls if a missing handler for a notification event is an error or a warning
#error_on_missing_handler = True

# change this for alternative sudo implementations
#sudo_exe = sudo

# What flags to pass to sudo
# WARNING: leaving out the defaults might create unexpected behaviours
#sudo_flags = -H -S -n

# SSH timeout
#timeout = 10

# default user to use for playbooks if user is not specified
# (/usr/bin/ansible will use current user as default)
#remote_user = root

# logging is off by default unless this path is defined
# if so defined, consider logrotate
#log_path = /var/log/ansible.log

# default module name for /usr/bin/ansible
#module_name = command

# use this shell for commands executed under sudo
```

```
# If set, configures the path to the Vault password file as an alternative to
# specifying --vault-password-file on the command line.
#vault_password_file = /path/to/vault_password_file

# format of string {{ ansible_managed }} available within Jinja2
# templates indicates to users editing templates files will be replaced.
# replacing {file}, {host} and {uid} and strftime codes with proper values.
#ansible_managed = Ansible managed: {file} modified on %Y-%m-%d %H:%M:%S by {uid} on {host}
# {file}, {host}, {uid}, and the timestamp can all interfere with idempotence
# in some situations so the default is a static string:
#ansible_managed = Ansible managed

# by default, ansible-playbook will display "Skipping [host]" if it determines a task
# should not be run on a host.  Set this to "False" if you don't want to see these "Skipping"
# messages. NOTE: the task header will still be shown regardless of whether or not the
# task is skipped.
#display_skipped_hosts = True

# by default, if a task in a playbook does not include a name: field then
# ansible-playbook will construct a header that includes the task's action but
# not the task's args.  This is a security feature because ansible cannot know
# if the *module* considers an argument to be no_log at the time that the
# header is printed.  If your environment doesn't have a problem securing
# stdout from ansible-playbook (or you have manually specified no_log in your
# playbook on all of the tasks where you have secret information) then you can
# safely set this to True to get more informative messages.
#display_args_to_stdout = False

# by default (as of 1.3), Ansible will raise errors when attempting to dereference
# Jinja2 variables that are not set in templates or action lines. Uncomment this line
# to revert the behavior to pre-1.3.
#error_on_undefined_vars = False

# by default (as of 1.6), Ansible may display warnings based on the configuration of the
# system running ansible itself. This may include warnings about 3rd party packages or
# other conditions that should be resolved if possible.
# to disable these warnings, set the following value to False:
#system_warnings = True
```

```
[inventory]
# enable inventory plugins, default: 'host_list', 'script', 'auto', 'yaml', 'ini', 'toml'
#enable_plugins = host_list, virtualbox, yaml, constructed

# ignore these extensions when parsing a directory as inventory source
#ignore_extensions = .pyc, .pyo, .swp, .bak, ~, .rpm, .md, .txt, ~, .orig, .ini, .cfg, .retry

# ignore files matching these patterns when parsing a directory as inventory source
#ignore_patterns=

# If 'true' unparsed inventory sources become fatal errors, they are warnings otherwise.
#unparsed_is_failed=False

[privilege_escalation]
#become=True
#become_method=sudo
#become_user=root
#become_ask_pass=False
```

```
[ssh_connection]

# ssh arguments to use
# Leaving off ControlPersist will result in poor performance, so use
# paramiko on older platforms rather than removing it, -C controls compression use
#ssh_args = -C -o ControlMaster=auto -o ControlPersist=60s

# The base directory for the ControlPath sockets.
# This is the "%(directory)s" in the control_path option
#
# Example:
# control_path_dir = /tmp/.ansible/cp
#control_path_dir = ~/.ansible/cp

# The path to use for the ControlPath sockets. This defaults to a hashed string of the hostname,
# port and username (empty string in the config). The hash mitigates a common problem users
# found with long hostnames and the conventional %(directory)s/ansible-ssh-%%h-%%p-%%r format.
# In those cases, a "too long for Unix domain socket" ssh error would occur.
#
# Example:
# control_path = %(directory)s/%%h-%%r
#control_path =

# Enabling pipelining reduces the number of SSH operations required to
# execute a module on the remote server. This can result in a significant
# performance improvement when enabled, however when using "sudo:" you must
# first disable 'requiretty' in /etc/sudoers
```

ДОДАТОК Б

Код скрипта, що здійснює підкючення та виконання команди show switch за допомогою бібліотеки paramiko:

```
import paramiko
import sys
import time
import pdb
#pdb.set_trace()

# setting parameters like host IP, username, passwd and number of iteration
# to gather cmds
HOST = "192.168.31.235"
USER = "iris"
PASS = "iris"
PORT = 22
ITERATION = 3

# A function that logins and execute commands
def fn():
    client1=paramiko.SSHClient()
    #add missing client key
    client1.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    #connect to switch
    client1.connect(HOST,username=USER,password=PASS,port=PORT)
    print("SSH connection to %s established: " + HOST)

    remote_conn = client1.invoke_shell()
    remote_conn.send("show switch")
    remote_conn.send("\n")
    remote_conn.send("n")
    remote_conn.send("\n")
    time.sleep(2)
    output = remote_conn.recv(10000)
    output = output.decode("utf-8")

    print(str(output))

#    stdin,stdout,stderr =client1.exec_command("config /n")
#    stdin,stdout,stderr =client1.exec_command("show /n")
#    print stdout.read()
    client1.close()
fn()
```

Вивід:

```
[iris@MiWiFi-R4AC-srv Desktop]$ python3 paramiko_example.py
SSH connection to %s established: 192.168.31.235


                    DGS-3120-24TC Gigabit Ethernet Switch
                            Command Line Interface

                          Firmware: Build 4.24.B001
          Copyright(C) 2018 D-Link Corporation. All rights reserved.

Ancible_test:admin#show switch
Command: show switch

Device Type                : DGS-3120-24TC Gigabit Ethernet Switch
MAC Address                : 54-B8-0A-CA-DA-40
IP Address                 : 192.168.31.235 (Manual)
VLAN Name                  : default
Subnet Mask                : 255.255.255.0
Default Gateway            : 0.0.0.0
Boot PROM Version          : Build 3.00.501
Firmware Version           : Build 4.24.B001
Hardware Version           : B1
Firmware Type              : RI
Serial Number              : R3V21F7000208
System Name                :
System Location            :
System Uptime              : 0 days, 0 hours, 35 minutes, 5 seconds
System Contact             :
Spanning Tree              : Disabled
GVRP                       : Disabled
IGMP Snooping              : Disabled
MLD Snooping               : Disabled
RIP                        : Disabled
RIPng                      : Disabled
DVMRP                      : Disabled
PIM                        : Disabled
PIM6                       : Disabled
OSPF                       : Disabled
OSPFv3                     : Disabled
VLAN Trunk                 : Disabled
Telnet                     : Enabled (TCP 23)
Web                        : Enabled (TCP 80)
SNMP                       : Disabled
SSL Status                 : Disabled
SSH Status                 : Enabled
802.1X                     : Disabled
Jumbo Frame                : Off
CLI Paging                 : Enabled
MAC Notification           : Disabled
Port Mirror                : Disabled
SNTP                       : Disabled
DHCP Relay                 : Disabled
VRRP                       : Disabled
HOL Prevention State       : Enabled
Syslog Global State        : Disabled
Single IP Management       : Disabled
Password Encryption Status : Enabled
DNS Resolver               : Disabled

Ancible_test:admin#
[iris@MiWiFi-R4AC-srv Desktop]$
```

ДОДАТОК В

Рекомендується перед виконанням плейбуків через Telnet модуль на комутаторах D’Link виконати команду: “disable clipaging”. Файл telnet.yml:

```
---
- hosts: switch_telnet
  gather_facts: false
  connection: local
  tasks:
  - name: telnet,login and execute command
    ignore_errors: true
    expect:
      command: telnet "{{ inventory_hostname }}"
      responses:
        (?i)username: "iris"
        (?i)password: "iris"
        (?i)#: "{{COMMAND}}\r\nlogout\r\nexit\r\nquit"
      echo: yes
    register: telnet_output


  - name: Debug output
    debug: var=telnet_output.stdout_lines
```

Запуск:

```
ansible-playbook raw_telnet.yml  -e '{"COMMAND":"show vlan"}'
```

Вивід:

```
TASK [Debug output] ************************************************************
ok: [192.168.31.235] => {
    "telnet_output.stdout_lines": [
        "Trying 192.168.31.235...",
        "",
        "Connected to 192.168.31.235.",
        "",
        "Escape character is '^]'.",
        "",
        "\u001b[0m\u001b[1;1H\u001b[2J",
        "",
        "                  DGS-3120-24TC Gigabit Ethernet Switch",
        "",
        "                         Command Line Interface",
        "",
        "",
        "                       Firmware: Build 4.24.B001",
        "",
        "          Copyright(C) 2018 D-Link Corporation. All rights reserved.",
        "",
        "UserName:iris",
        "",
        "PassWord:****",
        "",
        "",
        "Ancible_test:admin#show vlan",
        "",
        "Command: show vlan",
        "",
        "",
        "",
        "VLAN Trunk State\t: Disabled",
        ""
```

ДОДАТОК Г

Шаблони текстового аналізатора TextFSM для обробки текстової відповіді мережевого обладнання D’Link серій DES та DGS.

Шаблон обробки результату виконання команд конфігурування, вказує на позитивний результат виконання, наявність помилок та інших виключних ситуацій:

```
Value Result (Success.|Fail!|Next possible completions:|Ambiguous
token:|Available commands:|Invalid input detected at ^marker)

Start
  ^${Result} -> Record
```

Шаблон обробки результату виконання команди “*show switch*”:

```
Value  Device  (\S+)
Value Mac (\w+(-\w+){5})
Value IP (\d+(\.\d+){3})
Value VLAN (\S+)
Value Subnet (\d+(\.\d+){3})
Value Gateway (\d+(\.\d+){3})
Value Boot (Build \S+)
Value Firmware (Build \S+)
Value Hardware (\S+)
Value Name (\S+)
Value Location (\S+)
Value Telnet (\S+.+)
Value SSH (\S+.+)
Value Clipaging (\S+)

Start
  ^.*Device Type.*: ${Device}
  ^.*MAC Address.*: ${Mac}
  ^.*IP Address.*: ${IP}
  ^.*VLAN Name.*: ${VLAN}
  ^.*Subnet Mask.*: ${Subnet}
  ^.*Default Gateway.*: ${Gateway}
  ^.*Boot PROM Version.*: ${Boot}
  ^.*Firmware Version.*: ${Firmware}
  ^.*Hardware Version.*: ${Hardware}
  ^.*System Name.*: ${Name}
  ^.*System Location.*: ${Location}
```

```
  ^.*TELNET.*: ${Telnet}
  ^.*SSH.*: ${SSH}
  ^.*Clipaging.*: ${Clipaging}
  ^.*CLI Paging.*: ${Clipaging} -> Record
```

Шаблон обробки результату виконання команди “*show vlan*”:

```
Value VID (\S+)
Value VLAN_Name (\S+)
Value VLAN_Type (\S+)
Value Advertisement (\S+)
Value VLAN_Member_ports ( \S+|)
Value VLAN_Current_Tagged_Ports ( \S+|)
Value VLAN_Current_Untagged_Ports ( \S+|)
Value VLAN_Static_Tagged_Ports ( \S+|)
Value VLAN_Static_Untagged_Ports ( \S+|)

Start
  ^.*VID.*: ${VID} .*VLAN Name.* : ${VLAN_Name}
  ^.*VLAN Type.*: ${VLAN_Type} .*Advertisement.* : ${Advertisement}
  ^.*Member Ports.*:${VLAN_Member_ports}
  ^.*Current Tagged Ports.*:${VLAN_Current_Tagged_Ports}
  ^.*Current Untagged Ports.*:${VLAN_Current_Untagged_Ports}
  ^.*Static Tagged Ports.*:${VLAN_Static_Tagged_Ports}
  ^.*Static Untagged Ports.*:${VLAN_Static_Untagged_Ports} -> Record
```

Шаблон обробки результату виконання команди “*show ports*”:

```
Value Port (\S+)
Value Type (\S+| )
Value State (Enabled|Disabled)
Value Settings (\S+)
Value Connection_Speed (\S+|Link Down)
Value Address_Learning (Enabled|Disabled)

Start
 ^${Port} +${Type} +${State} +${Settings} +${Connection_Speed} +${Ad-
dress_Learning} -> Record
```

Шаблон обробки результату виконання команди “*show ports description*”:

```
Value Port (\S+)
Value Description (\S.*?)
```

```
Start
 ^${Port}
 ^.*Description:(?:\s+${Description})?\s*$$ -> Record
```

Шаблон обробки результату виконання команди “*show arpentry*”:

```
Value INTERFACE (\S+)
Value IP (\d+(\.\d+){3})
Value MAC (.+)
Value TYPE (\S+)

Start
 ^${INTERFACE}.+ ${IP}.+ ${MAC}.+ ${TYPE} -> Record
```

Шаблон обробки результату виконання команди “*show fdb*”:

```
Value VID (\d+)
Value VLAN_Name (\S+)
Value MAC_Address (\S+)
Value Port (CPU|\d+)
Value Type (\S+)

Start
 ^${VID} +${VLAN_Name} +${MAC_Address} +${Port} +${Type} -> Record
```

Шаблон обробки результату виконання команди “ *show address_binding dhcp_snoop binding_entry*”:

```
Value IP_Address (\d+(\.\d+){3})
Value MAC_Address (.+)
Value Status (\S+)
Value Left_Time (\d+)
Value Port (\d+)

Start
 ^ ${IP_Address} +${MAC_Address} +${Status} +${Left_Time} +${Port} -> Rec-
ord
```

Шаблон обробки результату виконання команди “*show ipif*”:

```
Value IP_Interface (\S+)
Value VLAN_Name (\S+)
Value Interface_Admin_State (Enabled|Disabled)
Value DHCPv6_Client_State (Enabled|Disabled)
```

```
Value Link_Status (\S+)
Value IPv4_Address (\S+)
Value IPv4_State (Enabled|Disabled)
Value IPv6_State (Enabled|Disabled)
Value DHCP_Option12_State (Enabled|Disabled)
Value DHCP_Option12_Host_Name (\S+)

Start
  ^.*IP Interface.*: ${IP_Interface}
  ^.*VLAN Name.* : ${VLAN_Name}
  ^.*Interface Admin State.*: ${Interface_Admin_State}
  ^.*DHCPv6 Client State.* : ${DHCPv6_Client_State}
  ^.*Link Status.* : ${Link_Status}
  ^.*DHCPv6 Client State.* : ${DHCPv6_Client_State}
  ^.*Link Status.*: ${Link_Status}
  ^.*IPv4 Address.*: ${IPv4_Address}
  ^.*IPv4 State.*: ${IPv4_State}
  ^.*IPv6 State.*: ${IPv6_State}
  ^.*DHCP Option12 State.*: ${DHCP_Option12_State}
  ^.*DHCP Option12 Host Name.*: ${DHCP_Option12_Host_Name} -> Record
```

Шаблон обробки результату виконання команди “*show iproute*”:

```
Value IPAddress (\d+(\.\d+){3})
Value Netmask (\d+)
Value Gateway (\d+(\.\d+){3})
Value Interface (\S+|\d+)
Value Hops (\d+)
Value Protocol (\S+)

Start
 ^(${IPAddress})/(${Netmask}).+ ${Gateway}.+ ${Interface}.+ ${Hops}.+
${Protocol} -> Record
```

Шаблон обробки результату виконання команди “*cable_diag ports all*”:

```
Value Required Port (\d+)
Value Type (\S+)
Value STATUS (Link Up|Link Down|-)
Value List Results (\S+ \d \S+|\S+|\S+ \S+|-)

Start
  ^\s*$$
  ^\w+\s+[TtYyPpEe]{4}.*$$ -> RESULT
  # Capture time-stamp if vty line has command time-stamping turned on
```

```
  ^Load\s+for\s+
  ^Time\s+source\s+is

RESULT
  ^\d+ -> Continue.Record
  ^${Port}\s+${Type}\s+${STATUS}\s*$$
  ^${Port}\s+${Type}\s+${STATUS}\s+${Results},* -> Continue
  ^\s+${Results},* -> Continue

Done
  ^.*
```