Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ«КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

Система керування конфігураціями пристроїв в мережі підприємства.

Текстова частина до курсової роботи за спеціальністю 121 «Інженерія програмного забезпечення»

Керівник курсової роботи
Кандидат технічних наук
Черкасов Д.І.
_____
(підпис)
«____» _____ 2021 р.

Виконав студент БП ІПЗ-3
Щербина С.С.
«____» _____ 2021 р.

Київ 2021

Календарний план виконання роботи

№	Назва етапу курсового проекту (роботи)	Термін виконання	Примітка
1.	Отримання завдання на курсову роботу	02.11.2020	
2.	Огляд літератури за темою роботи	15.11.2020	
3.	Написання першої частини курсової роботи	14.03.2021	
4.	Виконання практичної частини застосунку	19.04.2021	
5.	Написання другої частини курсової роботи	10.05.2021	
6.	Завершення роботи над текстовою частиною	13.05.2021	
7.	Створення презентації	15.05.2021	
8.	Захист роботи	24.05.2021	

Студент *Щербина С.С*.

Викладач *Черкасов Д.І.*

“____” _____

Зміст

Вступ

Комунікаційна мережа – надзвичайно важлива складова більшості сучасних підприємств. Її ефективність необхідна підприємству аби функціонувати, а неефективність може призвести до серйозних негативних наслідків. Велика мережа є складним поєднанням багатьох елементів, яка потребує спеціальних засобів для налаштування, профілактики та експлуатації. Однин з цих засобів - система керування конфігураціями пристроїв.

Система керування конфігураціями пристроїв - це програмне забезпечення, яке надає адміністраторам можливість відслідковувати роботу мережі в будь-який момент часу; створювати нові та редагувати існуючі конфігурації пристроїв; проводити тести на коректність налаштувань пристроїв у мережі.

Існує багато причин, через які мережа може почати поводити себе некоректно, деякі з них:

- Вихід з ладу пам’яті пристрою.
- Вихід з ладу іншого пристрою, функціональність якого впливає на конфігурацію інших пристроїв.
- Помилки та збої програмного забезпечення
- Помилки адміністратора
- Зловмисницькі дії

Автоматизований збір та керування інформацією дозволяють швидше знаходити причини неправильної поведінки мережі та оперативно відновлювати працездатність.

Також є багато довгострокових переваг використання таких систем:

- Колективна робота адміністраторів з системою;
- Розподіл обов`язків по управлінню мережею;
- Ретроспективне збереження множини конфігурацій кожного пристрою;

- Можливість коригування конфігурацій;
- Аналіз збережених конфігурацій;

Основною метою роботи є дослідження різних методів моніторингу, отримання та збереження конфігурацій пристроїв в мережі. Серед задач, які ставить дана робота:

- Розглянути системи керування конфігураціями як один із засобів автоматизації роботи мережі;
- Провести порівняльний аналіз існуючих рішень на ринку;
- Провести структурну розробку власного рішення, з детальним описом архітектури, компонентів системи, їхніх характеристик.
- Розробити базову версію клієнт-серверного застосунку системи керування пристроями в мережі підприємства.

Розділ 1. Огляд існуючих рішень

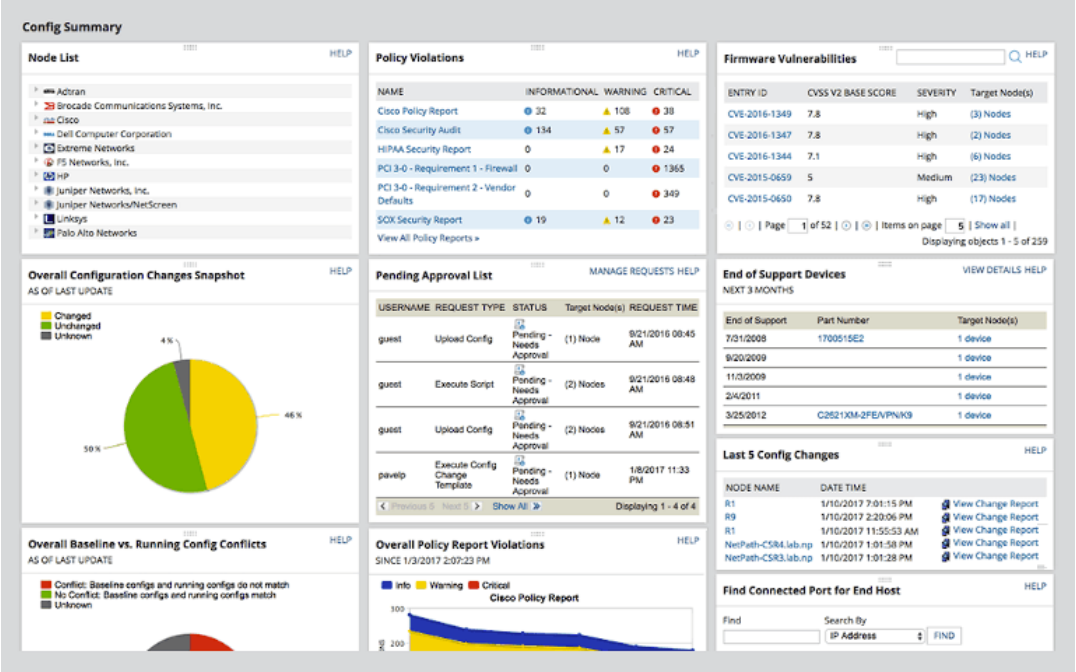
1.1 Порівняльний аналіз існуючих систем управління конфігураціями

У міру того, як мережі продовжують розвиватися та збільшуватися в розмірах, конфігурації стають все більш складними, та управління ними за допомогою звичайних методів стає незручним. Тому на ринку існує велика кількість інструментів для виконання цієї задачі.

В той час, як деякі інструменти надають лише основний функціонал, інші мають додаткові можливості, такі як звітування, сповіщення, можливість забезпечити безпеку, аудит та підтримку приладів від різних постачальників тощо.

Потужності кожної функції можуть відрізнятися в залежності від інструмента. Подібним чином вартість та масштабованість можуть відповідно змінюватися, і найкращою практикою є вибір інструменту, який відповідає поточним та майбутнім вимогам.

SolarWinds NCM



SolarWinds (Рис. 1)

SolarWinds® Network Configuration Manager (NCM) пропонує повне резервне копіювання і відновлення конфігурацій, а також підтримку різних постачальників пристроїв.

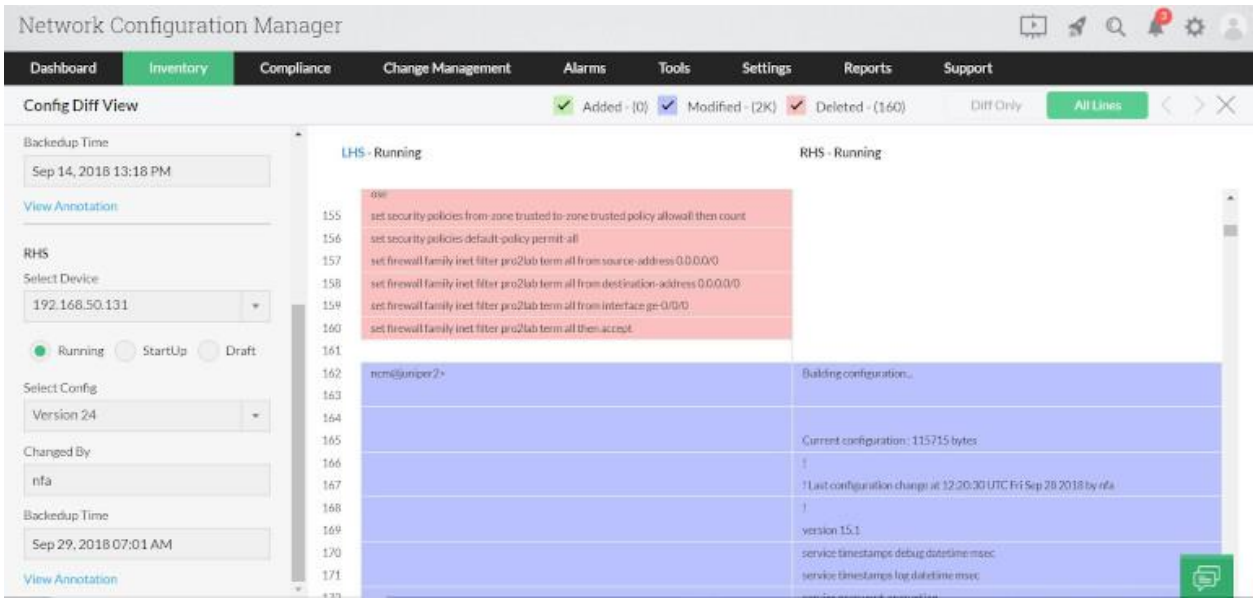
Завдяки розширеним функціям автоматизації мережі легко керувати складними змінами в мережах і скорочувати час, необхідний для виконання завдань, що повторюються.

NCM допомагає виявляти несанкціоновані зміни, перевіряти конфігурації і усувати порушення. У разі непередбачеих змін можна відновити останню версію конфігурації. Може бути інтегрований з Network Performance Monitor для виявлення і виправлення помилок конфігурації та аналізу продуктивності пристрою. Допомагає виявляти пристрої у мережі, контролювати конфігурацію обладнання та програмного забезпечення і відправляти своєчасні попередження, коли пристрої досягають кінця обслуговування.

NCM пропонує комплексне і централізоване подання всієї мережі і дозволяє адміністраторам переглядати відомості про пристрій, контролювати несанкціонований доступ і визначати мережеві зміни. Він також включає надійні можливості звітності для інформування зацікавлених сторін і інвесторів про зміни конфігурації, інвентаризації мережі, безпеки, планування та дотриманні корпоративних вимог.

Повнофункціональну безкоштовну пробну версію NCM можна отримати на 30 днів, що дозволяє перевірити, чи влаштовує це рішення.

ManageEngine NCM



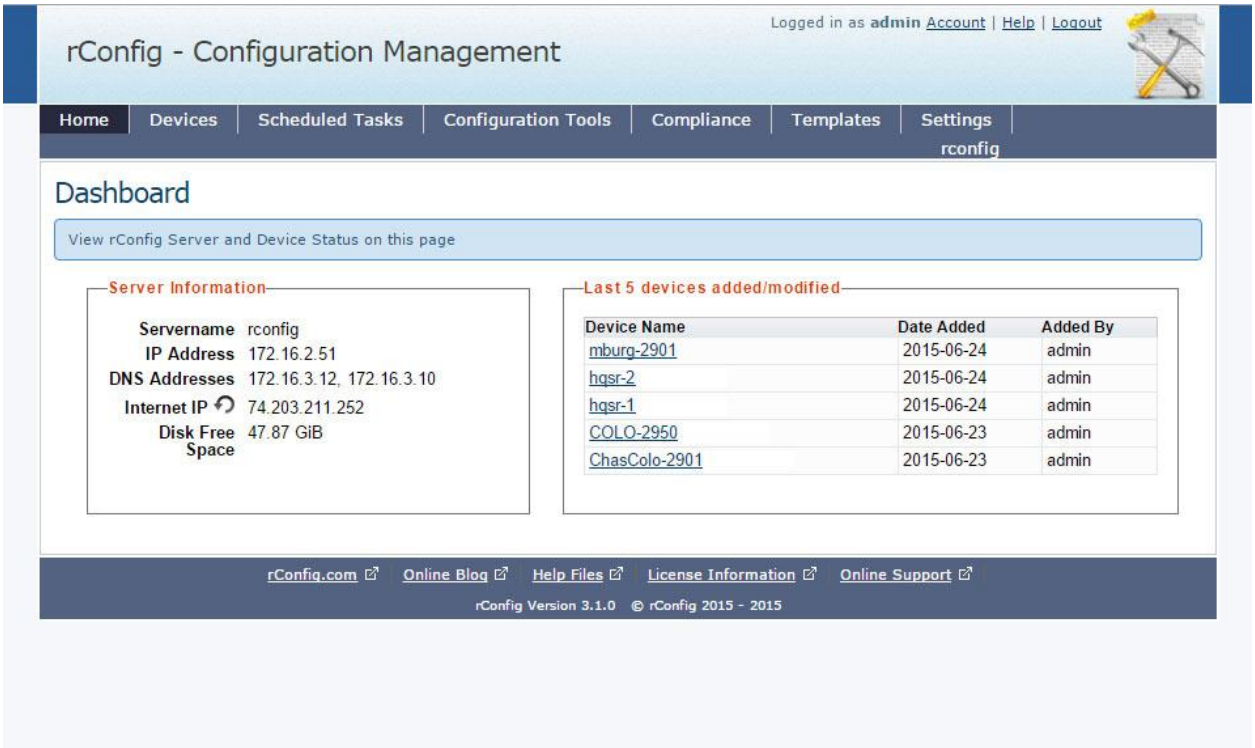
ManageEngine (Рис. 2)

ManageEngine Network Configuration Manager пропонує як базові, так і розширені функції для управління конфігураціями в мережі, а саме: управління конфігураціями, автоматичне резервне копіювання конфігурації, управління відповідністю пристроїв, підвищену безпеку мережі і багато іншого.

Цей інструмент також допомагає створювати детальні звіти про зміни, цілісність мережі і інвентаризації. Можливості моніторингу та оцінки цього інструменту допомагають відстежувати поточні зміни, визначати майбутні зміни і уникати несподіваних та небажаних змін.

Можна отримати безкоштовну пробну версію інструменту протягом 30 днів і перейти на платну версію, якщо результати будуть задовільними.

rConfig v3



rConfig v3 (Рис. 3)

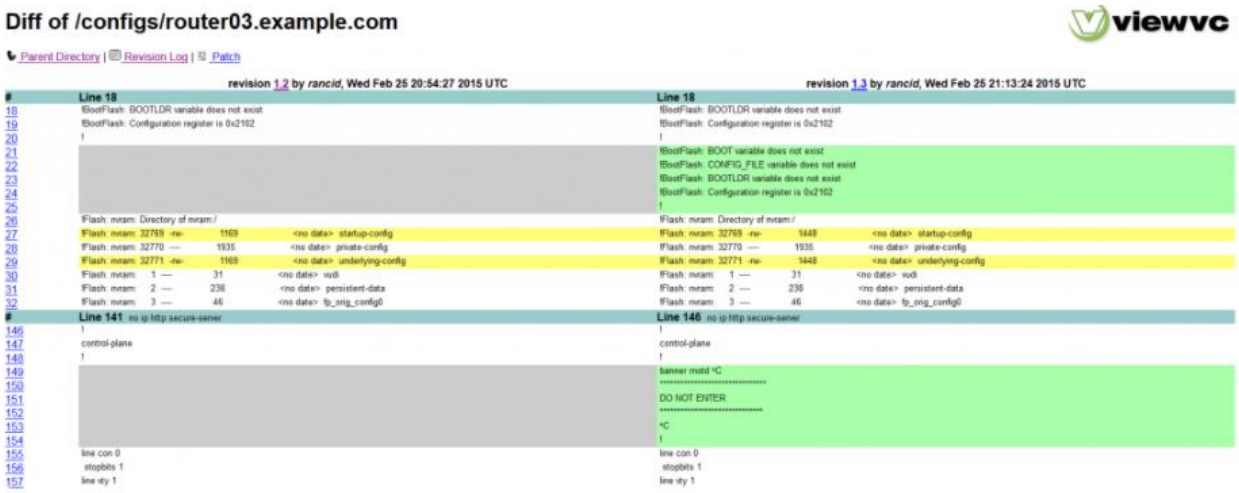
rConfig - один з найкращих безкоштовних інструментів управління конфігурацією мережі. Одна з його основних особливостей - сумісність з декількома операційними системами, в першу чергу з CentOS і Red Had Enterprise Linux. rConfig може виявляти, а також копіювати всі конфігурації з пристроїв у файли. Він може планувати встановлення конфігурації на пристрої. Можливо вносити зміни і виконувати дії з будь-якого пристрою в мережі.

Також можна встановити корпоративні політики, інструмент має вбудований диспетчер відповідності, який покаже вам, чи всі конфігурації відповідають політикам.

Найбільшою перевагою rConfig третьої версії є те, що інструмент є безкоштовним та з відкритим кодом. Це означає, що ви можете перевіряти і змінювати код в ньому на свій розсуд.

Найбільшим недоліком цього інструменту є відсутність утиліт аутентифікації. Це робить його майже непридатним для великих мереж.

RANCID



RANCID (Рис. 4)

Назва розшифровується як “Really Awesome New Cisco Config Differ.” Це безкоштовний інструмент, який розробляється вже багато років.

Хоча він не використовує веб-інтерфейси і не має графічного інтерфейсу, існує безліч посібників, які допоможуть розібратися в інструменті.

Це рішення надзвичайно універсальне і включає в себе базовий аудит, резервне копіювання та журнал дій. Незалежно від ваших постачальників, можливо ефективно налаштовувати та відстежувати конфігурації.

Порівняльна таблиця 1

	SolarWinds	ManageEngine	rConfig v3	RANCID
GUI	Детальний	Дуже детальний	Базовий	Відсутній
Система перевірки коректності конфігурацій	Network Performance Monitor	-	-	-
Резервні копії конфігурацій	+	+	+	+
Контроль доступу за ролями	+	-	-	-
Аудит та корпоративні вимоги	+	+	+	-
Мережеве відкриття	+	+	-	-
Сповіщення	Email, SMS, сценарії та додаткові(скрипти та програми)	Email, SMS та додаткові(скрипти та програми)	E-mail та додаткові(скрипти та програми)	Сторонні скрипти та програми
Журнал дій	-	+	-	+
Звітність	Детальна	Детальна	Достатня	Мінімальна
Ціна за 100 пристроїв	$2995 / рік	$4602 / рік	Open-source	Open-source

Таблиця 1.

Висновок до підрозділу 1.1

Для невеликих мереж доречно використати безкоштовне open-source рішення з базовим функціоналом та простим налаштуванням. Також обравши цей варіант, завжди можливо модифікувати інструмент скриптами та сторонніми програмами за бажанням адміністратора.

У великих комерційних мережах зазвичай вже не обійтися базовим функціоналом і краще обрати щось з безлічі платних рішень, що пропонують велику кількість інструментів, оновлення власного ПО та підтримку.

1.2 Порівняльний аналіз існуючих рішень для створення власної системи управління конфігураціями

Для побудови власної NCM можна обрати одну з цих популярних систем:

- Ansible
- Chef
- Puppet
- SaltStack

Архітектура

Ansible традиційно мав agentless архітектуру: існує master сервер, але відсутні клієнти на всіх agent приладах. У цьому підході доступ до кінцевих пристроїв отримується за допомогою SSH-з'єднання. Наразі підтримку було розширено і до master-agent архітектури. Інструмент є фаворитом у простоті налаштування, бо можливо обрати фундаментально різні реалізації архітектури.

Chef та SaltStack використовують (multi)master-agent архітектуру. Puppet використовує (multi)master-agent або stand-alone архітектури. За використання stand-alone архітектури кожний пристрій має повну копію налаштувань усіх пристроїв і періодично, або on-deman, виконує Puppet apply застосунок, що звертається до сервера з даними і за потреби робить зміни у своїй конфігурації.

Мова конфігурації

Chef використовує Ruby Domain Specific Language (Ruby DSL). Досить непроста мова для вивчення та орієнтована більше на розробників.

Puppet використовує власну мову Domain Specific Language (Puppet DSL). Також не дуже проста для вивчення, але орієнтована вже більше на адміністраторів.

Ansible та Saltstack використовують мову YAML(Python). Проста для розуміння і орієнтована на адміністраторів. Оскільки зазвичай Python вбудований у Unix та Linux дистрбутиви, процес налаштування буде швидшим.

Відмовостійкість

Усі рішення відмовостійкі. Це означає, що присутні декілька серверів або екземплярів master.

Ansible Tower працює з одним активним вузлом, який називається Primary instance(первинним екземпляром), і будь-якою кількістю неактивних вузлів, що називається Secondary instances(вторинними екземплярами). Secondary instance може стати Primary instance в будь-який час за визначеної ситуації.

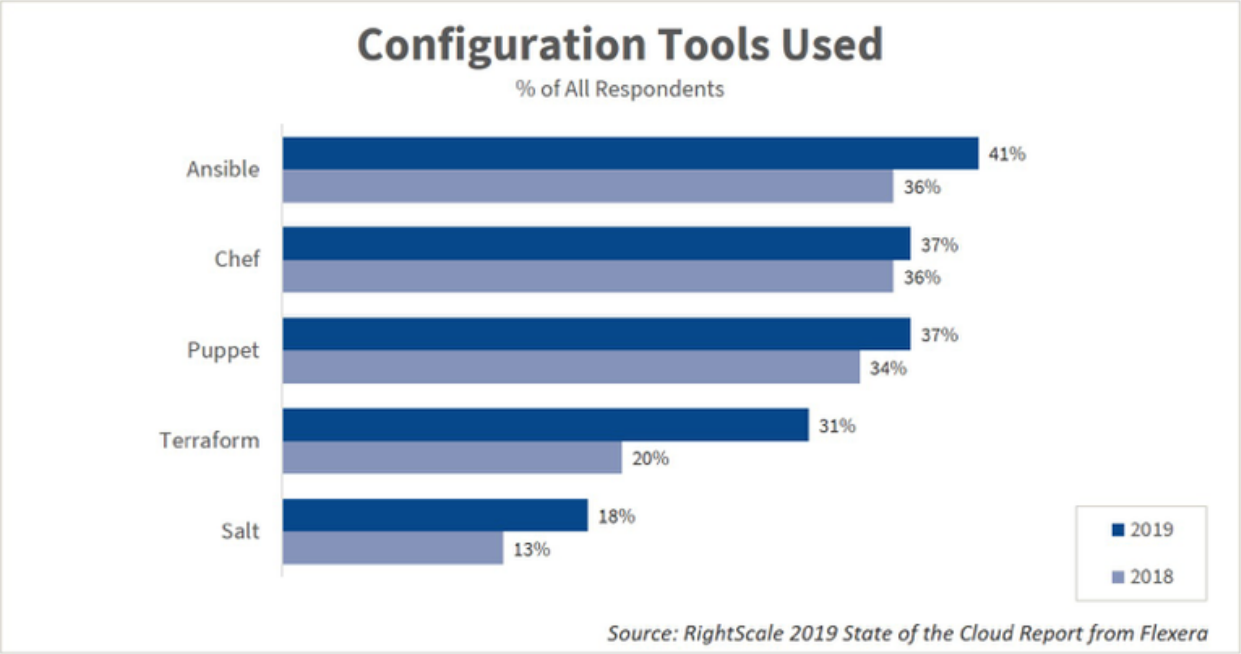
Chef має 2 групи серверів. Back-end кластер серверів (3), що забезпечують цілісність даних. Front-end група серверів (1 або більше), що отримують API запити. Навантаження має бути збалансованим, тому сервера можна горизонтально масштабувати, додаючи вузли.

Puppet має декілька серверів з базами даних та multi-master архітектуру. Можливо мати багато master серверів одночасно, але рекомендується використовувати їх як Primary та Secondary. Якщо Primary master недоступний, один з Secondary masters переймає його роль.

Saltstack також має multi-master архітектуру з різними варіантами налаштування. Також minions(agents) можуть тримати з'єднання одночасно з багатьма master.

Популярність

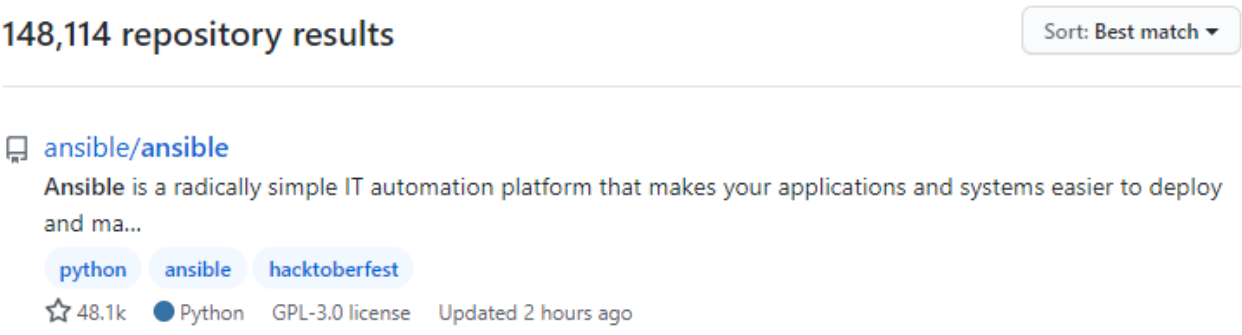
Розглянемо опитування у якому приняли участь 786 респондентів, 58% з яких працюють на підприємствах із понад 1000 працівників та пошукові запити в гітхабі.



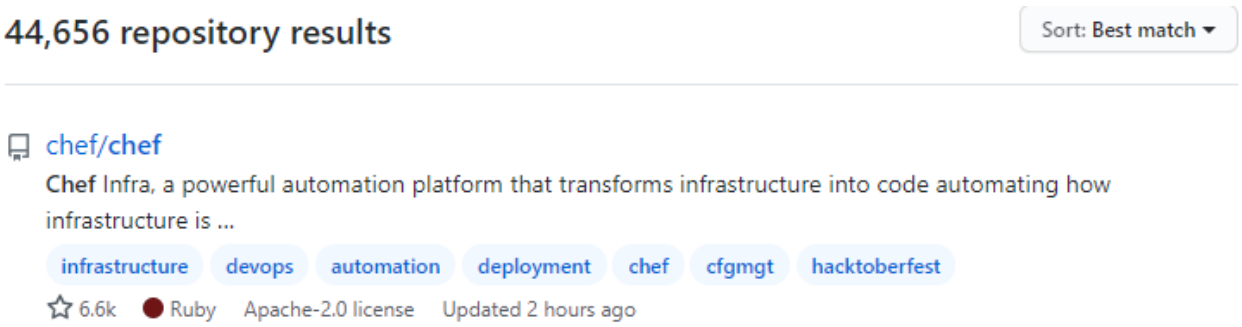
Опитування RightScale 2019 State of the Cloud (Рис. 5)

Як видно з графіку, Ansible займає майже половину ринку, і його популярність росте з кожним роком.

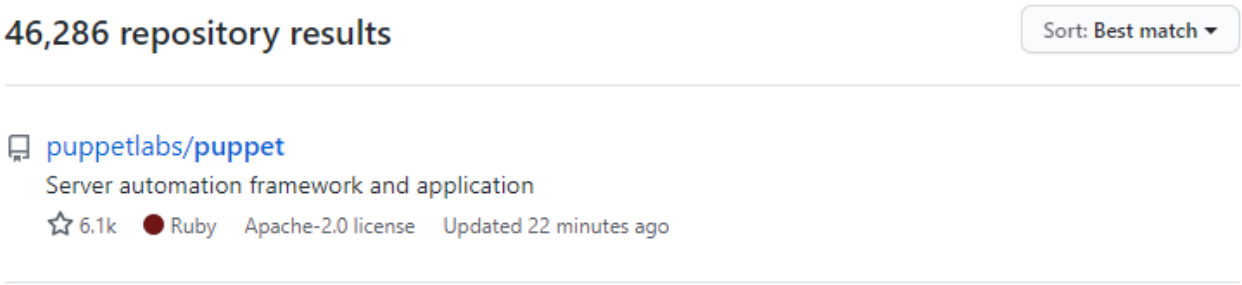
А ось на gihub Ansible набагато популярніший за конкурентів.



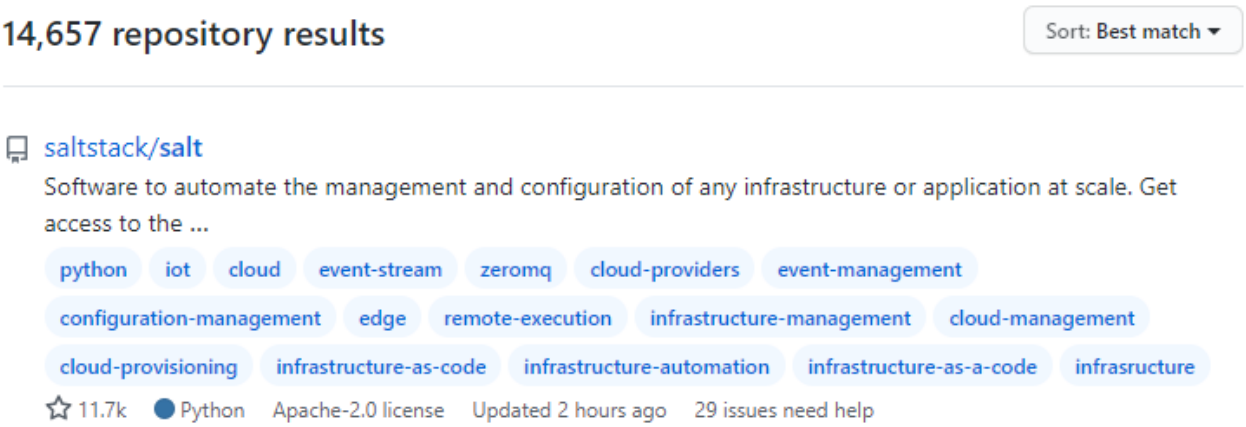
Кількість репозитроіїв з ім’ям “ansible” (Рис. 6)



Кількість репозитроіїв з ім'ям "chef" (Рис. 7)



Кількість репозитроіїв з ім'ям "puppet" (Рис. 8)



Кількість репозитроіїв з ім'ям "salt" (Рис. 9)

Таким чином, можна зробити висновок, що Ansible є найпопулярнішим рішенням і його популярність буде тільки зростати. А це означає збільшення кількості інформації про інструмент та його використання.

Порівняльна таблиця 2

	Ansible	Chef	Puppet	SaltStack
GUI	Ansbile Tower / AWX	Opscode Manage	Puppet Enterprise	SaltStack Enterprise
Відмовостійкість	Середня	Висока	Висока	Висока
Масштабованість	Висока	Висока	Висока	Висока
Складність налаштування	Низька	Середня	Середня	Середня
Популярність	Велика	Середня	Середня	Низька
Мова конфігурації	YAML (Python)	DSL (Ruby)	DSL (PuppetDSL)	YAML (Python)
Архітектура	master-agent, agentless	multimaster-agent	multimaster-agent	multimaster-agent, stand-alone

Таблиця 2.

1.3 Оркестрація та автоматизація

Термін автоматизація мережі застосовується до автоматизації мережевих процесів, таких як керування змінами, керування конфігураціями, виявлення та інвентаризація пристроїв тощо. Коли ці процеси виконуються системним адміністратором, то забирають багато часу і вразливі до помилок. Їхня автоматизація допомагає підприємствам економити час, зменшує кількість помилок, зменшує експлуатаційні витрати та покращує ефективність мережі.

Приклади інструментів мережевої автоматизації: Ansible, Puppet, Chief, Rancid тощо.

Мережева оркестрація стосується автоматизації взаємодії між різними типами пристроїв, доменів і потенційно інших систем у мережі. Оркестрація, як правило, вимагає можливості взаємодії з багатьма типами пристроїв від різних постачальників, а також у кількох доменах та системах керування, що вимагають програмних інтерфейсів, таких як REST API.

Оркестрація мережі - це наступний етап автоматизації мережі - її можна охарактеризувати як автоматизацію на основі політик чи подій. В той час як автоматизація передбачає "виконання одного конкретного завдання без участі людини", оркестрування йде набагато далі за рахунок автоматизації цілих "процесів", тобто "послідовності взаємопов'язаних завдань". Механізм оркестрації визначається набором правил або політик, встановлених організацією.

Приклади інструментів мережевої оркестрації: Terraform, CloudFormation, Ansible.

Порівняльна таблиця 3

Автоматизація мережі	Оркестрація мережі
Використовується для виконання окремих завдань низького рівня без втручання людини.	Використовується для запуску безлічі послідовних завдань або процесів високого рівня без втручання людини.
Зазвичай це робиться через CLI пристрою або через сторонні фреймворки, керовані сценаріями.	Робиться за допомогою програмних REST API, які дозволяють автоматизувати роботу пристроїв та платформ управління.
Обмежена для певного типу пристрою або постачальника.	Охоплює кілька мережевих служб, постачальників та середовищ.
Приклад: передача змін конфігурації у пристрій.	Приклад: надання мережевих послуг.

Таблиця 3.

Отже, оркестрація мережі існує на рівень вище за автоматизацію і включає її в себе. Вище згадані інструменти добре доповнюють один одного у правильно спроектованих мережах. Іноді рішення на ринку поєднують і оркестрацію, і автоматизацію – Ansible. Але першочергово це все таки рішення для автоматизації.

Зазвичай, засоби оркестрації використовуються у великих хмарних системах, а для мережі підприємства вистачить і системи керування конфігураціями пристроїв.

Висновок до розділу 1

Враховуючи мету використання, велику кількість інформації про інструмент, простоту налаштування та використання, можливість використання agenless архітектури та інші характеристики, було обрано Ansible та Ansible AWX(веб-інтерфейс, авторизація та планувальник) для створення власної системи.

Розділ 2. Структурна розробка власного рішення

Задача: розробка власної системи за допомоги інтеграції системи автоматизації, планувальника роботи, сповіщень, веб-інтерфейсу та власного додатку з веб-інтерфейсом для редагування, перевірки та огляду історії змін конфігурацій.

Схема складових компонентів системи

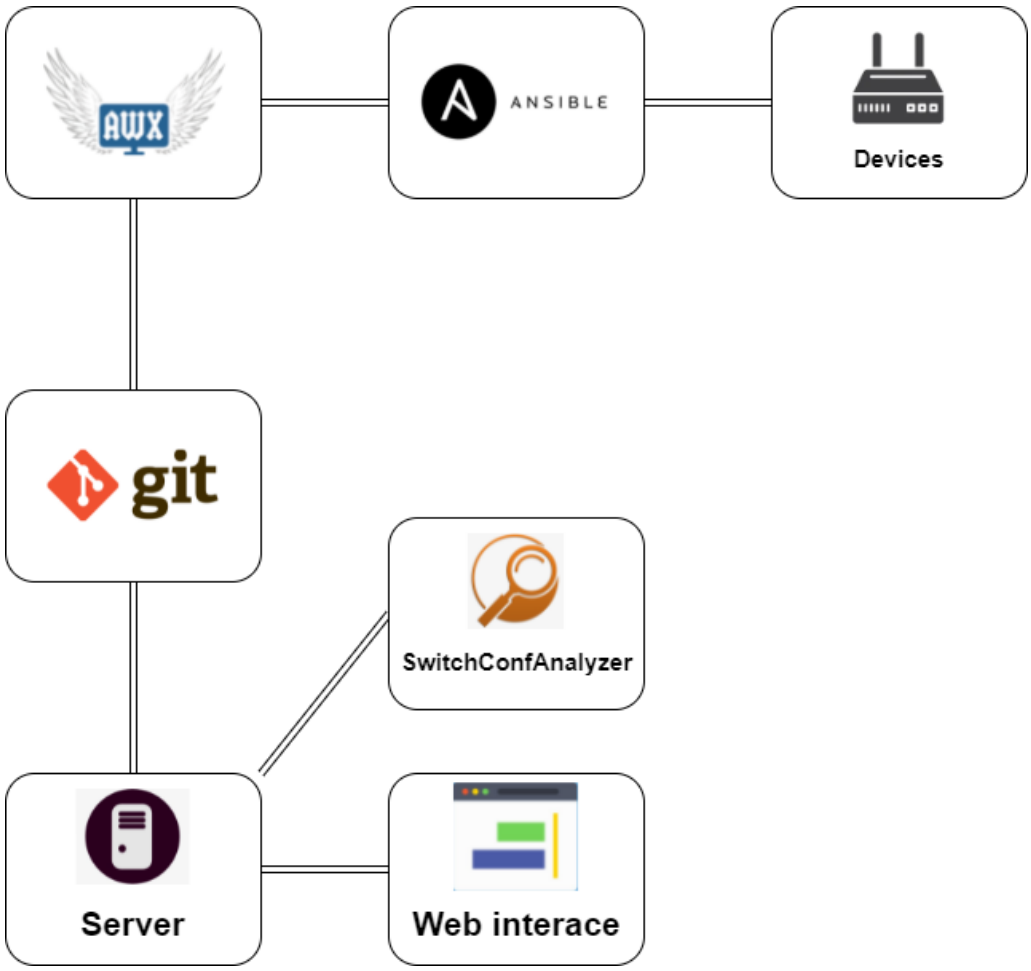


Схема складових компонентів системи(Рис. 10)

2.1 База даних на основі системи контролю версій Git

Однією з найпопулярніших та найпоширеніших систем котролю версіями є Git. Проект має відкритий код та добре підтримується. Всі розробники використовують системи контролю версій і Git можна побачити найчастіше.

Великий плюс Git – система має розподілену архітектуру. Існує основний репозиторій і безліч локальних, що тримають всю історію змін. Дуже популярний у 2010-х роках SVN(Subversion) наразі зменшує свою значимість. На відміну від Git, має централізовану архітектуру. У цьому випадку сервер, на якому знаходиться репозиторій, є єдиною точкою відмови.

Git - адресована файлова система, тобто сховище, де бажаний файл отримується за ключем. Тому для нескладних моделей Git є хорошим кандидатом для використання у якості надійної NoSQL бази даних зі збереженням історії змін.

2.2 Засіб аналізу конфігурацій SwitchConfAnalyzer

SwitchConfAnalyzer - скрипт на мові Python для аналізу конфігурацій multilayer комутаторів на основі IOS. У файлі шаблону можна вказати кілька елементів, яким повинна відповідати конфігурація. Можна вказати такі категорії:

- Інтерфейси доступу до комутатора
- Магістральні інтерфейси комутатора
- IP-інтерфейси
- Загальні елементи конфігурації
- Ієрархічні елементи конфігурації

2.3 Платформа автоматизації Ansible

Ansible - це засіб автоматизації з відкритим кодом або платформа, що використовується для таких IT-завдань, як управління конфігурацією, розгортання додатків, внутрішньосервісна оркестрація та забезпечення.

У розробленому рішенні має agentless архітектуру і використовується як засіб автоматизації конфігурування мережних приладів. Доступ до кінцевих пристроїв отримується за допомогою SSH.

Компоненти:

- Одна Primary instance нода (з можливістю розгорнути запасні Secondary instances), яка керує приладами через SSH.
- Inventory – текстовий файл, що містить ip адреси приладів
- Playbook – файл, написаний мовою YAML, що містить завдання для виконання, виконується разом з визначеним inventory. Складається з наступних компонентів:

Приклад Ansible Playbook (Рис. 11)

Ansible module-і виконують task-и. Для створення play можна поєднати одне або декілька task-ів. Дві або більше play можна об’єднати, щоб створити *playbook*. Ansible playbook - це списки завдань, які автоматично виконуються для хостів.

Кожний module виконує певне завдання і містить метадані, що визначають, коли і де виконується завдання, а також який користувач його

виконує. Існує велика кількість як вбудованих модулів, так і сторонніх, що встановлюються додатково.

Хоча можливо використовувати вбудовані модулі для отримання доступу до приладів, у розробленому playbook використовується module NAPALM для зручного доступу до приладів.

2.4 Ansible AWX

AWX створений для роботи над проектом Ansible, вдосконалюючи і без того потужний механізм автоматизації. Є рішенням з відкритим кодом, що поширюється під ліцензією GNU GPL.

Основною проблемою для користувачів Ansible було те, що платформа не мала хорошого графічного інтерфейсу. Навіть більше - графічний інтерфейс був настільки поганим, що на початку він навіть не був належним чином синхронізований з CLI, а це означає, що CLI та графічний інтерфейс могли дати 2 різні результати запитів про стан певного вузла.

AWX є вирішенням цієї проблеми. Це всеосяжний веб-інтерфейс для Ansible, що містить найважливіші функції Ansible, особливо ті, які краще відображаються як графічний, а не текстовий результат, такий як моніторинг вузлів у реальному часі.

Має простий у використанні інтерфейс, інформаційну панель та REST API для Ansible. Платформа забезпечує контроль доступу на основі ролей, планування завдань та графічне управління інвентарями. API REST та CLI дозволяють легко вбудовувати AWX в існуючі інструменти та процеси.

Особливості Ansible AWX:

- Рольовий контроль доступу: можливість створювати користувачів та об’єднувати їх у команди. Потім призначити доступ та правила до

інвентарю, облікових даних та ігрових книжок на індивідуальному рівні або на рівні команди.

- Планування роботи: об'єднавши облікові дані, playbook та inventory, створюється шаблон роботи. Шаблон є виконанням командного рядка «ansible-playbook», за винятком того, що це виконує сам інтерфейс. На додаток до запуску одного playbook, можливо створити шаблон роботи за допомогою редактора робочого циклу, який об'єднує запуск багатьох playbooks.
- Повністю задокументоване REST API: дозволяє інтегрувати Asible у існуючий набір інструментів та середовище
- Інформаційна панель: можливість швидкість і в реальному часі переглянути звіт про все оточення. Детальна інформація про виконання playbook-ів та хостів. На рівні playbook можливість переглянути інформацію про виконання конкретного task.
- Хмарна інтеграція: AWX сумісний з основними хмарними середовищами: Amazon EC2, Rackspace, Azure.

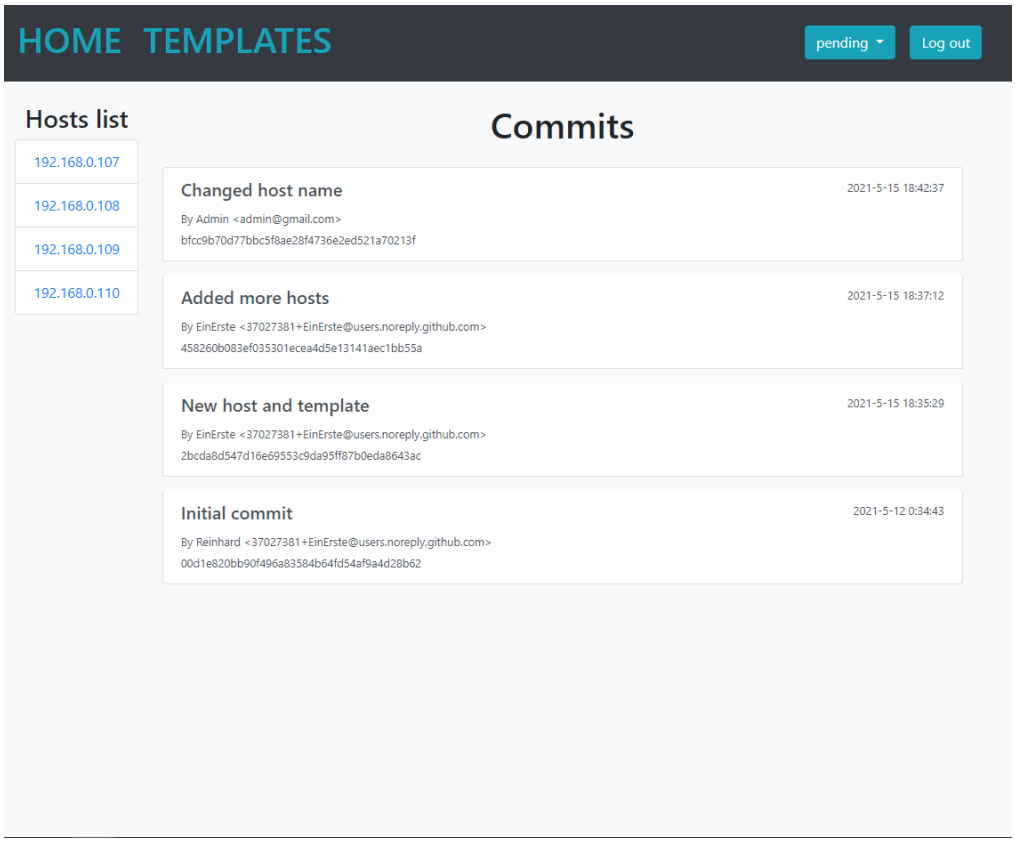
2.5 Розроблений додаток

Клієнт-серверне застосування, написане на мові JavaScript(NodeJS) з використанням фреймворку Express та бібліотеки node-git.

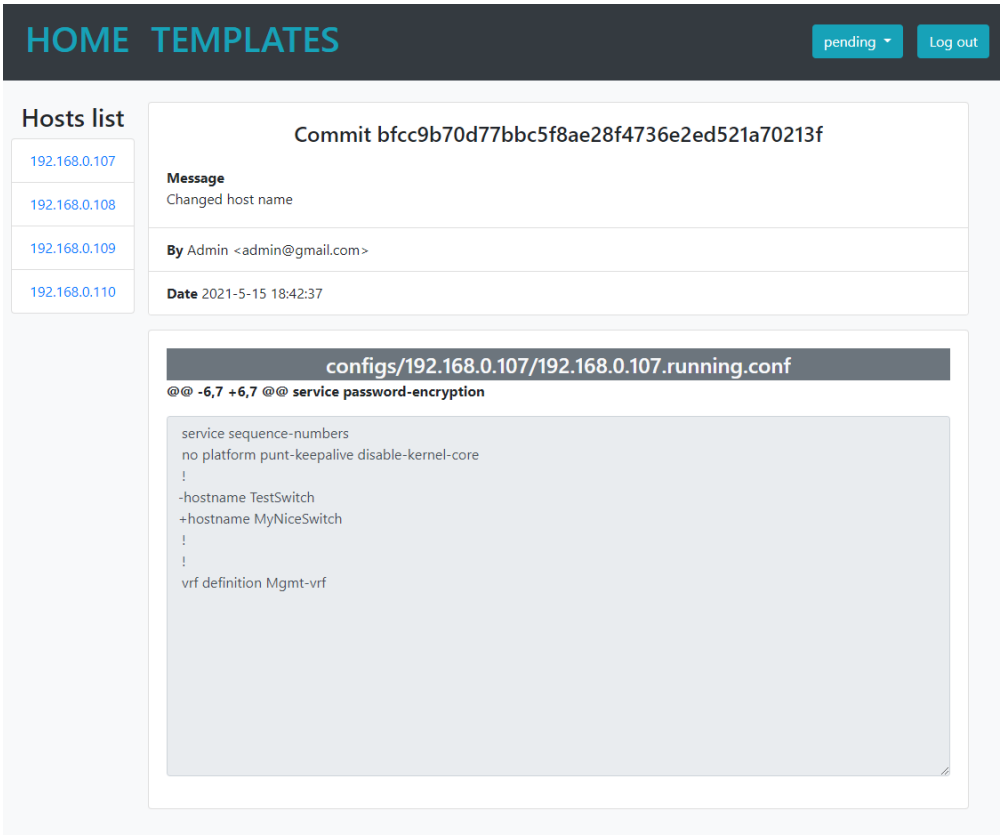
Основні функції:

- Інтеграція з платформою AWX
- Перегляд загальної історії змін
- Перегляд історії конфігурацій окремих пристроїв(хостів)
- Перегляд шаблонів для аналізу конфігурацій
- Створення нової конфігурації
- Можливість проаналізувати поточну або історичну конфігурації за допомогою вказаного шаблону

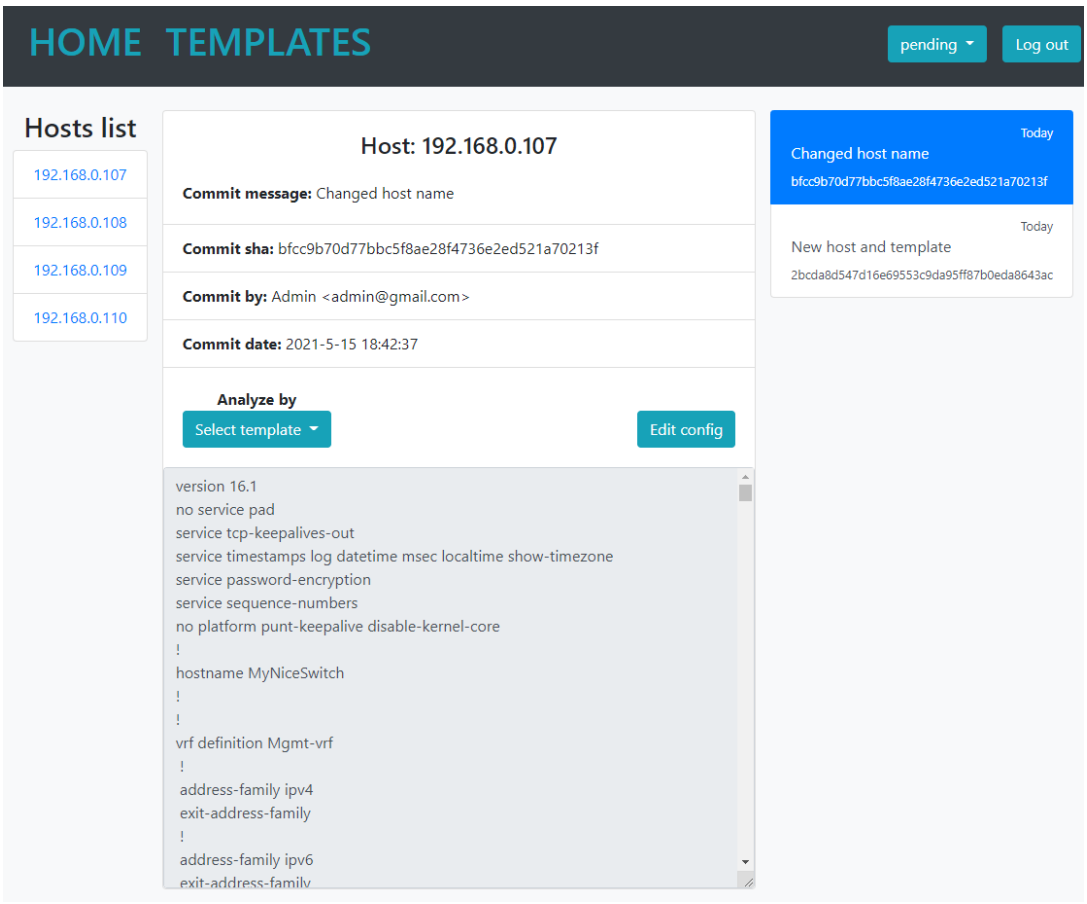
Веб-інтерфейс додатку



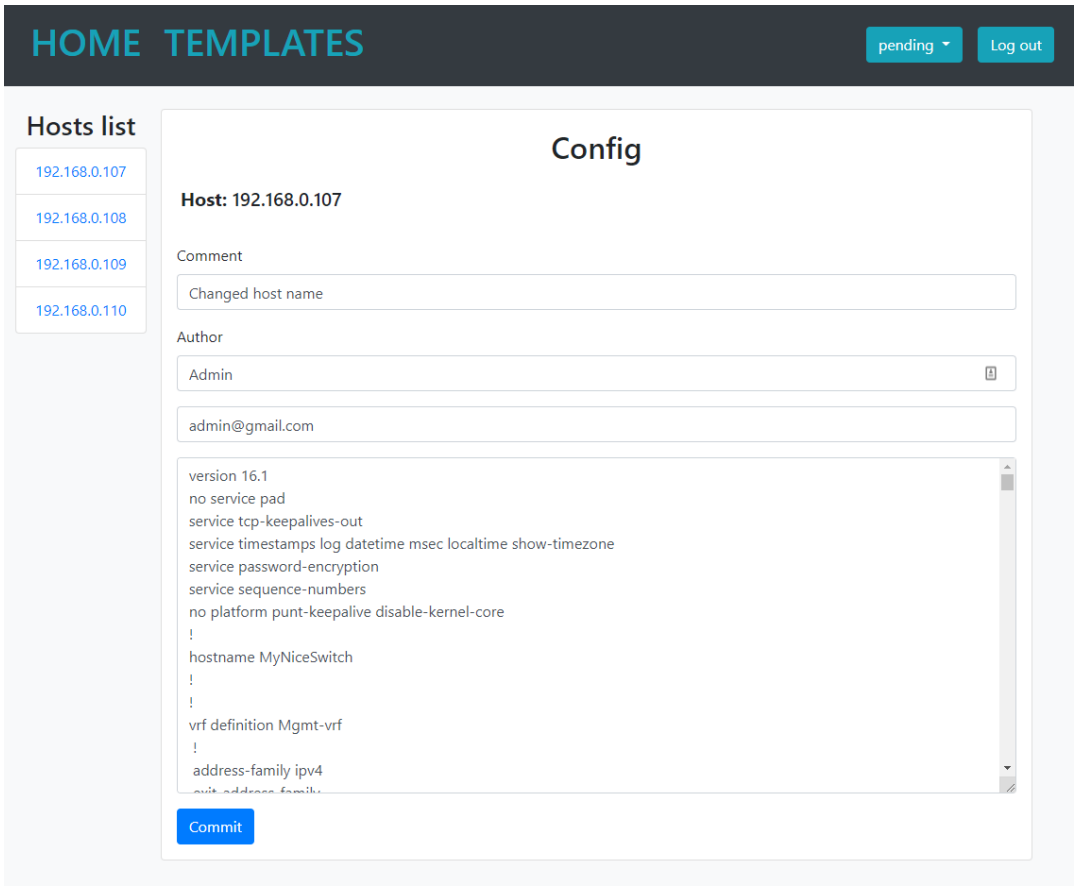
Головна сторінка (Рис. 12)



Перегляд інформації про зміни (Рис. 13)



Перегляд конфігурації (Рис. 14)



Створення нової конфігурації (Рис. 15)

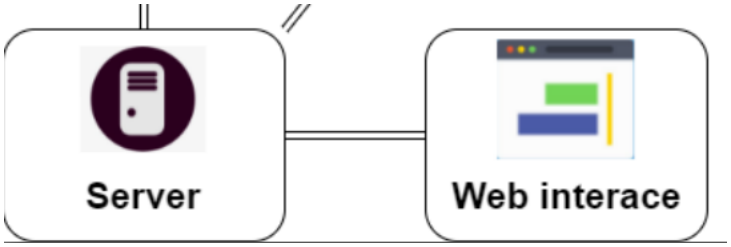
2.6 Опис функціонування системи

- Git використовується в якості NoSQL бази даних з історією змін.
- NAPALM модуль використовується в Ansible для отримання поточних конфігурацій та завантаження нових.
- Зміни до конфігурацій знаходяться у git гілці “pending”, що першопочатково є копією гілки “main”.
- Зміни до конфігурацій можливо робити напряму у git репозиторій, або використовуючи розроблений веб-інтерфейс, попередньо виконавши вхід за допомогою даних від AWX.
- Опціонально адміністратор може перевірити коректність конфігурацій, що реалізовано за допомогою модифікованого open-source Python скрипта SwitchConfAnalyzer.
- У AWX виконується playbook, що створює резервні копії. Потім інший playbook мерджить гілку “pending” у “main” та застосовує конфігурації на прилади.

Розділ 3. Детальна розробка компонентів системи

Було виконано розробку вказаних елементів

Playbook-и для Ansible(Рис 16)



Клієнт-серверний додаток з веб-інтерфейсом(Рис. 17)

Код

Створений код для ansible(Додаток А).

Пояснення до коду playbook-у, що створює запасні копії конфігурацій

Один раз на сервері виконується завдання на синхронізацію репозиторію та зміну гілки на гілку із запасними копіями конфігурацій

```
- name: Change branch to backup configs
  with_items:
    - pwd
    - git branch -r | grep -v '\->' | while read remote; do git branch --track "${remote#origin/}" "$remote"; done
    - git fetch --all
    - git pull --all
    - git checkout -b backup
  args:
    chdir: "{{ git_repo }}"
  when: git_sync == "enabled"
  delegate_to: 127.0.0.1
  run_once: True
```

Код backup.yml 1 (Рис. 18)

За допомогою модуля NAPALM збираються поточні конфігурації пристроїв

```yaml
- name: Get candidate(backup) config with NAPALM
  napalm_install_config:
    hostname: "{{ inventory_hostname }}"
    username: "{{ user }}"
    password: "{{ password }}"
    dev_os: "{{ os }}"
    provider: "{{ ios_provider }}"
    candidate_file: "{{ git_repo }}/configs/{{ inventory_hostname }}/{{ inventory_hostname }}.running.conf"
    commit_changes: False
```

Код backup.yml 2 (Рис. 19)

Коли зібрано усі конфігурації, гілка backup надсилається до віддаленого репозиторію.

```yaml
- name: Push backup branch
  with_items:
    - pwd
    - git fetch --all
    - git pull --all
    - git checkout backup
    - git add .
    - git commit -m "Updates {{ansible_date_time.date}}"
    - git push

  args:
    chdir: "{{ git_repo }}"
  when: git_sync == "enabled" and "{{ map('extract', hostvars)[-1] }}" == "{{ inventory_hostname }}"
  delegate_to: 127.0.0.1
```

Код backup.yml 3 (Рис. 20)

Пояснення до коду playbook-у, що надисилає конфігурації на пристрої

Гілки синхронізуються, потім гілка з вказаними(запасними або новими) конфігураціями з’єднується з гілкою поточних конфігурацій

```yaml
- name: Merge selected branch to main branch
  with_items:
    - pwd
    - git branch -r | grep -v '\->' | while read remote; do git branch --track "${remote#origin/}" "$remote"; done
    - git fetch --all
    - git pull --all
    - git checkout main
    - git merge {{ git_merge_branch }}
    - git push
  args:
    chdir: "{{ git_repo }}"
  delegate_to: 127.0.0.1
  run_once: True
```

Код push.yml 1 (Рис. 21)

Нові конфігурації надсилаються пристроям

```yaml
- name: Push new config with NAPALM
  napalm_install_config:
    hostname: "{{ inventory_hostname }}"
    username: "{{ user }}"
    password: "{{ password }}"
    dev_os: "{{ os }}"
    provider: "{{ ios_provider }}"
    config_file: "{{ git_repo }}/configs/{{ inventory_hostname }}/{{ inventory_hostname }}.running.conf"
    commit_changes: True
```

Код push.yml 1 (Рис. 22)

Висновок

Розроблено систему управління конфігураціями з наступними компонентами:

- база даних конфігурацій на основі системи контролю версій Git
- засіб перевірки коректності конфігурацій (SwitchConfAnalyzer)
- веб-інтерфейс для редагування та перегляду історії змін конфігурацій
- система автоматизації Ansible з модулем Napalm
- веб-інтерфейс адміністратора, планувальник роботи та система сповіщень(AWX)

Та функціямя:

- швидке відновлення конфігурації у випадку заміни пристроїв
- відновлення у разі пошкодження конфігурації внаслідок помилки
- історичний огляд змін в конфігураціях
- перевірка коректності конфігурацій
- повідомлення про помилки під час виконання

Виконано мету роботи:

- Розглянуто системи керування конфігураціями як один із засобів автоматизації роботи мережі;
- Проведено порівняльний аналіз існуючих рішень на ринку;
- Проведено структурну розробку власного рішення, з детальним описом архітектури, компонентів системи, їхніх характеристик.
- Розроблено базову версію клієнт-серверного застосунку системи керування пристроями в мережі підприємства.

Перелік джерел

1. Best Network Configuration Management Software & Tools[Електронний ресурс]:webservertalk.com – 2019. – Режим доступу: https://www.webservertalk.com/network-configuration-management-software
2. Хранители сети. Open source утилиты для управления, мониторинга и бэкапа настроек сетевого оборудования [Електронний ресурс]: xakep.ru – 2015. – Режим доступу: https://xakep.ru/2015/08/10/network-management/
3. Best Network Configuration Management Tools [Електронний ресурс]: tek-tools.com– 2020. – Режим доступу: https://www.tek-tools.com/network/best-network-configuration-management-tools
4. SolarWinds Network Configuration Manager [Електронний ресурс]: solarwinds.com – 2020. – Режим доступу: https://www.solarwinds.com/network-configuration-manager
5. ManageEngine Network Configuration Manager [Електронний ресурс]: – manageengine.com - 2020. – Режим доступу: https://www.manageengine.com/network-configuration-manager/
6. RANCID - Really Awesome New Cisco confIg Differ [Електронний ресурс]: – shrubbery.net- 2017. – Режим доступу: https://shrubbery.net/rancid/
7. rConfig – A Network Device Configuration Management Tool [Електронний ресурс]: – unixmen.com- 2020. – Режим доступу: https://www.unixmen.com/rconfig-network-device-configuration-management-tool/

8. Puppet's architecture [Електронний ресурс]: – puppet.com - 2020. – Режим доступу: https://puppet.com/docs/puppet/5.5/architecture.html
9. Chef vs Puppet vs Ansible vs Saltstack: Which Works Best For You? [Електронний ресурс]: – edureka.com - 2019. – Режим доступу: https://www.edureka.co/blog/chef-vs-puppet-vs-ansible-vs-saltstack/
10. Deployment Management Tools: Chef vs. Puppet vs. Ansible vs. SaltStack vs. Fabric [Електронний ресурс]: – overops.com - 2015. – Режим доступу: https://www.overops.com/blog/deployment-management-tools-chef-vs-puppet-vs-ansible-vs-saltstack-vs-fabric/
11. Ansible overtakes Chef and Puppet [Електронний ресурс]: – techrepublic.com - 2019. – Режим доступу: https://www.techrepublic.com/article/ansible-overtakes-chef-and-puppet-as-the-top-cloud-configuration-management-tool/
12. What is Network Orchestration? [Електронний ресурс]: – techrepublic.com - 2019. – Режим доступу: https://www.appviewx.com/education-center/what-is-network-orchestration/
13. Git as a NoSql database [Електронний ресурс]: – kenneth-truyers.net - 2016. – Режим доступу: https://www.kenneth-truyers.net/2016/10/13/git-nosql-database/
14. SwitchConfAnalyzer [Електронний ресурс]: – github.com - 2018. – Режим доступу: https://github.com/hans-vvv/SwitchConfAnalyzer
15. 5 Things You Can Do With AWX [Електронний ресурс]: – ansible.com - 2017. – Режим доступу: https://www.ansible.com/blog/5-things-you-can-do-with-awx

Додаток А

```
#Playbook for creating backups
- hosts: all
  gather_facts: yes
  connection: ssh

  vars:

    git_repo: ~/ansible/ansible-net-devices
    git_sync: enabled

  tasks:

    #Change branch to backup configs
  - name: Change branch to backup configs
    with_items:
      - pwd
      - git branch -r | grep -v '\->' | while read remote; do git branch --track
"${remote#origin/}" "$remote"; done
      - git fetch --all
      - git pull --all
      - git checkout -b backup
    args:
      chdir: "{{ git_repo }}"
    when: git_sync == "enabled"
    delegate_to: 127.0.0.1
    run_once: True

    #Obtain backup configurations and set new without commit to device
  - name: Get candidate(backup) config with NAPALM
    napalm_install_config:
      hostname: "{{ inventory_hostname }}"
      username: "{{ user }}"
      password: "{{ password }}"
      dev_os: "{{ os }}"
      provider: "{{ ios_provider }}"
      candidate_file: "{{ git_repo }}/configs/{{ inventory_hostname }}/{{
inventory_hostname }}.running.conf"
      commit_changes: False

    #Commit to and push 'backup' git branches
  - name: Push backup branch
    with_items:
```

```
      - pwd
      - git fetch --all
      - git pull --all
      - git checkout backup
      - git add .
      - git commit -m "Updates {{ansible_date_time.date}}"
      - git push

    args:
      chdir: "{{ git_repo }}"
    when: git_sync == "enabled" and "{{ map('extract', hostvars)[-1] }}" == "{{
inventory_hostname }}"
    delegate_to: 127.0.0.1
```

#Playbook for pushing specified branch to devices

```
- hosts: all
  gather_facts: yes
  connection: ssh

  vars:
    git_repo: ~/ansible/ansible-net-devices
    git_merge_branch: pending
  tasks:

    #To restore configs set git_merge_branch: backup
    #To push pedning configs set git_merge_branch: pending
  - name: Merge selected branch to main branch
    with_items:
      - pwd
      - git branch -r | grep -v '\->' | while read remote; do git branch --track
"${remote#origin/}" "$remote"; done
      - git fetch --all
      - git pull --all
      - git checkout main
      - git merge {{ git_merge_branch }}
      - git push
    args:
      chdir: "{{ git_repo }}"
    delegate_to: 127.0.0.1
    run_once: True

    #Push new configs to devices
```

```
  - name: Push new config with NAPALM
    napalm_install_config:
      hostname: "{{ inventory_hostname }}"
      username: "{{ user }}"
      password: "{{ password }}"
      dev_os: "{{ os }}"
      provider: "{{ ios_provider }}"
      config_file: "{{ git_repo }}/configs/{{ inventory_hostname }}/{{
inventory_hostname }}.running.conf"
      commit_changes: True
```