НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

УДК 004.9:631.1

«ПОГОДЖЕНО»	**«ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ»**
Декан факультету інформаційних технологій	Завідувач кафедри комп'ютерних наук
Глазунова О.Г., д.п.н., професор	Голуб Б.Л., к.т.н., доцент
«__» _____ 2022 р.	«__» листопада 2022 р.

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему «Аналіз систем управління конфігураціями»

Спеціальність 122 «Комп'ютерні науки»

Освітня програма «Інформаційні управляючі системи та технології»

Орієнтація освітньої програми освітньо-професійна

Гарант освітньої програми «Інформаційні управляючі системи та технології»

д.т.н, доцент _____ Бондаренко В.Є.

Керівник магістерської кваліфікаційної роботи

к.т.н., доцент _____ Глазунова О.Г.,

Виконав _____ Зінєвич Д.С.

КИЇВ-2022

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ

Завідувач кафедри комп'ютерних наук

Голуб Б.Л., к.т.н., доцент

_____

"29" жовтня 2020 року

З А В Д А Н Н Я

ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ СТУДЕНТОВІ

Зіневичу Денису Сергійовичу

Спеціальність 122 «Комп'ютерні науки»

Освітня програма Інформаційні управляючі системи та технології

Орієнтація освітньої програми освітньо-професійна

Тема магістерської кваліфікаційної роботи «Аналіз систем управління конфігураціями» затверджена наказом ректора НУБіП України від "29" жовтня 2022р. №1634 «С»

Термін подання завершеної роботи на кафедру "30" листопада 2022р.

Вихідні дані до магістерської кваліфікаційної роботи:

1) дані практичного використання систем;
2) дані про використані підходи та особливості дизайну систем. Перелік питань, що підлягають дослідженню

№ з/п	Питання, що підлягає дослідженню	Строк виконання	Примітка
1.	Аналіз предметної області	21.09.2021 - 24.10.2021	
2.	Дослідження доступних систем	30.11.2021 - 31.12.2021	
3.	Рішення типової задачі	01.02.2022-06.03.2022	
4.	Дослідження переваг та недоліків	11.03.2021-10.04.2021	
5.	Аналіз отриманих результатів	01.09.2021-13.11.2021	
6.	Попередній захист	30.11.2022	
7.	Захист	14.12.2022	

Дата видачі завдання "29" жовтня 2021 р. **Керівник магістерської кваліфікаційної роботи** _____ Глазунова О.Г.

(підпис) (прізвище та ініціали)

Завдання прийняла до виконання _____ Зіневич Д.С.

(підпис) (прізвище та ініціали)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ITIL - Information Technology Infrastructure Library

ISC - Internet Systems Consorcium

SDLC - software development lifecycle

MVP - minimal value product

Opex - Operational expenses

Capex - capiral expenses

ERB - embeeded RuBy

API - Application Programming Interface

RBAC - Role Based Access Control

YAML - AML Ain't Markup Language

ВСТУП

Системи управління є невід'ємною складовою будь-якої сучасної інфраструктури. Кожна інформаційна система у світі яка працює в продукційному режимі являє собою комплекс програмних та апаратних засобів. Не достатньо написати програму (код) і запусти її. Коли справа доходить до експлуатації системи виявляється що навіть для найпростішою системи на кшталт веб сайту необхідно мати цілий комплекс супровідних систем. Для кожної системи постають питання надійності, моніторингу, резервного копіювання, безпеки, випуску нових версій і т.д. Будь яка програма що надає сервіс кінцевим користувачам в тій чи іншій формі теж має бути сконфігурована.

В цій роботі я розгляну наявні проблеми галузі та проведу порівняльний аналіз деяких доступних на ринку систем.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Інформаційні системи потребували конфігурації завжди. Будь яка програма (окрім найпростіших) зазвичай розроблювалась з думкою про те що її можна буде запустити в різних режимах або середовищах. Мова звісно йде про аплікації що використовуються на серверах і забезпечують мережеву взаємодію.

Враховуючи наскільки малий був світ мереж та серверів багатьох проблем просто не існувало. Кожна компанія яка вже мала сервер в тим чи іншім вигляді могла собі дозволити утримувати системного адміністратора (або групу) який керував би аплікаціями на сервері.

Вочевидь зріст кількості хостів підключених до мереж був спровокований винайденням і розповсюдженням мережі Інтернет в січні 1983го року [1]. Проте в ті часи як самі комп'ютери так і підключення їх до мережі залишались прерогативою військових та освітніх організацій. Більшості людей вони все ще були не потрібні. На рис. 1 показано кількість хостів. Відповідну статистику вела компанія ISC (Internets Systems Consorcvium) до 2019 року і базувала її на кількості виданих адреса IPv4. З розвитком технологій, особливо віртуалізації, підрахунок кількості хостів перестав бути технічно можливим.

Як видно на графіку кількість машин в світі під'єднаних до мережі в 1994 році становила близько 2-3 мільйонів. Але вже в ті часи деяки компанії та заклади використовували групи серверів для задоволення власних потреб. Це означало що хтось отримував в підпорядкування групу машин які треба було запустити і утримувати. З ростом кількості утримуваних машин власне і з'явилась потреба в ліпших способах управління. Маючи один, два або навіть десять серверів одна людина без проблем могла впоратись з їх підтримкою.

Власне в ті часи теж з'явилась традиція надавати серверам імена використовуючи пантеони стародавніх богів, назви опер, міфічних істот з культових кінофільмів або книжок. Невпинний зріст кількості машин змусив шукати рішення яке б дозволило знизити навантаження на обслуговуючий персонал та зменшити кількість помилок. Кожний відділ який мав більш ніж одну машину в обслуговуванні займався винайденням роверу під власні потреби — використовуючи в основному інтерпретовані мови програмування такі як SH, Perl працівники писали міні програми які б полегшували їм життя та автоматизовували конкретні задачі. Відсутність автоматизації в свою чергу мала прямий вплив на кошт утримання систем — потрібно було витратити більше часу на конфігурацію, людський фактор частіше призводив до помилок та простоїв.

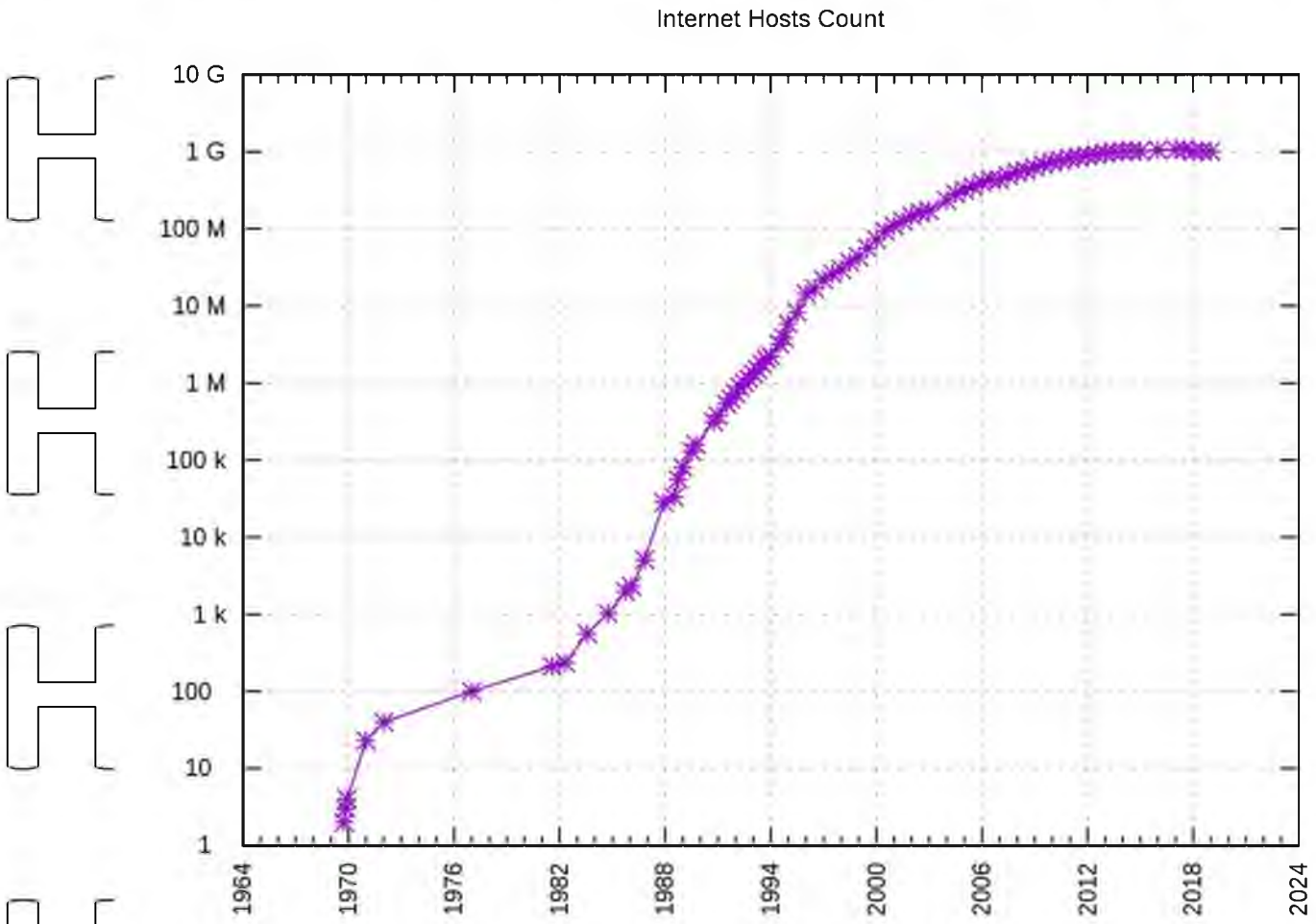

Рис.1 кількість хостів підключених до мережі Інтернет.

Створені власноруч системи автоматизації полігшували життя але мали очевидні вади — неможливість застосування в инших середовищах або конфігураціях. Простіше кажучи — кожна з таких систем могла працювати тільки на тому середовищі під яке вона була написана. Вартість застосування в инших умовах була близькою або еквівалентною написанню від нова.

Таким чином виникла потреба в системах управління конфігураціями. Системи які б дозволяли вирішувати типові задачі автоматизації та забезпечували певний рівень абстракції між користувачем та об’єктом управління. Тобто один і той сами код автоматизації міг би бути запущений наприклад на різних версіях операційної системи або взагалі на иншій ОС а результат виконання мав би залишитись незмінним.

Однією з перших таких систем була CFEnginge (Configuration Engine), написана Марком Бруггсом в 1993 році [2].

З розвитком технологій і геометричним зростом кількості серверів та доступних послуг в мережі Інтернет потреби в таких системах тільки зростали. На ринку почали з’являтись нові системи які намагались вирішити одну й ту саму задачу — управління великою кількістю машин. Деякі з них систем ми і розглянемо в цій роботі.

1.2 Управління конфігураціями

Сам термін з’явився в 50-х роках 20 сторіччя в Департаменті Оборони Сполучених Штатів Америки, і був визначений як: “Процес для встановлення і дотримування в припустимих межах продуктивності системи, функційних та фізичних атрибутів згідно з вимогами, дизайном і операційною інформацією впродовж її життєвого циклу”.

Стосовно галузі IT методологія ITIL [3] дає наступне визначення: “Процес відповідальний за гарантування того що активи необхідні для

забезпечення роботи сервісів контролюються відповідним чином, та точна і надійна інформація щодо цих активів є доступною там і тоді де потрібно. Така інформація включає подробиці про те як активи були сконфігуровані і зв'язки між ними"

Відповідно системи управління конфігураціями це програми які допомагають забезпечити виконання описаного вище процесу. Слід зазначити що система управління (програмний продукт) сам по собі не забезпечить виконання виробничого процесу, системи розглянуті в цій роботі є невід'ємною і головною, але тільки частиною процесу.

1.3 Мета проєкту та вхідні дані

Робота основана на досвіді автора котрий починаючи з 2005 року займається підтримкою різноманітних інформаційних систем.

В якості вхідних даних використані звіти з системи Jira різних проєктів та компаній, публічно доступний досвід деяких фірм.

Метою роботи є проведення порівняльного аналізу, висвітлення недоліків та переваг певних підходів та систем в залежности від умов застосування.

В роботі розглянуті найпоширеніші системи:

- CFEngine
- Chef
- Puppet
- Salt
- Ansible

1.4 Проблематика галузі

1.4.1 Популярність

Світом керує реклама і маркетинг. На жаль. Величезну роль в розповсюдженні технологій та інформаційних продуктів відіграє соціальний

фон — виступи на конференціях, історії успіху, спільнота що підтримує продукт. Великий вклад в домінування тих чи инших систем зробили саме ці чинники а не реальна технічна перевага. Тому зараз часто можна побачити доволі дивні комбінації систем та підходів які не рідко коштують більше ніж приносять користі.

1.4.2 Визначення

Визначення в ІТ галузі є доволі абстрактними, тому під те саме поняття при бажанні можна підвести часто дуже різні речі. Саме таким чином до списку потрапила система Ansible, яка на думку автора не є системою управління але набула шаленої популярності і наразі використовується дуже широко.

1.4.3 SDLC або його відсутність

Системи управління стають потрібні коли продукт потрібно масштабувати так коли він починає широко використовуватись. Наприклад для одного сайту рівня повин, скажімо https://hromadske.ua/, в такій системі не буде необхідності, все працюватиме на одному сервері який не вимагатиме великої кількості зусиль. Для системи рівня Rozetka.ua це буде життєвою необхідністю — тому що такого рівня система складається з багатьох компонентів пов’язаних між собою, має окремі середовища для розробки і тестування і т.д.

Життєвий цикл продуктів часто повторюється. Як правило спершу пишеться MVP і після цього приймається рішення про продовження чи припинення розробки. Не секрет що будь-який бізнес намагається заощадити кошти, зрештою це є основою економіки — витратити якнайменше і заробити якнайбільше. Тому на початкових стадіях бізнес майже ніколи не витрачає гроші на супутні системи. Це відповідно призводить до підвищення витрат уже в процесі експлуатації продукту. Стадії планування супутніх систем, нюанси розвитку і масштабування продукту на початку його розвитку часто пропускають, тож ці пробіли доводиться надолужувати вже на живо.

Виробничі системи експлуатуються ще багато років після завершення розробки. Такі системи прийнято називати “legacy”. І увесь час системою треба якось керувати. На додаток галузь IT є дуже динамічною — підходи і технології змінюються швидко. Тому не рідко систему написану скажімо 10 років тому доводиться переносити багато разі — з датацентру до “хмари”, потім до иншого хмарного провайдера, потім до контейнерів і так далі. Увесь цей час необхідно забезпечити надійність конфігурації і можливість масштабування системи.

1.4.4 Компетентність

Шалений темп розвитку галузі швидко призвів до кадрового голоду. На жаль це також вплинуло на якість працівників в галузі. Найпоширенішою проблемою менеджерів що наймають людей є пошук кандидатів з максимальним збігом списку технологій. Тобто наперед визначаються технології та інструменти і ведеться пошук працівників які б максимально відповідали такому спискові. Таким чином автоматично відрізаються потенційно придатні кандидати, часто з цінним досвідом.

Через такий підхід часто технічні рішення впроваджуються не беручи до уваги особливості і потреби конкретних проєктів, натомість просто копіюються рішення застосовані раніше де-інде.

1.5 Значення систем управління і конфігурації

Спершу розділимо інформаційні системи на три умовні розміри — малі, середні та великі.

Малі — від одного сервера до десяти.

Середні — від десяти до сотні.

Великі — від ста серверів.

Як було зазначено вище ефективність управління інфраструктурою напряму перекладається на операційну вартість (Opex). Для малих систем як правило зміни не є проблемними а от для середніх та великих можуть становити суттєві витрати. Наведу приклад одного з проєктів.

В 2011 році компанія Janrain Inc. прийняла рішення про впровадження системи Puppet. Первинне впровадження тривало близько року і зайнядо близько 600 людино-годин. Подальша підтримка системи та підключення до неї нових систем тривало ще два роки. В результаті експлуатації, в 2017-му році було прийнято остаточне рішення про заміну системи Puppet на Ansible. Вивід системи з експлуатації було закінчено лише в 2020 році. Впровадження Ansible тривало з 2015 по 2017 рік і зайняло більш ніж 1600 людино-годин.

Беручи середню вартість людино-години для компанії на той час (90$), можна порахувати що тільки на впровадження та вивід з експлуатації компанія витратила близько 200 тисяч доларів. Очевидно що для компанії enterprise рівня це не великі кошти, але це тільки кошт самих систем — не враховуючи час витрачений на пов'язані супутні проблеми, простої за збої систем, і т.д. Мало того, проблеми під час вибору та архітектурні помилки під час впровадження систем призвели до низької ефективності їх використання що суттєво зменшило отримані переваги та збільшило експлуатаційні кошти. Тому на масштабі проєктів з кількістю серверів більше тисячі і різноманіттям продуктів можна говорити про довготермінові видатки (або заощадження) на рівні мільйонів доларів.

2 МОДЕЛЮВАННЯ СИСТЕМИ

2.1 Методи push

Метод push (рис 2.) полягає на відсутності будь-яких спеціальних компонент в системі яку необхідно сконфігурувати. Увесь процес контролюється зі станції конфігурування, в найпростішому і найчастішому випадку — це є звичайна робоча станція.

Переваги:

- Брак необхідності встановлення додаткового ПЗ

- Простота в експлуатації- Швидкий запуск "з нуля" Недоліки:

- Підвищений кошт автоматизації самого процесу

- Більше залучення людини у процес, відповідно більша ймовірність людської помилки

- Більше навантаження на мережу і станцію конфігурування

- Необхідність мережевого та системного доступу

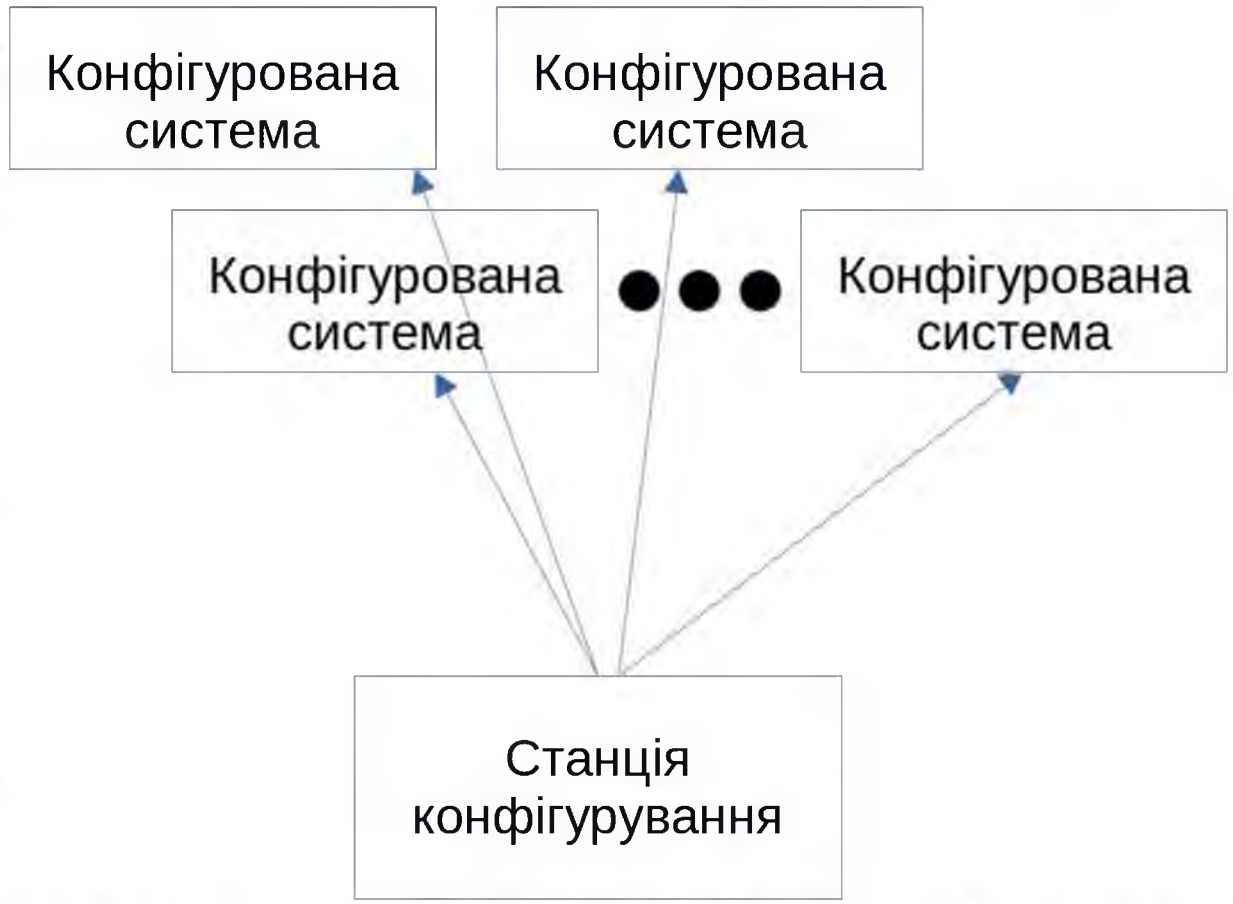

Рис 2. Метод "push"

2.2 Метод pull

Метод pull (рис. 3) полягає в тому що на кожній системі яку необхідно сконфігурувати має бути встановлено спеціяльний агент який власне і

відповідатиме за конфігурацію. Агент в свою чергу бере всю необхідну інформацію з серверу конфігурацій.

Переваги:

- Постійний контроль цільового стану системи
- Мінімізація участі людини
- Можливість застосування запобіжного механізму
- Відсутність прив'язки до наявності доступу Недоліки:
- Складний запуск в експлуатацію
- Складність контролю стану системи

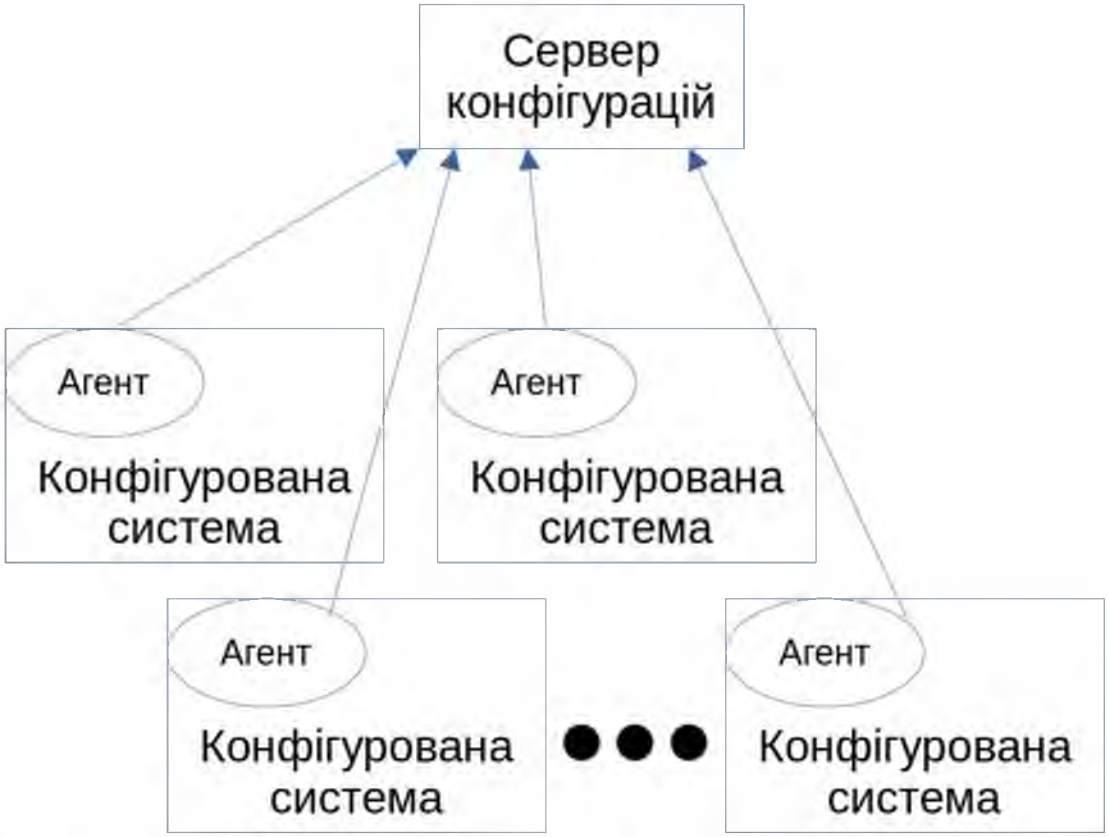

Рис. 3 Метод "pull"

2.3 Використані технології

2.3.1 CFEngine

Є однією з найстаріших систем, написана доктором Марком Бургессом в університеті Осло 1993 року. З того часу система була переписана кілька разів, актуальною версією є CFEngine3.

Цікавим є факт що це чи не найперший випадок коли системне адміністрування і управління конфігурацією було описано за допомогою математичної моделі.

Тоді ж були визначені деякі засадничі принципи за якими має бути побудована система конфігурацій:

- Робота системи в напрямку збіжності (конвергенції). Кожна операція зміни зроблена агентом повинна мати інваріантний характер.
- Замість описувати кроки які необхідно виконати система має описувати сам кінцевий стан.

2.3.3 Chef

В 2009-му році Adam Jacob написав систему “Chef” для своєї консалтингової компанії яка власне займалась адмініструванням серверів. Продукт було написано на Ruby та Erlang і для опису стану системи використовувалась мова Ruby (Ruby DSL), що зробило систему популярною в світі Ruby програмістів та дало поштовх для подальшої популяризації та розвитку.

Система використовує концепти “рецептів” (recipes) які в свою чергу збираються у “кулінарні книги” (cookbooks). Chef може працювати як серверному режимі так і самостійно (standalone). Chef використовується як основа для сервісу AWS Opsworks.

Чисельні проблеми з масштабуванням серверу призвели до того що в переважно використовується режим Chef solo, без встановлення серверу, відповідно сильно обмежується використання системи.

2.3.4 Puppet

Компанія Puppet Inc. була заснована в 2005 році і з того часу займається підтримкою продукту Puppet та випуском комерційної версії Puppet Enterprise. Різні компоненти продукту написані з використанням таких мов як Ruby (puppet), C++ (Facter) та Clojure (puppet server).

Продукт можна запустити як в серверному режимі так і окремо. Для опису стану системи продукт використовує так звані “маніфести” які збираються в групи — “каталоги”. Маніфести пишуться спеціальною мовою властивою тільки для цього продукту.

Деякий час продукт був доволі популярним завдяки активній спільноті що його просувала і підтримувала. Через проблеми з масштабуванням зацікавленість продуктом в певний момент впала.

2.3.5 Salt

Salt є свіжішим продуктом, прототип було випущено в 2011 році, і враховував нову реальність — масовий перехід до віртуалізації та хмарних сервісів. Тому продукт було написано з думкою про простоту розширення сторонніми розробниками та враховуючи потреби управління ресурсами в хмарних сервісах.

В якості мови опису конфігурації продукт використовує YAML що очевидно є значною перевагою. Сам же продукт написано мовою Python, що безумовно спрощує інтеграцію та написання модулів але водночас накладає відповідні обмеження.

Продукт може використовуватись як в серверному так і в самостійному режимах.

2.3.6 Ansible

Наймолодший продукт, випущений 2012 року. Продукт було створено керуючись наступними принципами:

- Мінімалізм
- Консистентність
- Відсутність агенту на конфігурованій системі
- Простота у застосуванні
- Модульність

Продукт написано на мові Python, що спрощує створення модулів і накладає обмеження водночас.

Через простоту використання та велику спільноту продукт швидко набув шаленої популярності і попри всі обмеження є чи не найбільше уживаною системою.

Основним елементом є “play” які потім збираються в “playbook”, що можна перекласти як “запис” та “книга записів” відповідно.

3 РОЗРОБКА СИСТЕМИ

Для того шоб порівняти деякі особливості систем вирішимо типову виробничу задачу за допомогою кожної з них:

Створення конфігураційного файлу в залежності від середовища та перезапуск відповідного сервісу.

Також зупинимось на деяких тонкощах та особливостях.

3.1 CFEngine

apache.cf bundle

agent apache { vars:

172_17_0_3::

“site_name” string => “site1”;

files:

"/etc/apache2/sites-templates/site.conf.tmpl" copy_from => secure_cp("/opt/conf/apache/site.conf.tmpl", "$(sys.policy_hub)"), perms => mog("0600","root","root");

"/etc/apache2/sites-available/site.conf" perms => mog("0600","root","root"), create => "true", edit_defaults => empty, classes => if_repaired("restart_apache"),

edit_template => "/etc/apache2/sites-templates/site.conf.tmpl";

commands:

restart_apache::

"/usr/sbin/service apache2

restart";

}

site.conf.tmpl

```
<VirtualHost *:80>
    ServerName ${apache.site_name}.acme.com

    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Вище лише приклад функціонального коду, для того щоб система запрацювала необхідно виконати ще ряд підготовчих операцій.

В додатку 1 наведено список дій які треба виконати для того щоб запустити систему в мінімальному робочому варіанті.

Основним “робочим” елементом продукту CFEngine є bundle. Bundle є логічною одиницею і потрібен для групування конфігураційних елементів. Тобто логічно конфігурацію однієї аплікації повністю описати у відповіднім bundle. В середині bundle описується бажаний стан конфігурованої одиниці. Порядок виконання визначений заздалегідь, якщо в bundle не було описано спеціальних умов то конфігурація буде виконана в наступним порядку:

meta

vars

defaults

classes

users

files

packages

guest_environments

methods processes

services

commands

storage

databases

reports

Наведені вище типи конфігурації в мові CFEngine мають назву promises — обіцянки, що власне віддзеркалює ідеологію продукту згідно з якою описується бажаний стан системи а не шлях його досягнення.

Кожен з типів конфігурації відповідає за контроль стану відповідних одиниць конфігурації. Наприклад files — забезпечує конфігурацію файлів (копіювання, шаблони, права, тощо), packages — операції з пакетами (встановлення, видалення), users — контроль користувачів. Деякі типи є особливими — наприклад meta та reports, що забезпечують довідкові дані та звіти про запуск відповідно.

Як видно з наведеного вище прикладу, продукт використовує власну мову описування конфігурації яка є доволі складною. На практиці це є чи не найбільшим відлякуючим фактором для широкого загалу.

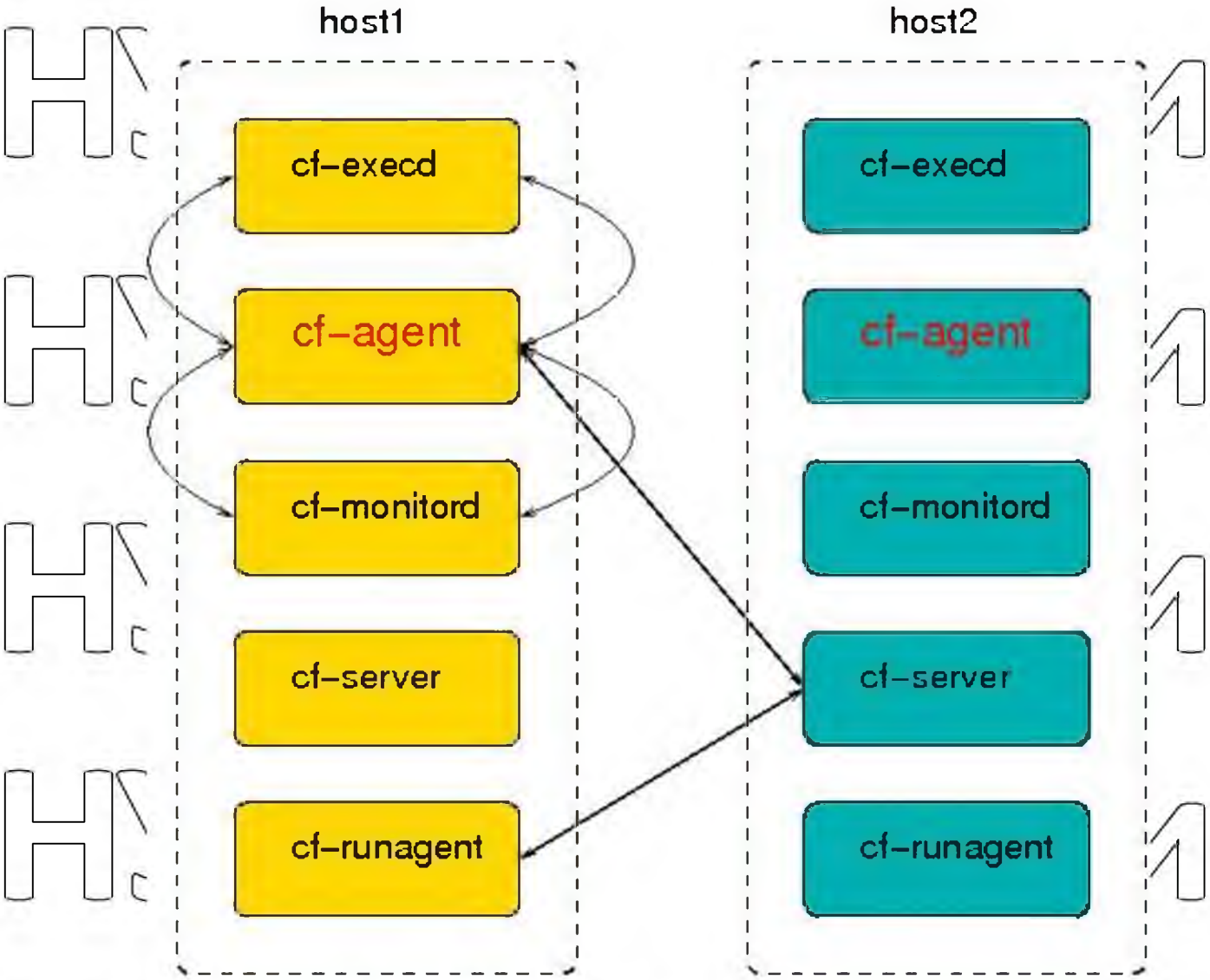

Рис. 4 Структура компонентів CFEngine

cf-promises

Компілятор та верифікатор, використовується для того щоб перевірити описану конфігурацію перед запуском.

cf-agent

Система яка власне робить зміни, той компонент який виконує усі необхідні маніпуляції на цільовій системі.

cf-serverd

Серверна компонента що може відправляти файли та отримувати запити на запуск наявних політик на певних хостах.

cf-execd

Сервіс планування що може бути доповненням до сервісу cron або працювати самостійно. Також він виконує функцію обгортки — запускаючи та збираючи вивід системи cf-agent.

cf-runagent

Допоміжна програма що може комунікувати з cf-serverd та зробити запит на виконання cf-agent з наявною конфігурацією. Таким чином програма може бути використана для примусового (поза графіком) виконання змін на хостах.

cf-report

Ця програма генерує підсумки та звіти у різноманітних форматах доступних для експорту та інтеграції з иншими системами. cf-know

Цей агент може генерувати мапу тем в форматі ISO (Topic Map) використовуючи політики що описують відомості про систему. Використовується для створення документації.

3.2 Chef

recipes/apache.rb template '/etc/apache2/sites-available/site.conf' do source 'site.conf.erb' mode '0440' owner 'root' group 'root'

variables(site_name: node['site_name']) notifies :restart, 'apache2_service[default]',:delayed end

apache_service 'default' do action :nothing end

attributes/default.rb default[site_name] = SiteName

templates/default/site.conf.erb

```
<VirtualHost *:80>
    ServerName <%= @site_name %>.acme.com
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

```
environment.rb name "development"
description "The master development branch"
cookbook_versions({
  "site" => ">= 1.0"
}) override_attributes ({
"site_name" => “dev-site”
})
```

В додатку 2 наведено список дій необхідних для мінімального запуску продукту.

Первинно написаний на Ruby продукт використовує Ruby DSL в якості мови опису конфігурацій. Очевидно ERB використовується для роботи з шаблонами. Це власне послужило потужним поштовхом для популяризації системи серед спільноти Ruby розробників. Використання вже знайомої мови програмування дозволяє заощадити час та кошти на початкових стадіях проєкту.

Також варто зазначити що через обмеження в 25 агентів для безкоштовної версії Chef Server продукт в основному використовується в режимі Chef Solo, тобто запуск відбувається зі станції конфігурації (часто

звичайної робочої станції). Таким чином продукт фактично втрачає частину доступного безкоштовно функціоналу.

Chef використовує кухонну термінологію. Наймешою одиницею є resource (ресурс), який описує бажаний стан якогось елементу — сервісу, пакету, файлу, тощо. Ресурси групуються в рецепти (recipe), які збираються в cookbooks (кулінарні книги). Таким чином впроваджується кілька рівнів абстракції, на кожному рівні додаються функціональні можливості що дозволяють обробляти що разу складніші конфігурації.

Recipe:

- написано на Ruby
- є колекцією ресурсів визначених за допомогою доступних шаблонів, допоміжний код може бути додано у разі необхідности
- повинен визначити все що є необхідним для того щоб сконфігурувати певну частину системи
- повинен зберігатися в книзі (cookbook)
- може бути включено до іншого рецепту
- може використовувати результати пошукового запиту або дані з пакунку даних (data bag)
- може декларувати залежність від одного (або більше) рецептів
- повинен бути доданий до списку запуску (run-list)
- завжди виконується в порядку зазначеному в списку запуску

Cookbook

Книга визначає сценарій та включає все необхідне для підтримки цього сценарію.

- рецепти що визначають які ресурси мають бути сконфігуровані та порядок в якім це має відбутись
- значення атрибутів що дозволяють визначити параметри середовища
- додаткові ресурси що дозволяють розширити стандартні можливості
- файли та шаблони

- файл metadata.rb що описує книгу та її залежности

НУБіП України

НУБіП України

НУБіП України

НУБіП України

НУБіП України

НУБіП України

НУБіП України

Рис. 5 Архітектура Chef Server

Компоненти системи Chef Server:

Клієнти — Chef Server в основному обслуговує запити від хостів що знаходяться під управлінням системи під час роботи клієнта. Також він обслуговує запити від осіб що підтримують книги та політики які зберігаються

в Chef Server, як правило з робочих станцій. Також запити що робляться за допомогою Chef management console (web інтерфейс).

Load Balancer - The signed URLs for cookbooks are pointed here, and then routed to cookbook storage, as required.

Chef Infra Server - Erchef є повною заміною основного API (core API) для Chef Infra Server, що забезпечило ліпше масштабування та видайність в порівнянні з попередніми версіями. API залишилось сумісним з оригінальною версією написаною на Ruby, це означає що книги та рецепти написані для старого API надалі працюватимуть без жодних проблем. Клієнт залишився на Ruby. Chef Manage - Chef Manage є web інтерфейсом для Chef Infra Server, що використовує Chef Infra Server API для всіх комунікацій з Chef Infra Server.

Bookshelf - Bookshelf використовується для зберігання вмісту книг (cookbook content) - файлів, шаблонів і т.д., всього що було завантажено на сервер разом з книгою. Вміст книги контролюється за допомогою контрольної суми. Якщо дві різні книги або дві різні версії тієї самої книги містять в собі однаковий файл або шаблон — bookshelf збереже тільки одну копію.

Вміст книг яким керує Bookshelf зберігається в звичайних файлах окремо від серверу та пошукових індексів репозиторіїв. Всі книги зберігаються в окремому репозиторії.

Messages - chef-elasticsearch є обгорткою для Elasticsearch і дає доступ до його REST API для індексування і пошуку. Всі повідомлення додаються до окремого пошукового індексу.

PostgreSQL - база PostgreSQL використовується в якості основного місця зберігання даних для Chef Infra Server.

3.3 Puppet

```
nodes.pp node
'site1.acme.com' {
  include dev-site
}
```

```
class dev-site {
$site_name ="site1"
}
```

site.conf.erb

```
<VirtualHost *:80>
    ServerName <%= @site_name %>.acme.com
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

site.pp

```
class site_conf {
service { 'apache2':
ensure => 'running',
enable => true
 }

file { '/etc/apache2/sites-available/site.conf':
content => template('site/site.conf.erb'),
owner => root, group => root, mode
=> 644,
notify => Service['apache2']
}
}
```

В додатку 3 наведена процедура підключення агента до сервера.

Також написаний на Ruby продукт використовує свою мову для опису стану системи. Для роботи з шаблонами використовується ERB.

Щоб сконфігурувати ноди Puppet сервер компілює конфігураційну інформацію в каталог що описує бажаний стан ноди де запущено агента. Кожен агент робить запит і отримує індивідуальний каталог.

Каталог описує бажаний стан кожного керованого ресурсу на певній одній ноді. Маніфест натомість може містити умови і логіку що описує конфігурацію ресурсів на групі серверів.

Щоб створити каталог для конкретного агента сервер компілює:

- Дані з агенту — факти про систему та сертифікати
- Зовнішні дані, такі як результат роботи функцій або дані з PE консолі
- Маніфести що можуть містити додаткову логіку що описує бажаний стан ресурсів на групі серверів.

В процесі компіляції сервер виконує необхідні обчислення та функції для того щоб на виході отримати статичний каталог та передати його агенту.

A resource describes some aspect of a system, such as a specific service or package. You can group resources together in classes, which generally configure larger chunks of functionality, such as all of the packages, configuration files, and services needed to run an application.

Мову Puppet структуровано навколо визначень ресурсів. Ресурс визначає бажаний стан об’єкту який конфігурує Puppet. Всі інші елементи мови додають рівні абстракції для збільшення гнучкости і зручности конфігурування.

Так само як визначаються окремі ресурси можна визначити клас для того щоб керувати групою ресурсів. В той час як ресурс визначає стан конкретного об’єкту, клас дозволяє згрупувати усі необхідні ресурси необхідні для забезпечення функціонування якоїсь аплікації — файли, пакети, сервіси, і т.д. Менші класи можуть бути об’єднані в більші що визначають більші структурні одиниці — наприклад сервер баз даних, сервер аплікацій і т.д.

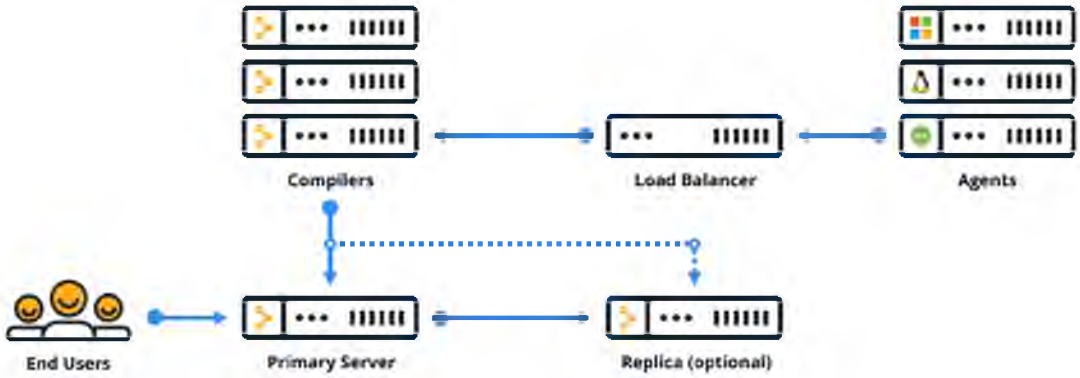

Рис. 6 Архітектура puppet

Facter

Facter являє собою крос-платформену бібліотеку та є невід'ємною частиною продукту. Він використовується для виявлення доступної інформації про систему на певній системі, пізніше знайдені “факти” є доступними в puppet маніфестах в постат змінних.

3.4 Salt

site.sls

/etc/apache2/sites-available/site.conf:

file.managed:

- source: salt://apache/site.conf

- user: root

- group: root

- mode: 644

- attrs: ai

- template: jinja - defaults: site_name: "default-site"

service.running:

- reload: True

- enable: true - watch:

- file: /etc/apache2/sites-available/site.conf

pillar/edit/site.sls

{% if grains['id'].startswith('dev') %}

site_name: “dev-site”

{% elif grains['id'].startswith('qa') %}

site_name: “qa-site” {% else %}

site_name: “prod-site”

{% endif %}

site.conf

<VirtualHost *:80>

ServerName {{ pillar[‘site_name’]}}.acme.com

ServerAdmin webmaster@localhost

DocumentRoot /var/www/html

ErrorLog ${APACHE_LOG_DIR}/error.log

CustomLog ${APACHE_LOG_DIR}/access.log combined

</VirtualHost>

В додатку 4 наведено список дій необхідних для мінімального запуску сервера та агента.

Salt базується на зберіганні конфігурацій або “статусу” в легких до зрозуміння структурах даних. Основними пунктами концепції продукту є:

Простота — легкість в адмініструванні.

Розширюваність — легкість при додаванні модулів або розширюванні наявних.

Детермінованість — таке саме виконання коду кожного разу.

Багатошаровість — забезпечує шари абстракції.

Концепції управління Salt

Salt master

Сервер на якому запущено програму salt-mater є відповідно управляючим сервером. Salt-master забезпечує зв’язану платформу для автоматизації та оркестровки підпорядкованих систем.

Salt minion

Salt minion може бути будь-яка керована за допомогою Salt система. На керованій системі може бути запущено сервіс salt-minion або можна використати salt-ssh чи salt-proxy. Salt-minion запущений як сервіс може виконувати команди на системі без участі майстер сервера.

Salt proxy

Salt proxy використовується у випадках коли на пристрої неможливо запустити сервіс minion. Proxy сервіс отримує команди від серверу, транслює і передає команди у відповідний спосіб для певного пристрою (SSH, REST, і т.д.), і передає результати виконання назад на сервер.

Salt SSH

Salt SSH було додано як альтернативний засіб комунікації з підпорядкованими системами. В цьому випадку немає вимоги встановлювати minion на сервері, вистачить наявного сервісу SSH.

SaltStack Config

SaltStack Config забезпечує інтуїтивний інтерфейс користувача що дозволяє виконувати складні задачі. За допомогою цього сервісу можливо створити та запланувати задачі. Також цей інтерфейс дозволяє розподіли задачі на адміністративному рівні — надати різний рівень доступу в залежності від кваліфікації працівника.

Можливості SaltStack Config включають:

- Web інтерфейс

- Ролевий контроль доступу (RBAC)
- Підтримку декількох майстер серверів
- Централізований кеш для задач та подій
- Інтеграцію з LDAP, SAML, OIDC, Active Directory
- Політики безпеки на рівні галузевих стандартів, таких як CIS та DISA STIGS
- Звітність
- API (eAPI)

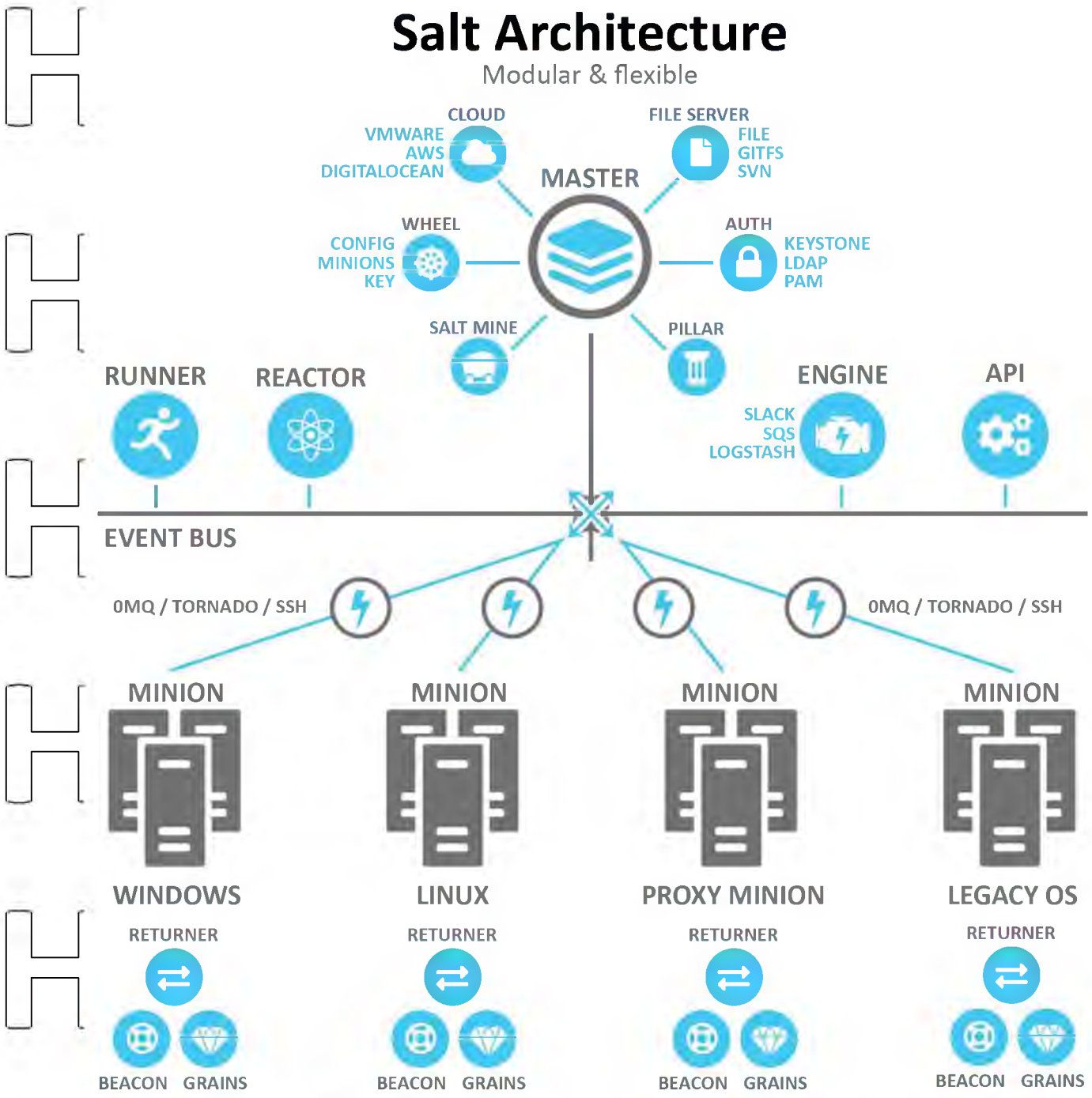

Рис 7. Архітектура SaltStack

Топологія Salt

Salt використовує два порти міньйонів для комунікації з майстер сервером. Ці порти використовуються одночасно для передачі даних до Шини Даних (Message Bus). Salt використовує ZeroMQ в якості шини даних, що

дозволяє створити асинхронну мережеву топологію та забезпечує максимально швидку комунікацію. **Система відкритих подій**

Система подій (event system) слугує для міжпроцесної комунікації — між майстром та міньйонами:

- Події видно і на майстрі і на міньйонах одночасно.
- Події можуть прослідковуватись та перевірятись з обох кінців.
- Шина даних забезпечує основу для оркестровки та моніторинг в режимі реального часу

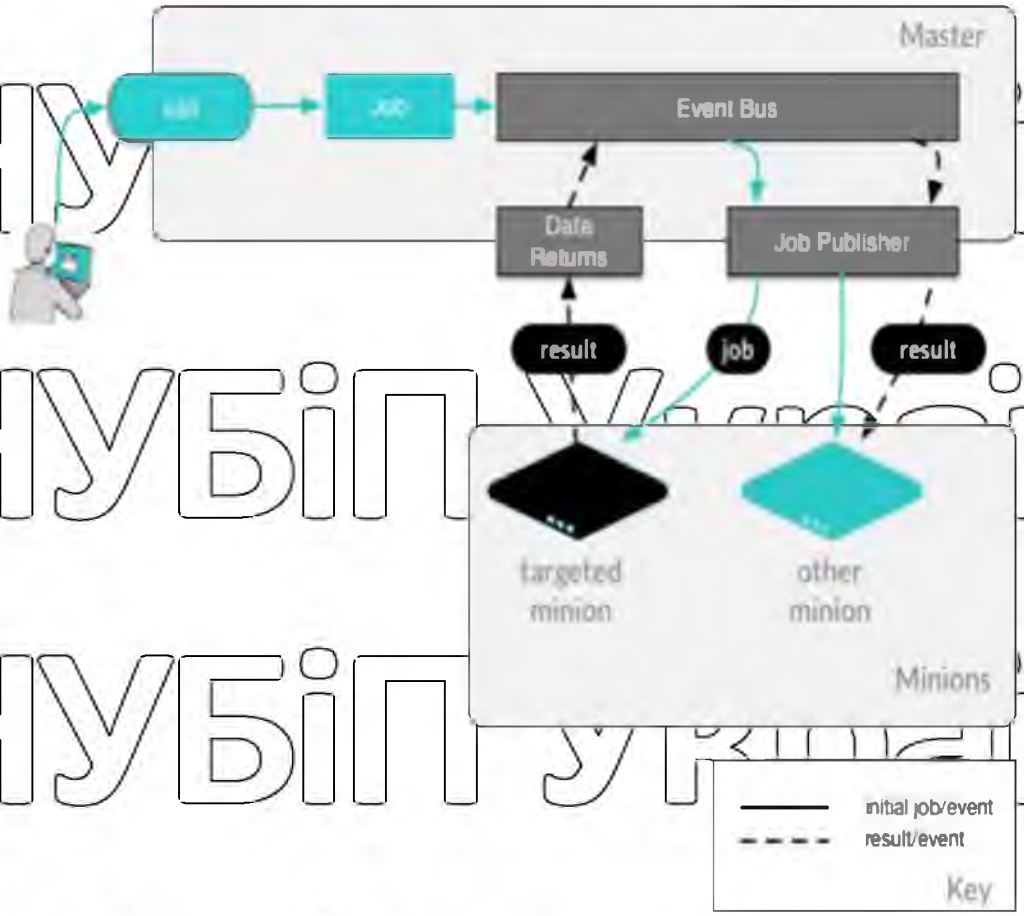

Рис 8 Система повідомлень/подій

Швидкісна комунікаційна шина

Однією з найбільших переваг Salt є швидкість виконання. Шина комунікації що базується на подіях є більш ефективною ніж рішення вищого рівня, на штиб HTTP. Система віддаленого запуску є основою для усіх инших

компонент, забезпечуючи можливість децентралізованого віддаленого запуску і розподілу навантаження відповідно.

Формат конфігурації YAML Salt

Формат YAML є приязним людям форматом серіялізації для будь-яких мов програмування. Це не мова розмітки як XML, що використовує теги для відмічання тексту. Натомість YAML фокусується на структурах даних, таких як списки та словники.

3.5 Ansible

site.yaml

- hosts: dev_servers vars: site_name: "dev-site"

- name: Template a file to /etc/apache2/sites-available/site.conf ansible.builtin.template: src: site.j2 dest: /etc/apache2/sites-available/site.conf owner: bin group: wheel mode: '0644'

site.j2

<VirtualHost *:80>
ServerName {{ site_name }}.acme.com
ServerAdmin webmaster@localhost
DocumentRoot /var/www/html
ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>

Ansible є інструментом автоматизації. З його допомогою можна конфігурувати системи, встановлювати програмне забезпечення, та диригувати більш складні задачі, такі як тяглі встановлення або так звані безперервні накочувані оновлення (rolling updated).

Головною метою Ansible є простота та легкість у використанні. Продукт також серйозно сфокусовано на безпеці та надійності, використовуючи мінімум рухомих частин, використовуючи OpenSSH в якості транспорту, та мові що запроєктована навколо ідеї приязности для людей незнайомих з продуктом.

Згідно з ідеологією Ansible простота можлива для інфраструктур будь-якого масштабу, тому система націлена на максимально широкий загал користувачів — програмістів, адміністраторів, реліз інженерів, тощо.

Ansible керує серверами без використання агенту. Таким чином відпадає потреба у встановленні агенту, контролі його стану і т.д. Так як OpenSSH є широко розповсюдженим та зрілим продуктом — безпека комунікації є гарантованою. Ansible є децентралізованою системою та використовує авторизаційні дані машини на якій його запущено.

Ansible контролює бажаний стан ресурсів на керованих машинах.

Базове середовище Ansible включає всього лиш три компоненти:

Контрольна нода (Control node)

Система на якій встановлено Ansible є керуючою.

Керована нода (Managed node)

Система якою керує Ansible

Список інвентаризації (Inventory)

Список логічно групованих контрольованих систем. Список створюється на контрольній ноді та описує системи якими необхідно керувати.

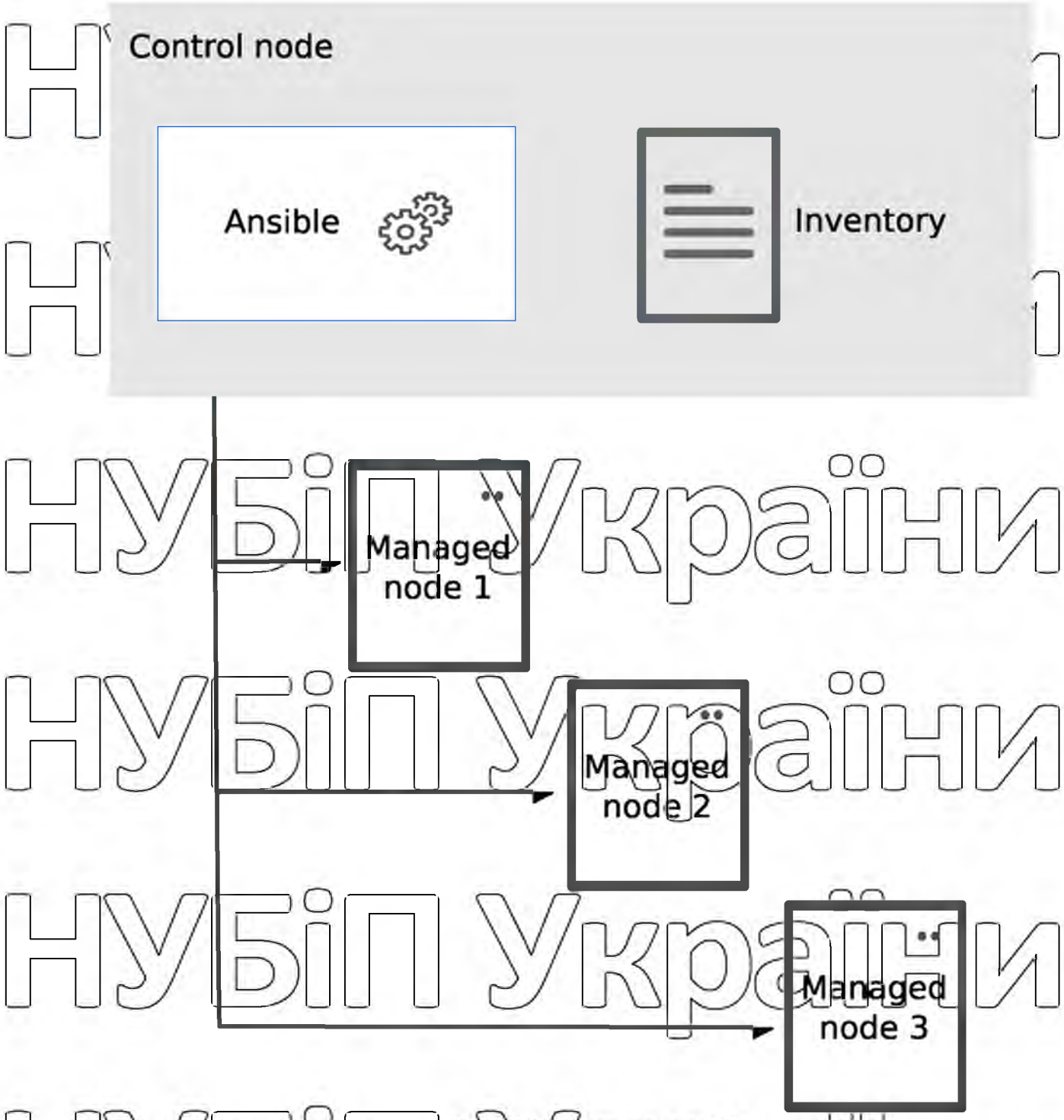

Рис. 9 Структура Ansible

4 РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

Автор має безпосередній досвід експлуатації інформаційних систем в якості системного адміністратора та IT менеджера. Протягом п’ятнадцяти років отримано досвід експлуатації як взагалі без систем управління конфігураціями так і з кожною з описаних.

4.1 Термінологія

Варто зазначити що на думку автора не всі системи підпадають під визначення “система управління конфігураціями”. На практиці на жаль, в основному через низький рівень освіченості, поняття змішуються.

З точки зору IT інфраструктури системою управління конфігураціями можна назвати тільки ту систему яка забезпечує безперервний контроль за станом об’єктів управління. Тобто таку систему яка працює в автономному режимі і не потребує втручання оператора. Таким чином до таких систем можна віднести CFEngine, Chef, Puppet та Satl.

Системи які для запуску чину конфігурації потребують оператора варто окреслити як “системи забезпечення” (provisioning system). Такі системи не функціонують самостійно. До таких системи відноситься безумовно Ansible, а також беручи до уваги ліцензійні обмеження продукту Chef — Chef Solo.

На практиці в залежности від рівня зрілості проєкту та інфраструктури багато хто намагається вже маючи впроваджену систему Ansible допровадити її до стану системи управління конфігураціями. Для досягнення цієї мети використовуються сторонні продукти або пишуться локальні рішення. Наприклад не рідким є випадок коли замість того щоб запускати Ansible зі станції управління пишеться обгортка яка запускає Ansible на кожній конфігурованій ноді що якийсь час за допомогою cron або systemd.

4.2 Мови програмування

Для довгострокової експлуатації дуже важливим виявляється факт на якій мові написано той чи инший продукт.

Мови поділяються на інтерпретовані та компільовані.

Компільовані мови натомість поділяються на мови статичної компіляції та мови динамічної компіляції.

4.3 Переносність

Це є важливим тому що в нормальному продукційному циклі програмні продукти експлуатуються роками або десятиліттями. Це означає що протягом життєвого циклу програмного продукту він може бути запущений на великому розмаїтті операційних систем та середовищ.

Інтерпретовані мови в цьому випадку серйозно програють. Конфігурація написана на Ansible скажімо 8 років тому під Ubuntu 12 або 14 сьогодні просто не запуститься на Ubuntu 22. Це збільшує кошт експлуатації самої системи управління конфігураціями.

Окрім того існує пряма залежність від інтерпретатора. Якщо за замовчуванням в Ubuntu 14 був Python 2.4 то на сьогодні в Ubuntu 22 маємо Python3.7, відповідно Ansible який працював з Ubuntu 14 більше не працюватиме.

Модульність таких продуктів довгостроково теж не рідко відіграє злу роль. Щоб запустити якусь більш-менш складну конфігурацію на новішій системі необхідно впевнитись що не тільки саме ядро системи конфігурації є сумісним з певною системою але і всі використані модулі.

На практиці системи що використовують інтерпретовані мови (Python, Ruby) потребують постійної підтримки та оновлення. У разі необхідності підтримки операційних систем різних версій — ускладнення коду.

4.4 Продуктивність та стабільність

Майже всі продукти написані на інтерпретованих мовах програмування маю або мали поважні проблеми з продуктивністю та стабільністю.

На практиці це означає що в якийсь момент сервер або агент перестають працювати. Це призводить до збоїв, збільшення кошту експлуатації системи (необхідно додавати моніторинг і т.д.).

Найчастіші проблеми це:

- Сервер не витримує великої кількості агентів (Chef, Puppet)

- Через витік пам’яті або подібні помилки агент перестає запускатись або виконує не всі кроки з конфігурації.

Саме з цієї причини свого часу серверну частину Chef було переписано з Ruby на Erlang.

У випадку Puppet часто спостерігались проблеми з агентами. Хоча згідно з документацією система мала б виконувати всі кроки під час кожного запуску, на практиці виявилось що досить легко по кожному запуску отримати різний результат виконання.

Серед представлених систем лише CFEngine написано на мові C, що означає повну незалежність від стану середовища. Якщо є готовий агент для певної операційної системи то CFEngine працюватиме на цій системі завжди, незалежно від зміни версій пакетів на цій системі.

4.5 Мова конфігурації

Мова конфігурації — мова яку використовує продукт для опису об’єкту конфігурації, те з чим люди що експлуатують цей продукт матимуть безпосередній контакт увесь час. Це є мабуть найпершою річчю на яку звертають увагу при виборі системи. З точки зору людини простіші мови виграють. Часто програмісти які через брак кадрів змушені займатись адмініструванням вибирають мови або максимально наближені або безпосередньо мови програмування (Ruby, Python). Простіше кажучи — вкрай важко переконати людину що вона має вивчити ще якусь мову програмування “тільки” задля конфігурації систем.

4.6 Складність планування

Як було зазначено у вступній частині етап планування часто пропускається. Тому не рідко усвідомлення необхідності в системі управління конфігураціями приходить в той час коли команда вже не справляється з керуванням інфраструктурою в ручному режимі. В таких випадках як правило перевагу надають системі з якою більша частина існуючої команди вже має

досвід. Ніхто в таких випадках не враховує особливості проєкту та кошти пов’язані з експлуатацією тих чи инших систем.

4.7 Мода та переконання

Мода та маркетинг мають суттєвий вплив на технологічний світ. На сьогодні яскравим прикладом цього явища є Kubernetes — платформа що дозволяє керувати контейнерами. Ринок “гуде” і майже ніхто не розбирається чи доцільне використання цієї системи для конкретного продукту — всі хочуть запустити свій продукт саме в Kubernetes. Такі самі хвилі моди можна прослідкувати для систем управління конфігураціями. Протягом останніх 15 років різні системи набували та втрачали популярність, що не було пов’язано з технічними особливостями.

4.8 Масштабування

Вкрай важливий параметр про який часто забувають — масштабування. Керувати десятком серверів чи кількома тисячами — різні за рівнем складности задачі. Нижче наведено графік масштабування CFEngine в порівнянні до Ansible.

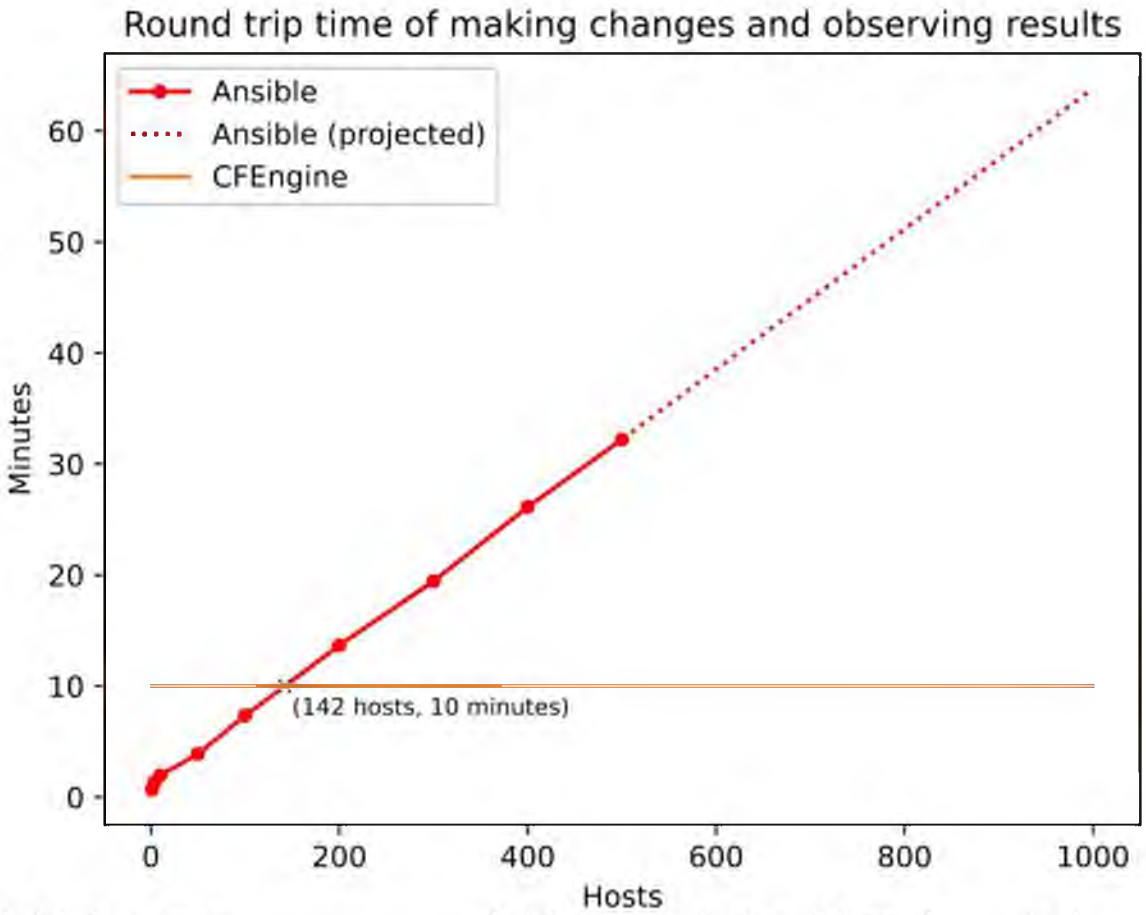

Рис. 9 Масштабування Ansible в порівнянні до CFEngine

Графік зображує залежність часу за який зміни фактично відбудуться від кількості хостів на який зміни впроваджуються.

На практині ситуація виглядає ще гірше і можна з впевненістю сказати що підхід “push” є неприйнятним для інфраструктури більшої за 50 серверів. В той час як такі системи як CFEngine що використовують підхід “pull” до певного моменту (більше тисячі агентів) не потребують жодних модифікацій пов’язаних з продуктивністю взагалі, системи що використовують “push” вимагають запуску на справді потужних станціях, великої пропускної здатності мережі та ряду оптимізацій.

Навіть за наявності швидкої мережі, потужної станції та оптимізованих налаштувань — запуск такої системи як Ansible на скажімо 20 машин займає багато часу та потребує уваги оператора.

ВИСНОВКИ

Системи управління конфігураціями є невід'ємною частиною будьякої продукційної IT інфраструктури. Від вибору системи і підходу до адміністрування напряму залежить кошт експлуатації інфраструктури. Скільки часу (грошей) коштуватиме зміна конфігурації на N хостах ? Скільки людей необхідно для підтримки тисячі серверів, п'ятеро чи десятеро ? Це питання на які можна дати відповідь за допомогою правильного планування і вибору відповідної системи.

Практика також показала що аргументи про складність деяких систем є мізерними. Кілька фактів із практики:

- Велика інфраструктура це завжди складно. Складні задачі не вирішуються просто.
- Спрощення складних задач породжує технічний борг. Якщо питання проігнорували або відклали зараз — воно неодмінно коштуватиме дорожче в майбутньому.
- Не варто робити ставку тільки на наявні знання інженерів. Технічний персонал здатен засвоїти нові підходи і продукти досить швидко, навіть персонал початкового рівня.

На сьогоднішній день найнадійнішою системою є CFEngine — це зрілий продукт написаний з науковим підходом. На жаль непринадність мови CFEngine відлякує потенційних користувачів цієї системи.

Система Salt є чудово спроєктованою але на жаль матиме проблеми з довгостроковою експлуатацією як будь-яка инша що має інтерпретовану мову за основу. Втім для однорідних проєктів здатних забезпечити тяглу однорідність протягом тривалого часу це буде мабуть найкращий вибір.

Для гетерогенних систем непоганою комбінацією може бути одночасне використання двох систем — CFEngine та Ansible. Проте такий підхід вимагатиме більше зусиль в плануванні та розмежуванні.

В майбутньому можливо з'явиться умовно ідеальна система —

система що використовуватиме ідеологію та підхід CFEngine/Salt, буде написана на компільованій мові зі статичною компіляцією та використовуватиме простий для пересічної людини синтаксис (YAML або подібний). Враховуючи всі проблеми з якими стикаються великі інфраструктури — потреба в такій системі існує вже давно, проте створення такого продукту є клопітким та довгим процесом.

Для малих інфраструктур можна зупинити свій вибір на Ansible, для середніх — Salt.

Додаток 1

Список дій необхідний для того щоб запустити один сервер на одного агента CFEngine.

Сервер:

wget https://cfengine-package-repos.s3.amazonaws.com/community_binaries/Community-3.18.2/agent_ubuntu20_x86_64/cfengine-community_3.18.2-1.ubuntu20_amd64.deb

dpkg -i cfengine-community_3.18.2-1.ubuntu20_amd64.deb

cp -r /var/cfengine/masterfiles/* /var/cfengine/inputs/

service cfengine3 start

Агент:

cf-agent -B $server_hostname

Додаток 2

Сервер

chef-server-ctl reconfigure

chef-server-ctl user-create USER_NAME FIRST_NAME LAST_NAME EMAIL 'PASSWORD' --filename FILE_NAME

// на цій стадії буде згенеровано приватний ключ

chef-server-ctl user-create janedoe Jane Doe janed@example.com 'abc123' --filename /path/to/janedoe.pem

chef-server-ctl org-create short_name 'ACME Inc.' --association_user user_name --filename ORGANIZATION-validator.pem

Робоча станція

knife configure

knife cookbook create web_site

knide cookbook upload web_site

Клієнт

/etc/chef/client.rb log_level :info log_location STDOUT node_name "nodename"

chef_server_url "https://chef.acme.com/acme"

syntax_check_cache_path '/root/.chef/syntax_check_cache'

validation_client_name 'chef-validator'

validation_key '/root/acme-validator.pem'

service chef-client start

Додаток 3

Сервер

```
# apt-get install puppetserver
# systemctl start puppetserver

// Підключення агента
# puppetserver ca sign --certname <name>
```

Агент

```
# apt-get install puppet-agent
# puppet config set server puppetserver.acme.com --section main
# puppet ssl bootstrap
// підписати сертифікат на сервері
# puppet ssl bootstrap
```

Додаток 4

Сервер

systemctl start salt-master

Агент

/etc/salt/mmion: master:

saltmaster.acme.com

salt-minion -d

Для успішної роботи необхідно провести обмін ключами.

На сервері:

salt-key -L

salt-key -a minion.acme.com

СПИСОК ВИКОРИСТАНИХ МАТЕРІЯЛІВ

1.History of The Internet — Wikipedia. URL: https://en.wikipedia.org/wiki/History_of_the_Internet

2.CFEngine — Wikipedia. URL: https://en.wikipedia.org/wiki/CFEngine

3.ITIL — Wikipedia. URL: https://uk.wikipedia.org/wiki/ITIL

4.CFEngine case study. URL: https://cfengine.com/case-studies/LinkedIn_CFEngine_Case_Study.pdf

5. Science of Computer Programming 49 (2003) 1 – 46

6. Science of Computer Programming 51 (2004) 197 – 213

7.Greg Sanker, IT Change Management: A Practitioner's Guide, ISBN: 0117083658, 9780117083653

8.Vratislav Podzimek (2020): Ansible|CFEngine, Solution for the whole infrastructure lifetime, URL: https://cfengine.com/wp-content/uploads/2020/09/AnsibleCFEngine_whitepaper_1.pdf

9. David Wilson (2019): Operon: Extreme Performance For Ansible, URL: https://sweetness.hmmz.org/2019-10-28-operon.html

10. O. Tange (2011): GNU Parallel - The Command-Line Power Tool, ;login: The USENIX Magazine, February 2011:42-47.

11.James Loope, Managing Infrastructure with Puppet, O'Reilly Media, Inc., ISBN: 9781449307639

12.Bas Meijer, Lorin Hochstein, René Moser, Ansible: Up and Running: Automating Configuration Management and Deployment the Easy Way, O'Reilly Media, ISBN: 978-1098109158

13.Matthias Marschall, Chef Infrastructure Automation Cookbook, Packt, ISBN: 9781785287947

14.Colton Myers, Learning SaltStack, Packt Publishing, ISBN: 978-1785881909

15.Evi Nemeth, Garth Snyder, Trent Hein, Ben Whaley, Dan Mackin. UNIX and

Linux System Administration Handbook, Addison-Wesley Professional, ISBN: 978-0134277554
16.Puppet documentation, URL: https://puppet.com/docs/puppet/7/intro_puppet_language_and_code.html#intro_puppet_language_and_code
17. Chef documentation, URL: https://docs.chef.io/server/
18. BMC change management blog, URL: https://www.bmc.com/blogs/configuration-management/