Issue 15, 2024

https://doi.org/10.23939/sisn2024.15.370

УДК 621.311.2

АНАЛІЗ ПРОГРАМНИХ ІНСТРУМЕНТІВ ДЛЯ АВТОМАТИЗАЦІЇ ФУНКЦІЙ КОНФІГУРУВАННЯ І УПРАВЛІННЯ В IT ІНФРАСТРУКТУРАХ

Микола Орлов[1], Юрій Дмитрів[2]

[1] Національний університет "Львівська політехніка",
кафедра інформаційних систем та мереж, Львів, Україна,
[2] Національний університет "Львівська політехніка",,
кафедра систем штучного інтелекту, Львів, Україна
[1] E-mail: orlov.nv.86@gmail.com, ORCID: 0009-0007-9835-6177
[2] E-mail: ydmytriv@gmail.com, ORCID: 0009-0001-7427-5387

Проаналізовано групу програмних інструментів, що функціонально зорієнтовані на автоматизовану реалізацію процесів конфігурування та управління в IT інфраструктурах. Профіль наукового дослідження фокусується на методиці, яка у фаховому середовищі називається "Інфраструктура як код" (IaC – Infrastructure as Cod) і є однією з базових методик, які в системному поєднанні реалізовують методологію DevOps. Зазначена методологія активно використовується в процесах динамічного формування, розгортання та супроводження корпоративних IT інфраструктур в багатьох сучасних успішних високотехнологічних компаніях для досягнення найкращих результатів ведення бізнесу, його ефективності, гарантованої успішності та захищеності. В статті розглянуто два базових підходи до побудови програмних інструментів, що реалізують методику IaC, йдеться про так звані декларативний та імперативний підходи.

При цьому основна увага зосереджується на формуванні множин факторів переваг та недоліків притаманних програмним інструментам Terraform, ARM, Ansibleta ClaudFormation. Фіксація уваги дослідників на перелічених вище чотирьох програмних інструментах пояснюється їх лідируючими позиціями в доволі розлогій лінійці можливих альтернативних програмних продуктів, які дозволяють комплексно реалізувати методику IaC в контексті повноцінного та повнофункціонального системного впровадження методології DevOps в конкретних реалізаціях корпоративних IT інфраструктур. Узагальнений висновок сформований авторами оригінальної наукової розвідки полягає в тому, що на даний час не існує одного чітко виокремленого серед інших універсального програмного інструменту, який би в повній мірі задовольнив весь спектр вимог та потреб. Потенційними користувачами при цьому виступають спільноти DevOps фахівців та замовників – власників та менеджерів сучасних динамічних високотехнологічних та успішних компаній, фірм та бізнесів, які опираються в своїй діяльності на сучасні інформаційні системи та технології.

Ключові слова: програмні інструменти; методологія DevOps; методика IaC; автоматизація функцій конфігурування; управління IT інфраструктурою; декларативний підхід; імперативний підхід.

Постановка проблеми

В умовах швидкого розвитку технологій та зростання складності IT інфраструктур, однією з ключових вимог для ефективного функціонування є автоматизація процесів конфігурування та

управління. Проте, на сучасному ринку існує велика кількість програмних інструментів, призначених для цих цілей, і вибір найбільш оптимального та ефективного інструменту може бути викликом для організацій.

Однією з центральних проблем є недостатня увага до аналізу та порівняння функціональних можливостей різних програмних інструментів для автоматизації конфігурування та управління ІТ інфраструктурою. Наявність широкого спектру інструментів ускладнює вибір оптимального рішення, а недостатня обґрунтованість вибору може призвести до невідповідності потребам організації, перевищення бюджету або зниження ефективності використання інфраструктури.

Також важливою проблемою є неоднорідність стандартів і підходів до розробки програмних інструментів для автоматизації ІТ інфраструктури, що ускладнює їхню інтеграцію та використання у комплексі з іншими технологіями.

Тому, для забезпечення ефективного та оптимального використання програмних інструментів для автоматизації функцій конфігурування і управління в ІТ інфраструктурах, необхідно провести аналіз доступних інструментів, визначити їхні переваги та недоліки, а також врахувати специфіку конкретної організації та її потреби.

Аналіз останніх досліджень та публікацій

В останні роки значно зросла увага до інструментів для автоматизації функцій конфігурування та управління в ІТ інфраструктурах. Основною метою цих інструментів є спрощення та оптимізація процесів управління ІТ-ресурсами, що включає моніторинг, налаштування, розгортання та вирішення проблем у ІТ-системах.

У зазначеній статті [1] розглянуті розповсюджені складнощі у керуванні ІТ-проєктами, включаючи часті зміни та недостатньо документовані вимоги, проблеми комунікації між різними учасниками проекту, вплив ІТ-інфраструктури на безпеку та управління даними, ускладнення узгодженої роботи різноманітних ІТ-компонентів, а також виклики, пов’язані із соціальним відокремленням через перехід до режиму віддаленої роботи.

Дослідники [2] вивчають вимоги, цілі та проблеми щодо планів цифрової трансформації та побудови сучасних ІТ-інфраструктур компаній, таких як гіперконвергентна, дезагрегована гіперконвергентна та комбінована. Цифрова трансформація – це не просто модне слово. Керівники високого рівня здійснюють пошук шляхів, щоб зробити інфраструктуру більш гнучкою та чутливою, зокрема розроблення стратегії цифрової трансформації ІТ-інфраструктури шляхом модернізації. Модернізація ІТ-інфраструктури може включати стандартне апаратне або програмне забезпечення гіперконвергентної інфраструктури як частину плану локальних, хмарних, гібридних багатохмарних або периферійних обчислень. Іншим варіантом, на думку дослідників є дезагрегована гіперконвергенція, також HCI 2.0, подібний до стандартної гіперконвергенції. Такий підхід – найновіша парадигма в еволюції конвергентного центру опрацювання даних, компонування ІТ інфраструктури.

У статті [3] розглядаються особливості інфраструктури як коду (IaC) як нового засобу керування інфраструктурою, аналізують . інструменти та фреймворки, які допомагають установам використовувати цю практику. Дослідники вважають аналогією інфраструктури як коду – набір креслень для будівництва будинку. Подібно до того, як креслення використовуються для керівництва побудовою фізичної структури, IaC використовується для формування та управління ІТ-інфраструктурою організації. Подібно до того, як креслення визначають матеріали, розміри та план будинку, сценарії IAC визначають ресурси, конфігурації та залежності ІТ-інфраструктури. Подібно до того, як будівельник використовує креслення, щоб побудувати будинок, ІТ-фахівці використовують IAC для надання та керування ресурсами послідовним і автоматизованим способом.

У статті [4] висувається концепція керування якістю ІТ-послуг у ІТ-інфраструктурі, яка ґрунтується на абстрагуванні реальних компонентів ІТ-інфраструктури та їх перетворенні у логічні об’єкти управління.

У роботі [5] розглядається використання методів багатокритеріальної оптимізації для створення сучасної бізнес-моделі управління ІТ на підприємстві, особливо в контексті переходу до сервісно-орієнтованої інформаційної економіки. Дослідники [6] аналізують проблеми удосконалення ІТ-інфраструктури, що сприятиме зростанню ефективності процедур управління підприємством та розробленню інноваційної архітектури ІТ інфраструктури. Дослідники розкривають загальні підходи до формування системної архітектури; ІТ-сервісів для вирішення задач підприємства. Запропоновано логічну модель ІТ-інфраструктури підприємства. Дослідники [7] відзначають, що Забезпечення інфраструктури включає всі необхідні компоненти для ІТ взагалі і охоплює необхідні технічні операції у ІТ-відділі, включаючи загальнодоступні телекомунікаційні мережі, Інтернет та телекомунікаційне обладнання, а також технічні засоби та послуги (апаратне та програмне забезпечення) для користувачів.

Згідно з останніми дослідженнями, використання таких інструментів сприяє підвищенню гнучкості ІТ-інфраструктур та їх здатності швидко адаптуватися до змін у бізнес-вимогах. Вони також сприяють кращій інтеграції різних ІТ-систем, що є критично важливим для сучасних бізнес-моделей, які вимагають швидкої реакції на ринкові зміни. Враховуючи вищевказане, можна зробити висновок, що інструменти для автоматизації функцій конфігурування та управління в ІТ інфраструктурах відіграють ключову роль у підтримці стабільності, безпеки та адаптивності сучасних ІТ-систем. Вони є невід'ємною частиною ефективного управління ІТ-інфраструктурою та забезпечення її відповідності до сучасних бізнес-вимог.

Формулювання цілі статті

Метою дослідження є аналіз переваг та недоліків використання методики IaC для автоматизації функцій конфігурування та управління в корпоративних ІТ інфраструктурах.

Для досягнення поставленої мети необхідно було виконати наступні завдання:

1) Порівняти декларативний та імперативний підходи та програмні інструменти Ansible, Terraform, ARM, CloudFormation, які використовуються для реалізації методики IaC.

2) Визначити критерії оцінювання ефективності використання методики IaC для процесів автоматизації конфігурування та управління в ІТ інфраструктурах.

Виклад основного матеріалу

Обґрунтування отриманих результатів дослідження

В сучасному високотехнологічному світі актуальність автоматизації процесів управління ІТ інфраструктурою зростає надвисокими темпами. Фахівці, що опікуються корпоративними ІТ інфраструктурами стикаються з рядом концептуальних викликів, таких як високі темпи розвитку, постійні зміни в царині зростаючих вимог та обмеженість ресурсів, а також прагнення збільшення їх ефективності. У цьому контексті використання інструментів автоматизації, до прикладу таких програмних продуктів, як Ansible, CloudFormation та Terraform, стає критично важливим.

Актуальність теми дослідження визначається сучасними тенденціями та викликами в галузі розробки, тестування, впровадження та підтримки програмного забезпечення, які вимагають високої швидкості, якості, надійності, безпеки, масштабованості та гнучкості процесів автоматизації конфігурування та управління ІТ інфраструктурою. Важливість вирішення зазначених вище завдань обумовлює необхідність аналізу та оцінки переваг, недоліків, ризиків та викликів застосування програмних інструментів в цих процесах, а також розробки рекомендацій щодо їх оптимізації та підвищення ефективності.

Інноваційність дослідження полягає і в тому, що в процесі його виконання був проведений комплексний порівняльний аналіз таких програмних інструментів як Ansible, ARM, ClaudFormation та Terraform функціонал яких передбачає автоматизовану реалізацію функцій конфігурування та управління ІТ інфраструктурою. Автори здійснили спробу системно подати ряд нових оригінальних

візій важливої наукової проблеми вдосконалення процесів автоматизації управління корпоративними IT інфраструктурами.

Основними аспектами цього дослідження є ефективність використання інструментів автоматизації, масштабованість, надійність та здатність до адаптації з врахуванням швидкозмінних вимог до корпоративних IT середовищ. Узагальненою метою проведення наукового дослідження є розробка рекомендацій та формування кращих практик в сфері автоматизації процесів вдосконалення IT інфраструктур, що покликане сприяти підвищенню їх ефективності та надійності.

Об'єктом дослідження є процеси конфігурування та управління в корпоративних IT інфраструктурах.

Предметом дослідження є методика, яка у фахових колах іменується "Інфраструктура як код" (IaC, Infrastructure as Code) і щораз частіше використовується для автоматизації зазначених процесів.

IaC – це практика управління та розгортання IT інфраструктури засобами програмного коду, на заміну ручних процесів, або інтерактивних інструментів. Реалізація методики IaC дозволяє використовувати однакові підходи та інструменти, аналогічні тим, що використовуються для розробки програмного забезпечення, формування, внесення змін та реалізації процедур чергового видалення застарілої версії IT інфраструктури. Наведемо далеко не повний перелік засобів (IaC) для створення і управління IT інфраструктурою: Terraform, AWS CloudFormation, Ansible, Chef, Puppet, Azure Resource Manager (ARM) Templates, SaltStack, Nomad, Juju, Pulumi, Rancher.

Порівняння підходів та технологій, що реалізують методику IaC

Наведемо означення ряду базових понять та термінів, які використовуються у даній статті.

Методологія (грец. μεθοδολογία — вчення про метод) — сукупність прийомів дослідження, що застосовуються в науці; вчення про методи пізнання та перетворення дійсності [8]. Ме́тод або мето́да (від грец. μέθοδος – "шлях крізь") – систематизована сукупність кроків, які потрібно здійснити, щоб виконати певну задачу чи досягти певної мети; поняття тотожне алгоритму дій і технологічному процесу[9]. Методика – сукупність взаємопов'язаних способів і прийомів доцільного й ефективного проведення певної роботи [10]. Засіб – те, що служить знаряддям у якій-небудь дії, справі або спеціальна дія, що дає можливість здійснити що-небудь, досягти чогось[8]. Спосіб – певна дія, прийом або система прийомів, яка дає можливість зробити, здійснити що-небудь, досягти чогось [8].

Декларативний підхід

Декларативний підхід в реалізації методики IaC полягає в тому, що попередньо проводиться опис бажаного стану IT інфраструктури, а не формування конкретних кроків для його досягнення. При цьому використовується високорівнева мова, що дозволяє задавати параметри, обмеження та залежності між ресурсами. В подальшому цей опис передається відповідному інструменту IaC, який визначає набір дій, які належить виконати, щоб привести IT інфраструктуру до бажаного стану. При цьому не слід контролювати хід процесу виконання, а лише його результат.

Суттєвими перевагами декларативного підходу є зокрема те, що при його використанні фахівцям зазвичай легше читати та розуміти код, оскільки він фокусується на діях, які необхідно виконати, а не способах їх реалізації; легше підтримувати та змінювати код, оскільки він не залежить від послідовності його виконання та не містить деталей реалізації; легше забезпечити ідемпотентність та консистентність IT інфраструктури, оскільки інструмент IaC самостійно вирівнює фактичний та бажаний стани; легше інтегрувати з іншими інструментами та платформами, оскільки код є універсальним та абстрактним.

Зазначимо ряд недоліків декларативного підходу, зокрема: він вимагає більш високого рівня абстракції та знань мови IaC, оскільки не відображає реальні процеси та операції; є менш ефективним при виконанні завдань, пов'язаних з маніпулюванням структурами даних та продуктивністю, оскільки не дозволяє використовувати низькорівневу оптимізацію та налаштування; може призво-

дити до неочікуваних та небажаних змін IT інфраструктури, якщо код містить помилки, або не враховує всі можливі сценарії розвитку подій.

Імперативний підхід

Імперативний підхід в реалізації методики IaC полягає в тому, що при його реалізації описується послідовність кроків, які належить виконати для досягнення бажаного стану IT інфраструктури. При цьому використовується низькорівнева мова, що дозволяє задавати команди, умови, цикли та змінні. В подальшому виконується код, з обов'язковим контролем процесу виконання та перевіркою результату.

До переваг імперативного підходу відносять: наявність можливості точніше контролювати стан та поведінку IT інфраструктури, оскільки він відображає реальні процеси та операції; легше розуміння програмістами-початківцями, оскільки за конструкціями він схожий на конструкції традиційних мов програмування; універсальність підходу, оскільки дозволяє використовувати функції та можливості мови IaC; вища ефективність при виконанні деяких типів завдань, включаючи маніпуляції зі структурами даних та оптимізації продуктивності.

Недоліки імперативного підходу: не так легко масштабується на великих проєктах, оскільки вимагає більш об'ємного коду та деталей реалізації; складніше проводити налагоджування через більшу схильність до помилок та несумісностей; не гарантує ідемпотентності та консистентності інфраструктури, оскільки він залежить від порядку виконання інструкції та фактичного стану IT інфраструктури; може бути несумісним з іншими інструментами та платформами, оскільки має специфіку для мови IaC.

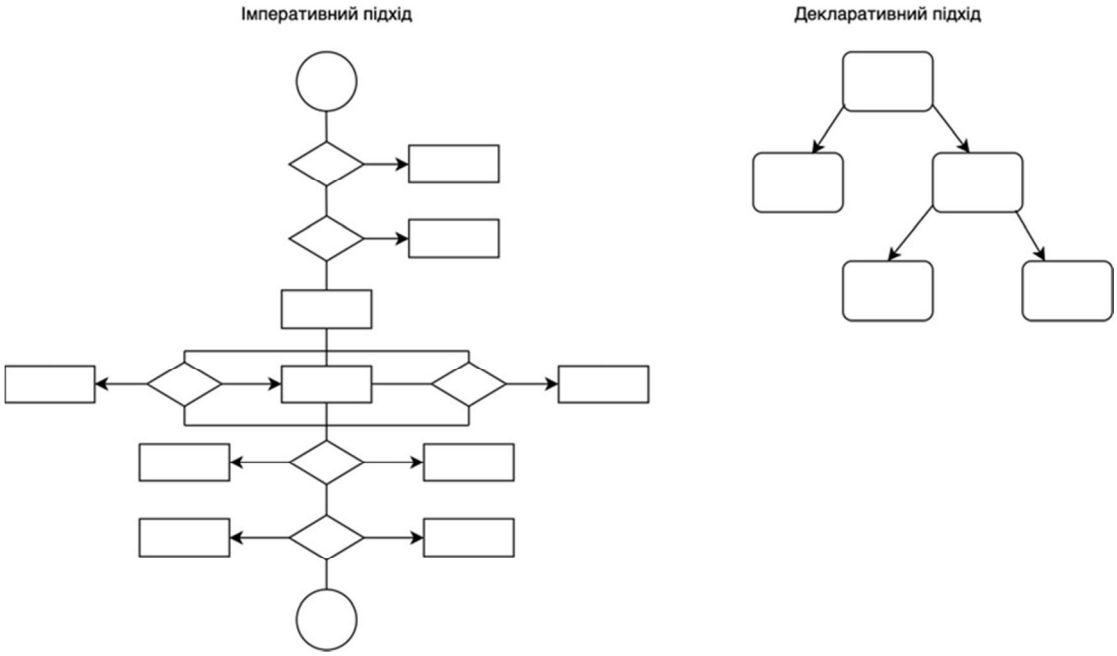

Рис. 1. Структурне подання імперативного та декларативного підходів в реалізаціях методики IaC

На рисунку зображені імперативний та декларативний підходи до програмування чи конфігурації систем.

Імперативний підхід характеризується тим, що програміст вказує, яким чином досягнути певного результату крок за кроком. Тобто, у імперативному підході програміст зосереджується на описі послідовності команд або інструкцій, які потрібно виконати, щоб отримати бажаний результат. Наприклад, в коді може бути зазначено кожен крок для створення об'єкту чи виконання операції.

Декларативний підхід, навпаки, описує бажаний стан системи чи результат, не вказуючи як це потрібно зробити. Замість того, щоб детально описувати послідовність дій, програміст вказує, який стан чи результат він хоче отримати, і система самостійно вирішує, як досягнути цього результату. Наприклад, в декларативному підході можна описати потрібний конфігураційний файл без зазначення кроків для його створення чи зміни.

Ключова різниця між цими підходами полягає в тому, що імперативний підхід складається з послідовності конкретних команд для досягнення мети, тоді як декларативний підхід сконцентрований на описі бажаного результату, не вказуючи як саме його досягти. У результаті декларативний підхід зазвичай вважається більш декларативним, елегантним та менш залежним від конкретних реалізацій.

Проведено порівняння двох базових підходів до впровадження методики IaC, а саме декларативного та імперативного, що реалізуються інструментами Ansible, Terraform, ARM, CloudFormation.

Програмні засоби реалізації методики IaC – інструменти Terraform і ARM

Terraform – є одним з найпопулярніших інструментів IaC, що реалізує методику IaC з використанням декларативного підходу. Terraform використовує власну мову HCL (HashiCorp Configuration Language), яка дозволяє описувати в одному коді ресурси різних провайдерів, таких як Azure, AWS, Google Cloud тощо. Terraform зберігає відомості про стан IT інфраструктури в файлі, що дозволяє визначати різницю між фактичним та бажаним станами та вносити необхідні зміни. В інструменті Terraform також підтримуються модулі, змінні та функції, що полегшує повторне використання та параметризацію коду.

ARM (Azure Resource Manager) – інструмент, що реалізує методику IaC, з використанням імперативного підходу. ARM використовує мову JSON (JavaScript Object Notation), що дозволяє описувати ресурси Azure в термінах шаблонів. ARM виконує шаблони в заданому порядку, викликаючи API Azure для створення, зміни або видалення ресурсів. ARM також підтримує змінні, параметри, функції та залежності, що дозволяють налаштовувати та умовно виконувати код.

Для порівняння інструментів Terraform і ARM використаємо наступні критерії: сумісність з різними провайдерами інфраструктур; легкість у вивченні та використанні мови IaC.; можливість контролю та налаштовування процесу виконання коду; швидкість та надійність виконання коду; підтримка модульності та повторного використання коду; підтримка процесів співпраці та версійної ідентифікації коду.

Використання в процесі аналізу наведених критеріїв, дозволило сформувати наступні висновки:

Terraform має перевагу над ARM у сумісності з продуктами різних провайдерів інфраструктури, оскільки забезпечує взаємодію більш як з 200 провайдерами, тоді як ARM працює тільки з Azure; ARM має суттєву перевагу над Terraform у легкості вивчення та використання мови IaC, оскільки JSON є більш поширеною та стандартною мовою, аніж HCL, а також має в своєму складі більше інструментів та обширнішу документацію; ARM має перевагу над Terraform у можливостях контролю та налаштування процесів виконання коду, оскільки дозволяє використовувати умови, цикли, змінні та функції, тоді як Terraform виконує код в автоматичному режимі, враховуючи при цьому залежності між ресурсами; Terraform має перевагу над ARM у швидкості та надійності виконання коду, оскільки зберігає відомості про стан інфраструктури в файлі, що дозволяє визначати різницю між фактичним та бажаним станами та вносити виключно необхідні зміни, тоді як ARM виконує кожного разу код повністю, що може призводити до помилок та затримок; Terraform має перевагу над ARM у підтримці модульності та повторного використання коду, оскільки дозволяє використовувати модулі, що є незалежними блоками коду, які можуть бути використані в різних проєктах, тоді як ARM використовує шаблони, що є цілісними файлами коду, які можуть використовуватись тільки в одному проєкті; Terraform має перевагу над ARM у підтримці співпраці та підтримці послідовних версій коду, оскільки дозволяє використовувати системи конт-

ролю версій, такі як Git, для зберігання, відстеження та обміну кодом, тоді як ARM для реалізації зазначеної функції вимагає використання спеціальних сервісів Azure, таких як Azure DevOps [12].

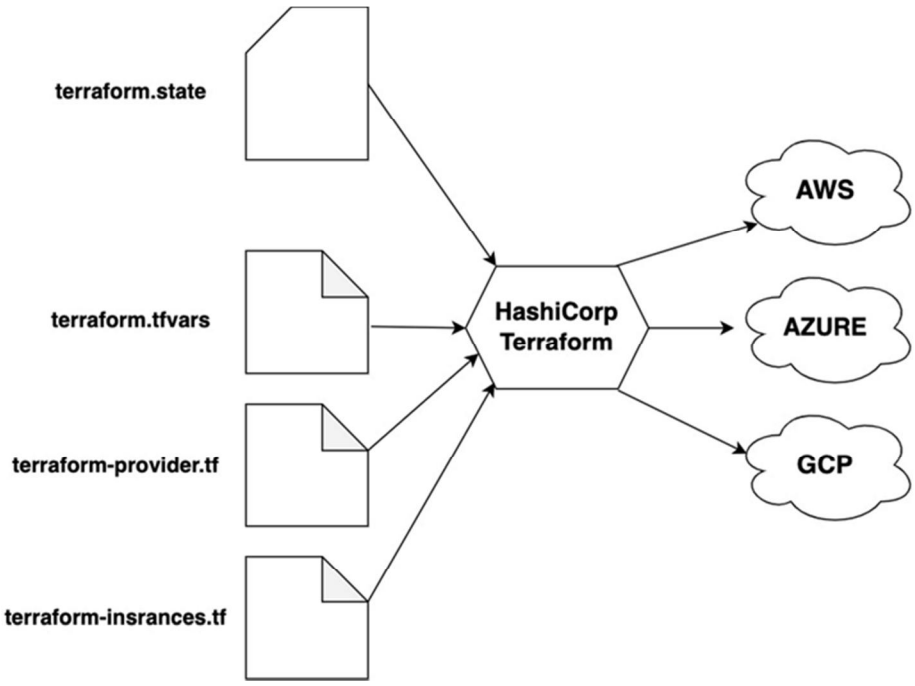

Рис. 2 Схема основних компонентів Terraform

Розробка та тестування конфігураційних файлів Terraform для управління IT інфраструктурою

Під час розробки конфігураційних файлів Terraform для управління IT інфраструктурою важливо враховувати кожен з аспектів та ресурсів, що будуть розгортатися. Також важливо тестувати ці конфігурації, щоб переконатися, що вони працюють на очікуваних платформах. Наведемо приклади коду для створення та тестування конфігураційних файлів в інструменті Terraform.

Приклад коду для створення IT інфраструктури у файлі main.tf:

```
# Визначення провайдера AWS
provider "aws" {
region = "us-east-1"
}

# Створення EC2 інстансу
resource "aws_instance" "example" {
ami           = "ami-0c55b159cbfafe1f0"
instance_type = "t2.micro"
}
```

У наведеному прикладі створюється інстанс EC2 в AWS us-east-1.

Приклад коду для тестування конфігураційного файлу за допомогою Terraform і Terratest у файлі main_test.go:

```
package test

import (
        "testing"
        "github.com/gruntwork-io/terratest/modules/terraform"
```

```
    "github.com/stretchr/testify/assert"
)

func TestTerraformEC2Creation(t *testing.T) {
    terraformOptions := &terraform.Options{
    TerraformDir: "../",
    }
    defer terraform.Destroy(t, terraformOptions)
    terraform.InitAndApply(t, terraformOptions)
    instanceID := terraform.Output(t, terraformOptions, "instance_id")
    assert.NotEmpty(t, instanceID)
}
```

У наведеному прикладі використовується бібліотека Terratest для написання тестів з метою перевірки, чи був створений інстанс EC2 після застосування конфігураційних файлів Terraform. Наведені приклади коду сприяють кращому розумінню процесів розроблення та тестування конфігураційних файлів Terraform для управління IT інфраструктурою.

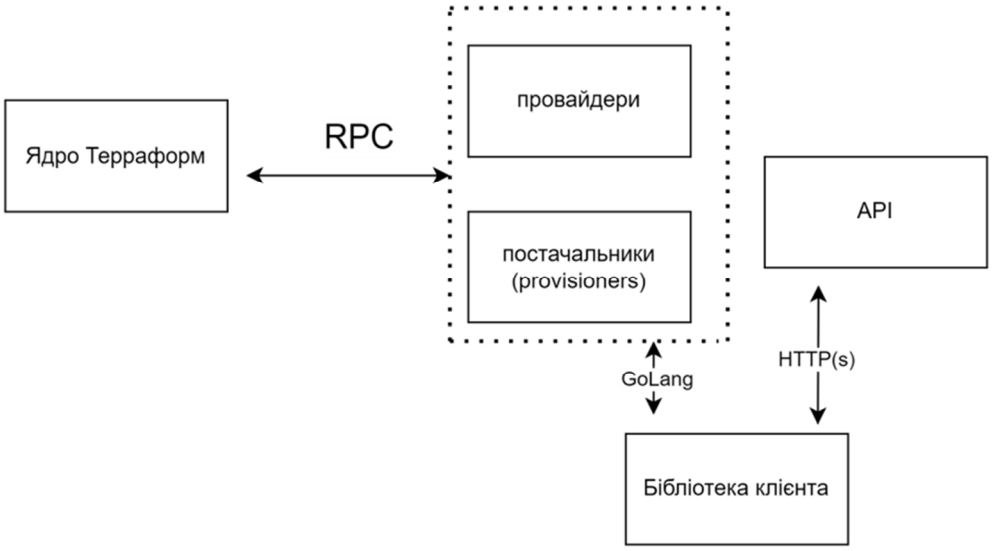

Рис. 3. Архітектура програмного інструменту Terraform.

Наведемо приклад шаблону ARM (Azure Resource Manager), який можна використовувати для створення віртуальної машини в Azure. Цей шаблон створює віртуальну машину з базовими налаштуваннями.

```
{
 "$schema": "https://schema.management.azure.com/schemas/2019-04-
   01/deploymentTemplate.json#",
 "contentVersion": "1.0.0.0",
 "parameters": {
  "vmName": {
   "type": "string",
   "metadata": {
    "description": "Ім'я віртуальної машини."
   }
  },
```

```
    "vmSize": {
      "type": "string",
      "defaultValue": "Standard_DS1_v2",
      "allowedValues": [
        "Standard_DS1_v2",
        "Standard_DS2_v2",
        "Standard_DS3_v2",
        "Standard_DS4_v2"
      ],
      "metadata": {
        "description": "Розмір віртуальної машини."
      }
    },
    "adminUsername": {
      "type": "string",
      "metadata": {
        "description": "Ім'я адміністратора."
      }
    },
    "adminPassword": {
      "type": "securestring",
      "metadata": {
        "description": "Пароль адміністратора."
      }
    }
  },
  "resources": [
    {
      "type": "Microsoft.Compute/virtualMachines",
      "apiVersion": "2020-12-01",
      "name": "[parameters('vmName')]",
      "location": "[resourceGroup().location]",
      "dependsOn": [],
      "properties": {
        "hardwareProfile": {
          "vmSize": "[parameters('vmSize')]"
        },
        "osProfile": {
          "computerName": "[parameters('vmName')]",
          "adminUsername": "[parameters('adminUsername')]",
          "adminPassword": "[parameters('adminPassword')]"
        },
        "storageProfile": {
          "imageReference": {
            "publisher": "MicrosoftWindowsServer",
            "offer": "WindowsServer",
            "sku": "2019-Datacenter",
            "version": "latest"
          },
```

```
            "osDisk": {
              "createOption": "FromImage"
            }
          },
          "networkProfile": {
            "networkInterfaces": [
              {
                "id":                                    "[resourceId('Microsoft.Network/networkInterfaces',
        concat(parameters('vmName'), '-nic'))]"
              }
            ]
          }
        }
      }
    ],
    "outputs": {}
}
```

Інструменти AWS CloudFormation в реалізаціях методики IaC

AWS CloudFormation – засіб, з допомогою якого можна подавати IT інфраструктуру як код (Infrastructure as Code – IaC)[13, 15] та надає можливість автоматизованого створення та управління IT інфраструктурою в хмарному середовищі Amazon Web Services (AWS) (Розробники Amazon Web Services. Перший випуск – 25.02.2011) [11]. Основною метою AWS CloudFormation є забезпечення ефективного та повторюваного створення ресурсів AWS, дозволяючи при цьому визначити IT інфраструктуру як код та управляти нею у вигляді конфігураційних файлів.

Базові поняття та терміни інструментарію AWS CloudFormation:

Подамо означення ряду базових понять та термінів, які використовуються в документації на інструменти AWS CloudFormation: шаблон CloudFormation JSON або YAML файл, який містить опис IT інфраструктури та ресурсів, які повинні бути створені в середовищі AWS. Шаблон визначає конфігурацію всієї IT інфраструктури у вигляді коду; ресурси – це основні конструктивні блоки IT інфраструктури, які описуються в шаблоні. Ресурс може бути, наприклад, екземпляром EC2, базою даних RDS, або групою Auto Scaling; стек – це набір ресурсів, які створюються та управляються як єдине ціле. Ресурси в стеку визначають взаємозв'язки та порядок їх створення; параметри – це значення, які можна вказати при запуску стеку для настроювання шаблону. Це в свою чергу дозволяє створювати більш гнучкі та параметризовані стеки; властивості – атрибути та параметри, які визначають поведінку ресурсів в шаблоні. Властивості визначають конфігураційні параметри ресурсів; статус стеку – інформація про те, чи стек успішно створений, чи відбулася помилка в процесі розгортання. Статус стеку може бути "CREATE_COMPLETE", "ROLLBACK_IN_PROGRESS", тощо; макроси – розширення можливостей CloudFormation за допомогою власних або вбудованих макросів. Це дозволяє автоматично виконувати певні дії або замінювати частини шаблону; регіон – конкретна географічна область, в якій створюються та розгортуються ресурси. CloudFormation підтримує роботу в багатьох регіонах, де забезпечується функціонування AWS; динамічні посилання – спеціальні функції в шаблоні, які дозволяють отримувати значення під час розгортання, наприклад, з виведення іншого ресурсу чи властивості стеку.

AWS CloudFormation надає розширений набір функціональних можливостей для автоматизації процесів створення та управління IT інфраструктурою в хмарному середовищі AWS, дозволяє визначати IT інфраструктуру у вигляді коду, використовуючи JSON або YAML файли, що іменуються шаблонами. Це робить процес створення та управління ресурсами більш прозорим і керо-

ваним. Завдяки AWS CloudFormation можна автоматизувати процес створення та розгортання IT інфраструктури [17]. Це реалізується за допомогою завантаження шаблону в консоль AWS або через командний рядок, що забезпечує консистентність та ефективність.

AWS CloudFormation дозволяє легко масштабувати IT інфраструктуру "вгору" або "вниз" за допомогою параметрів та шаблонів, що забезпечує пристосування розмірів IT інфраструктури з врахуванням можливих змін в навантаженні. Можливість формування послідовності версій шаблонів та стеків дозволяє керувати розвитком та версіями IT інфраструктури, забезпечуючи можливість відкатів до попередніх версій [18]. Реалізація використання параметрів (змінних), які дозволяють змінювати конфігураційні значення при створенні або оновленні стеку. Це робить шаблони більш гнучкими та забезпечує можливість повторного їх використання. В AWS CloudFormation реалізована можливість відстеження подій та моніторингу стану стеків засобами AWS Management Console та рядом інших інструментів, включених до складу AWS.

В інструменті підтримуються різні типи ресурсів, включаючи обчислювальні засоби, мережеві складові, бази даних та сервіси AWS. AWS CloudFormation дозволяє розгортати ресурси в різних регіонах, надаючи можливість створення розподіленої IT інфраструктури для забезпечення широкої доступності. CloudFormation інтегрується з іншими сервісами AWS, такими як AWS Identity and Access Management (IAM), AWS Lambda, AWS CloudTrail, що суттєво розширює його функціональність [19, 20].

Сфери використання AWS CloudFormation:

Веб-застосунки. Створення та управління IT інфраструктурою для веб-застосунків, включаючи сервери, бази даних, CDN та інші компоненти. CloudFormation дозволяє створювати резервні копії, масштабувати ресурси та оновлювати IT інфраструктуру під час наступних релізів.

Аналіз даних: визначення та розгортання IT інфраструктури для опрацювання та аналізу великих обсягів даних, використовуючи послуги Amazon EMR, Amazon Redshift і Amazon S3.

DevOps інфраструктура [14]. Автоматизація розгортання DevOps інфраструктури, включаючи створення тестових середовищ, CI/CD пайплайнів, інтеграцію зі засобами тестування та моніторингу[15].

Багаторівнева інфраструктура. Створення складних багаторівневих застосунків, що включають мережі, бази даних, сервери та інші ресурси.

Інфраструктура для мікросервісів. Використання AWS CloudFormation для автоматизації створення та керування інфраструктурою при формуванні архітектури мікросервісів, в якій кожен сервіс може бути поданий окремим стеком.

Системи моніторингу та логування. Розгортання та налаштування IT інфраструктури для систем моніторингу та логування, використовуючи такі сервіси, як Amazon CloudWatch та Amazon CloudTrail.

Демонстрації та тестування. Використання CloudFormation для швидкого розгортання IT інфраструктури з метою проведення демонстрацій, навчання або тестування, з можливістю швидкого відтворення динамічних сценаріїв.

Відновлення після відмов. Забезпечення можливості швидкого відновлення IT інфраструктури після відмови або аварії, що скорочує час відновлення та підвищує надійність системи.

Робота з архітектурними шаблонами. Використання CloudFormation для визначення та розгортання IT інфраструктури згідно з архітектурними шаблонами AWS, що включають в себе рекомендації та найкращі практики.

Наведемо приклад коду CloudFormation для створення EC2 віртуальних машин в AWS Cloud:

```
AWSTemplateFormatVersion: "2010-09-09"
Mappings:
  RegionMap:
    us-east-1:
      AMI: "ami-0ff8a91507f77f867"
```

```
    us-west-1:
      AMI: "ami-0bdb828fd58c52235"
    us-west-2:
      AMI: "ami-a0cfeed8"
    eu-west-1:
      AMI: "ami-047bb4163c506cd98"
    sa-east-1:
      AMI: "ami-07b14488da8ea02a0"
    ap-southeast-1:
      AMI: "ami-08569b978cc4dfa10"
    ap-southeast-2:
      AMI: "ami-09b42976632b27e9b"
    ap-northeast-1:
      AMI: "ami-06cd52961ce9f0d85"
Parameters:
  EnvType:
    Description: Environment type.
    Default: test
    Type: String
    AllowedValues: [prod, dev, test]
    ConstraintDescription: must specify prod, dev, or test.
Conditions:
  CreateProdResources: !Equals [!Ref EnvType, prod]
  CreateDevResources: !Equals [!Ref EnvType, "dev"]

Resources:
  EC2Instance:
    Type: "AWS::EC2::Instance"
    Properties:
      ImageId: !FindInMap [RegionMap, !Ref "AWS::Region", AMI]
      InstanceType: !If [CreateProdResources, c1.xlarge, !If [CreateDevResources, m1.large,
     m1.small]]
  MountPoint:
    Type: "AWS::EC2::VolumeAttachment"
    Condition: CreateProdResources
    Properties:
      InstanceId: !Ref EC2Instance
      VolumeId: !Ref NewVolume
      Device: /dev/sdh
  NewVolume:
    Type: "AWS::EC2::Volume"
    Condition: CreateProdResources
    Properties:
      Size: 100
 AvailabilityZone: !GetAtt EC2Instance.AvailabilityZone
```

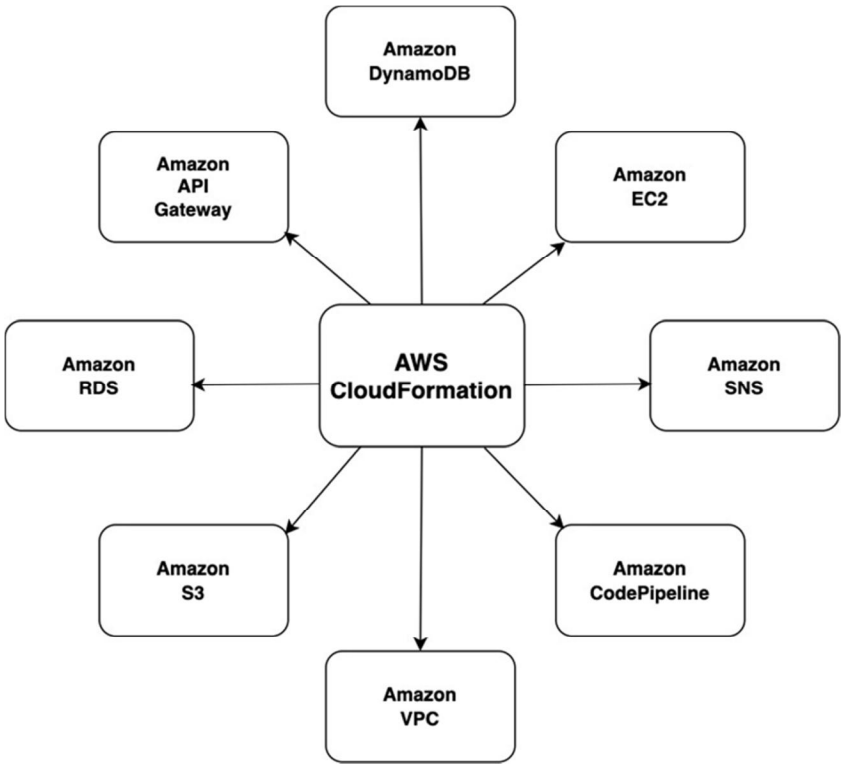

Рис. 4. Схема взаємодії модулів AWS
з програмним інструментом CloudFormation

На даному рисунку зображено взаємодію AWS Cloudformation з іншими сервісами AWS такими як RDS (служба керування реляційними базами даних), API Gateway (служба керування інтерфейсами програмування додатків), AWS DynamoDB (служба керування нереляційними базами даних NoSQL), AWS EC2 (сервіс обчислювальних ресурсів у хмарному середовищі AWS), AWS SNS (сервіс, який дозволяє створювати та керувати темами повідомлень та розсилати повідомлення до різних типів отримувачів), AWS CodePipeline (сервіс, який дозволяє створювати та автоматизувати пайплайни для постачання програмного забезпечення в хмарному середовищі AWS), AWS VPC (служба, яка дозволяє створювати та налаштовувати власні віртуальні мережі в хмарному середовищі Amazon Web Services), AWS S3 (хмарне сховище даних).

Конкурентні переваги AWS CloudFormation

Глибока інтеграція з AWS. Оскільки AWS CloudFormation є одним з сервісів Amazon Web Services, він глибоко інтегрований з іншими сервісами AWS, що забезпечує його ефективне функціонування в інфраструктурі AWS.

Масштабованість. Інструмент CloudFormation легко масштабується під роботи зі складними конфігураціями та великими проєктами, забезпечуючи ефективне управління IT інфраструктурою на будь-якому рівні складності.

Активна спільнота та ресурси. AWS CloudFormation має чисельну та активну спільноту користувачів та опирається на велику кількість найрізноманітніших ресурсів, таких як шаблони та реальні ситуативні приклади, що сприяє їх швидкому вивченню та вдосконаленню.

Конкурентні недоліки AWS CloudFormation

Складність освоєння. Запуск та управління інструментом CloudFormation може виявитись надскладним завданням для новачків, особливо при використанні ними складних та обширних шаблонів [17].

Можливість затримок в оновленнях. Оновлення та нові функції можуть вводитися з певними затримками в порівнянні з іншими інструментами управління IT інфраструктурою.

Інші інструменти з легшим вивченням. Конкуренти можуть пропонувати інструменти, які вважаються більш прийнятними для вивчення та використання, особливо для менших за чисельністю команд та при реалізації менш складних проєктів.

Відсутність інтеграції зі сторонніми хмарами. AWS CloudFormation не підтримує інтеграцію з іншими популярними хмарами, такими як, Azure або Google Cloud Platform, що може виявлятись суттєвим недоліком для фірм та компаній, які використовують декілька постачальників хмарних рішень [18].

Загальна характеристика програмного інструменту Ansible

Ansible – інструмент, який використовується для автоматизації управління конфігурацією та оркестрування інформаційних систем (Автор – Michael DeHaan. Розробники – Red Hat. Перший випуск – 20.02.2012 р.)[20].

Ansible написаний мовою Python і розповсюджується з використанням ліцензії GPLv3. Ansible є класичним програмним інструментом, що використовується для управління конфігуруванням (Configuration Management) базовою відмінністю якого від інших інструментів IaC є саме його специфічне функціональне призначення. Підкреслимо, якщо основним призначення IaC інструментів, таких як Terraform, CloudFormation, є формування IT інфраструктури і в другу чергу реалізація функції управління існуючою IT інфраструктурою, то основним призначенням інструменту Ansible є управління вже існуючою інфраструктурою. Слід відзначити, що доволі часто дані інструменти використовуються для створення і управління IT інфраструктурою, а в окремих випадках використовуються в комбінованому підході. Наприклад, за допомогою Ansible генеруються CloudFornation маніфести, які використовуються для створення необхідних ресурсів, після чого Ansible продовжує більш точне налаштування створених ресурсів [21].

Ansible використовує “Push” метод управління конфігураціями, іншими словами для використання Ansible не потрібно встановлювати агенти на серверах, якими потрібно керувати, достатньо встановити з’єднання з ними за допомогою протоколу SSH [22].

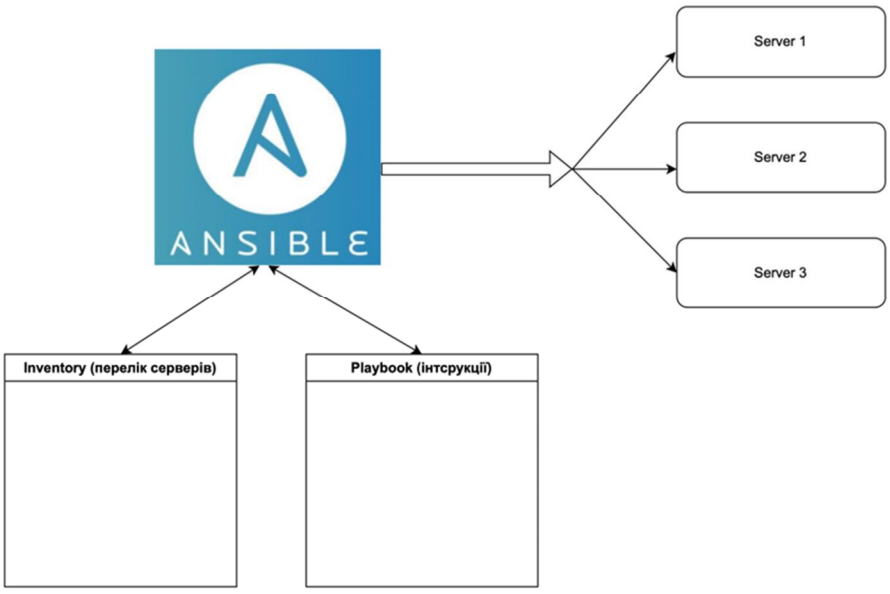

Рис. 5. Схема використання інструментів Ansible для управління конфігурацією серверів в IT інфраструктурі Наведемо ряд характеристик інструменту Ansible:

Декларативне конфігурування. Ansible використовує YAML-файли для опису бажаного стану системи. Можна визначити хмарні ресурси, такі як віртуальні машини, мережі, бази даних та інші, і

задати їхні параметри в конфігураційних файлах. Це робить процес конфігурування IT інфраструктури зрозумілим та керованим [23–25].

Модулі для хмарних платформ. Ansible містить в своєму складі модулі для роботи з різними хмарними платформами, такими як AWS, Azure, Google Cloud, і багатьма іншими. Ці модулі дозволяють взаємодіяти з API хмарами, створювати, масштабувати, або видаляти ресурси, використовуючи засоби Ansible.

Інтеграція з іншими інструментами. Ansible може інтегруватись з іншими інструментами та сервісами. Наприклад, можна сумісно використовувати Ansible з Terraform для визначення виду IT інфраструктури і при цьому задіяти Ansible для конфігурування ресурсів, що були створені з допомогою Terraform.

Динамічне управління файлами Inventory. Ansible може використовувати динамічні інвенторі файли, в яких зберігається перелік ресурсів, які знаходяться під управлінням, для автоматичного визначення у хмарі наявних ресурсів. Це означає, що є можливість додавати або видаляти ресурси в хмарі і при цьому Ansible автоматично врахує ці зміни при наступному запуску. Це є особливо корисним, у випадках коли під управлінням Ansible знаходяться одночасно сотні чи навіть тисячі задіяних ресурсів, що суттєво ускладнило б процедури внесення змін у inventory файли.

Захист конфіденційності даних. Ansible дозволяє шифрувати конфіденційні дані, такі як паролі чи ключі API, з метою гарантування безпеки в процесах автоматизації управління хмарною IT інфраструктурою.

Використання Jinja шаблонів. Інструмент дозволяє генерувати Kubernetes маніфести та CloudFormation шаблони з використанням умов, циклів, можливістю використання змінних, що значно підвищує гнучкість процесів конфігурування.

Використання колекцій Ansible Galaxy. Ansible Galaxy – онлайн-ресурс та командний інструмент для обміну, пошуку та ідентифікації ролей в Ansible, які використовуються для структуризації та повторного використання вже сформованої конфігурації та раніш вирішених задач. Ansible Galaxy дозволяє користувачам знаходити, спільно використовувати та вносити зміни в колекції ролей Ansible.

Загалом Ansible як і Terraform, на відміну від AWS CloudFormation, надає можливість абстрагування від конкретних деталей хмарних платформ, що дозволяє ефективно управляти IT інфраструктурою в різних типах хмар з використанням “універсального” програмного інструменту.

Наведемо приклад Ansible коду для створення EC2 віртуальних машин в AWS Cloud:

```
- name: Launching EC2 instances
  community.aws.ec2_instance:
    #aws_access_key: "{{ec2_access_key}}"
    #aws_secret_key: "{{ec2_secret_key}}"
    profile: "{{ aws_boto_profile }}"
    key_name: "{{ aws_demo_key }}"
    security_group: "{{ aws_security_group }}"
    instance_type: "{{ item.value.instance_type }}"
    image_id: "{{ aws_ami_id }}"
    state: present
    wait: yes
    wait_timeout: 300
    region: "{{ aws_region }}"
    tags:
      Name: "{{ item.value.name }}"
    detailed_monitoring: no
    vpc_subnet_id: "{{ vpc_subnet_list | random }}"
    network:
      assign_public_ip: yes
  loop: "{{ lookup('dict', ec2_new_list, wantlist=True) }}”
```

Переваги програмного інструменту Ansible

Перелічимо переваги, притаманні програмному засобу Ansible

Простота вивчення та використання. Ansible відомий як доволі простий інструмент, що легко піддається вивченню. Його конфігурування відбувається за допомогою синтаксису YAML, що робить його легко доступним при першому ознайомленні.

Агент-лесс архітектура. Ansible працює за принципом "агент-лесс", що означає відсутність необхідності встановлювати агенти на віддалених системах. Це спрощує процедуру його використання та підтримки.

Крос-платформність: Ansible підтримує крос-платформні операційні системи, що дозволяє управляти інфраструктурою з використанням систем різних типів.

Масштабованість. Інструмент Ansible легко масштабується для роботи з великими і складними інфраструктурами та масивом найрізноманітніших пристроїв.

Інтеграція з іншими інструментами. Ansible добре інтегрується з іншими інструментами і сервісами, такими як Jenkins, Docker, Kubernetes та ін.

Спільнота та ресурси. Ansible має активну спільноту користувачів, а також обширну базу знань і ресурсів, яка містить готові модулі та шаблони.

Недоліки, притаманні Ansible

Затримки у реальному часі. Ansible працює через протокол SSH та не надає можливостей для моніторингу та управління в режимі реального часу, що може бути критичним недоліком для деяких сценаріїв.

Швидкість виконання. У порівнянні з іншими інструментами автоматизації, Ansible в деяких сценаріях може бути менш продуктивним, особливо це стосується роботи у великих інфраструктурах.

Обмеження для динамічних конфігурацій. Для динамічного конфігурування, такого, як горизонтальне масштабування, Ansible може вимагати додаткових налаштувань.

Відсутність механізмів ведення історії змін. Ansible не надає вбудованого механізму для зберігання та відстеження історії змін конфігурування.

Недостатньо розвинені можливості моніторингу. В Ansible менше розвинені засоби моніторингу порівняно з деякими конкуруючими інструментами, що може бути критично важливим для сценаріїв з великою кількістю серверів.

Висновки

Сьогодення науково-виробничої галузі комп'ютингу характеризується стрімкими інформаційно-технологічними проривами, одним з яких є прорив пов'язаний з розвитком та масштабним втіленням системної методології DevOps. Зазначена методологія природнім чином реалізує чотири етапи життєвого циклу складних інформаційних систем, саме системного аналізу, системного проектування, системної інтеграції та системного адміністрування. Таке сприйняття та узагальнення об'єкту та предмету методології DevOps є надійно верифікованим впродовж всього періоду розвитку та чисельних втілень концепту "складна інформаційна система".

Автори статті проаналізували особливості методики IaC, яка є однією з важливих та актуальних в контексті впровадження методології DevOps в корпоративних IT інфраструктурах. Розглянуто особливості декларативного та імперативного підходів в реалізаціях методики IaC.

У роботі проведено порівняння чотирьох популярних інструментів, які використовуються для управління IT інфраструктурами в хмарах: Ansible, Terraform, CloudFormation та ARM. Проаналізовано особливості, переваги та недоліки Ansible, Terraform, CloudFormation та ARM, а також сценарії їх застосування. Сформовано та обґрунтовано висновок про те, що кожен з інструментів має індивідуальну нішу та специфіку при використанні. На даний час не існує універсального інструменту, який задовільняв би повний спектр потреб. Обирати інструмент рекомендується в залежності від конкретних потреб, цілей та вимог. Сформовано множину критеріїв, які використовуються в процесах прийняття відповідних рішень. Мова зокрема йде про: модульність та універсальність;

використання AWS чи Azure в якості основної хмарної платформи, гнучкість та контроль процесів автоматизованого конфігурування та управління в IT інфраструктурах. Подальші дослідження зосереджуватимуться на формалізації процедур вибору програмних інструментів для автоматизації функцій розгортання та управління IT інфраструктурами з використанням методики IaC. Передбачається розроблення методів, які базуватимуться на онтологічному підході до IaC та алгоритмах машинного навчання.

Список літератури

1. Трофименко О., Логінова Н. (2023). Аналіз проблем управління IT-проєктами / О. Трофименко, Н. Логінова Інформаційні управляючі системи і технології (ІУСТ-ОДЕСА-2023) : матеріали XI Міжнародної науково-практичної конференціі (21–23 вересень 2023 р. Одеса) / вип. ред. В.В. Вичужанін. 2023. С. 213–216. URI https://hdl.handle.net/11300/27080

2. James Alan Miller (2023). A guide to how digital transformation IT infrastructure works URL: https://www.google.com/search?q=Perspectives+and+Implications+for+the+Development+of+Information+Infrastructures&oq=Perspectives+and+Implications+for+the+Development+of+Information+Infrastructure

3. Ask Palo (2023). What is Infrastructure As Code? URL: https://blog.palo-it.com/en/ask-palo-what-is-infrastructure-as-code

4. Колеснік В. М., Ролік О. І. (2021). Підхід до управління якістю іт-послуг в іт-інфраструктурі на основі логічних об’єктів управління з потенційним часом звільнення Вісник Вінницького політехнічного інституту. № 1, С. 88–94 doi https://doi.org/10.31649/1997-9266-2021-154-1-88-94

5. Копійка О. В. (2018). Зміна бізнес-моделі управління IT на підприємстві у зв’язку з розвитком сервісно-орієнтованої інформаційної економіки / О. В. Копійка, А. Г. Кондратюк // Математичне моделювання в економіці. 2018. № 1. С. 56–66. – Режим доступу: http://nbuv.gov.ua/UJRN/mmve_2018_1_6.

6. Довгий С. О., Копійка О. В. (2017). Підвищення ефективності управління підприємством за рахунок трансформації IT-інфраструктури Математичне моделювання в економіці, , №. 1–2(8) [14 С. 7–17

7. Гармаш А. О., Бєлова Т. Г. (2021). Стратегії забезпечення інфраструктури IT компаній *XXVI International scientific and practical conference Topical issues of practice and science*,18–21 May, London, UK, 1-3

8. Яременко В., Сліпушко О. (2001). Академічний тлумачний словник української мови: В 4 т. Т. 2., К.: АКОНІТ.

9. Словник української мови. – https://sum20ua.com/?wordid=50848&page=1613&searchWord=%D0%BC%D0%B5%D1%82%D0%BE%D0%B4#lid_50848

10. Мельничук О. С. (1985). Словник іншомовних слів. Київ

11. Veselin Kantsev, Madhu Joshi and Kartikey Pandey (2017). Implementing DevOps on AWS Published by Packt Publishing Ltd.

12. Mitesh Soni (2017). Implementing DevOps with Microsoft Azure: Automate your deployments and incorporate the DevOps culture. Published by Packt Publishing Ltd.

13. Stephane Jourdan, Pierre Pomès (2017). Infrastructure as Code (IAC): Cookbook Published by Packt Publishing Ltd.

14. Jennifer Davis and Katherine Daniels (2017). Effective DevOps. Published by O’Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472

15. Gene Kim, Jez Humble, Patrick Debois and John Willis (2016). The devops handbook. IT Revolution Press

16. Yevgeniy Brikman (2022). Terraform: Up and Running Writing Infrastructure as Code. Sebastopol: O’Reilly Media 1005 Gravenstein Highway North.

17. Jez Humble and David Farley (2010). Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley Professional.

18. Michael T. Nygard (2017). Release It! Design and Deploy Production-Ready Software. The Pragmatic Bookshelf.

19. Jez Humble, Joanne Molesky and Barry O’Reilly (O’Reilly) (2015). Lean Enterprise. Oreilly & Associates Inc.

20. Bas Meijer, Lorin Hochstein & Rene Moser (2022). Ansible: Up and Running: Automating Configuration Management and Deployment the Easy Way, 2nd Edition,

21. Jeff Geerling (2015). Ansible for DevOps: Server and configuration management for humans.

22. Waqas Irtaza (2021). IT Infrastructure Automation Using Ansible: Guidelines to Automate the Network, Windows, Linux, and Cloud Administration.

23. Mohamed Alibi (2018). Ansible Quick Start Guide: Control and monitor infrastructures of any size, physical or virtual. Packt Publishing.

24. Karen Tovmasyan (2020). Mastering AWS CloudFormation: Plan, develop, and deploy your cloud infrastructure effectively using AWS CloudFormation. Packt Publishing

25. Christina Tucker (2023). Resolving CloudFormation Stack Creation Failure. Independently Published.

References

1. Trofymenko O., Loginova N. (2023). Analysis of IT project management problems / O. Trofymenko, N. Loginova Information control systems and technologies (IUST-ODESA-2023): materials of the 11th International Scientific and Practical Conference (September 21–23, 2023) Odesa) / issue ed. V.V. an outsider 2023. P. 213 – 216. URI https://hdl.handle.net/11300/27080

2. James Alan Miller (2023). A guide to how digital transformation IT infrastructure works URL: https://www.google.com/search?q=Perspectives+and+Implications+for+the+Development+of+Information+Infrastructures&oq=Perspectives+and+Implications+for+the+Development+of+Information+Infrastructure

3. Ask Palo (2023). What is Infrastructure As Code? URL: https://blog.palo-it.com/en/ask-palo-what-is-infrastructure-as-code

4. Kolesnik V. M., Rolik O. I. (2021). An approach to managing the quality of IT services in the IT infrastructure based on logical management objects with a potential release time Bulletin of the Vinnytsia Polytechnic Institute. No. 1, p. 88–94doi https://doi.org/10.31649/1997-9266-2021-154-1-88-94

5. Kopiyka O. V. (2018). Changing the business model of IT management at the enterprise in connection with the development of a service-oriented information economy / O. V. Kopiyka, A. G. Kondratyuk // Mathematical modeling in economics. 2018. No. 1. P. 56–66. – Access mode: http://nbuv.gov.ua/UJRN/mmve_2018_1_6.

6. Dovgyi S. O., Kopiyka O. V. (2017). Increasing the efficiency of enterprise management due to IT infrastructure transformation Mathematical modeling in economics, , no. 1–2(8) [14 P. 7–17

7. Garmash A. O., Belova T. G. (2021). Strategies for ensuring IT infrastructure of companies XXVI International scientific and practical conference Topical issues of practice and science, 18–21 May, London, UK, 1–3

8. Yaremenko V., Slipushko O. (2001). Academic explanatory dictionary of the Ukrainian language: Vol 4 (T.2), K.: AKONIT.

9. Dictionary of the Ukrainian language. – https://sum20ua.com/?wordid=50848&page=1613&searchWord=%D0%BC%D0%B5%D1%82%D0%BE%D0%B4#lid_50848

10. Dictionary of foreign words. Structure. O. S. Melnychuk, Kyiv,1985

11. Veselin Kantsev, Madhu Joshi and Kartikey Pandey (2017). Implementing DevOps on AWS Published by Packt Publishing Ltd.

12. Mitesh Soni (2017). Implementing DevOps with Microsoft Azure: Automate your deployments and incorporate the DevOps culture. Published by Packt Publishing Ltd.

13. Stephane Jourdan and Pierre Pom s (2017). Infrastructure as Code (IAC): Cookbook Published by Packt Publishing Ltd.

14. Jennifer Davis and Katherine Daniels (2017). Effective DevOps. Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472

15. Gene Kim, Jez Humble, Patrick Debois and John Willis (2016). The devops handbook. IT Revolution Press

16. Yevgeniy Brikman (2022). Terraform: Up and Running Writing Infrastructure as Code. Sebastopol: O'Reilly Media 1005 Gravenstein Highway North.

17. Jez Humble and David Farley (2010). Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley Professional.

18. Michael T. Nygard (2017). Release It! Design and Deploy Production-Ready Software. The Pragmatic Bookshelf.

19. Jez Humble, Joanne Molesky and Barry O'Reilly (O'Reilly) (2015). Lean Enterprise. Oreilly & Associates Inc.

20. Bas Meijer, Lorin Hochstein & Rene Moser (2022). Ansible: Up and Running: Automating Configuration Management and Deployment the Easy Way, 2nd Edition,

21. Jeff Geerling (2015). Ansible for DevOps: Server and configuration management for humans.

22. Waqas Irtaza (2021). IT Infrastructure Automation Using Ansible: Guidelines to Automate the Network, Windows, Linux, and Cloud Administration.
23. Mohamed Alibi (2018). Ansible Quick Start Guide: Control and monitor infrastructures of any size, physical or virtual. Packt Publishing.
24. Karen Tovmasyan (2020).. Mastering AWS CloudFormation: Plan, develop, and deploy your cloud infrastructure effectively using AWS CloudFormation. Packt Publishing
25. Christina Tucker (2023). Resolving CloudFormation Stack Creation Failure. Independently Published.

ANALYSIS OF SOFTWARE TOOLS FOR AUTOMATION OF CONFIGURATION AND MANAGEMENT FUNCTIONS IN IT INFRASTRUCTURES

Mykola Orlov[1], Yurii Dmytriv[2]

[1] Lviv Polytechnic National University, Information Systems and Networks Department, Lviv, Ukraine
[2] Lviv Polytechnic National University, Department of Artificial Intelligence Systems, Lviv, Ukraine
[1] E-mail: orlov.nv.86@gmail.com, ORCID: 0009-0007-9835-6177
[2] E-mail: ydmytriv@gmail.com, ORCID: 0009-0001-7427-5387

The work by the authors, using a systematic approach, analyzes a group of software tools that are functionally oriented towards the automated implementation of configuration and management processes in IT infrastructures. The research profile focuses on a methodology known in the professional environment as "Infrastructure as Code" (IaC) and is one of the foundational methodologies implemented in a systemic combination within the DevOps methodology. This methodology is actively used in processes of dynamic formation, deployment, and maintenance of corporate IT infrastructures in many modern successful high-tech companies to achieve the best business performance, efficiency, guaranteed success, and security. The article discusses two basic approaches to building software tools that implement the IaC methodology, namely the declarative and imperative approaches.

The main emphasis is placed on the formation of a set of advantages and disadvantages inherent in software tools such as Terraform, ARM, Ansible, and CloudFormation. The focus of researchers on these four software tools is explained by their leading positions in a fairly extensive lineup of possible alternative software products that allow for a comprehensive implementation of the IaC methodology in the context of full and functional systemic deployment of the DevOps methodology in specific implementations of corporate IT infrastructures. The authors' generalized conclusion of original scientific research is that there is currently no single clearly distinguished universal software tool among others that fully satisfies the entire spectrum of requirements and needs. Potential users in this context are communities of DevOps professionals and clients – owners and managers of modern dynamic high-tech and successful companies, firms, and businesses that rely on modern information systems and technologies.

Key words: software tools; DevOps methodology; IaC methodology; automation of configuration functions; IT infrastructure management; declarative approach; imperative approach.