**МІНІСТЕРСТВО ОБОРОНИ УКРАЇНИ**

**ВІЙСЬКОВИЙ ІНСТИТУТ ТЕЛЕКОМУНІКАЦІЙ ТА ІНФОРМАТИЗАЦІЇ ІМЕНІ ГЕРОЇВ КРУТ**

КАФЕДРА № 21

(Автоматизованих систем управління)

**НАУКОВА РОБОТА**

**ТЕМА: „МОДЕЛЬ ВИКОРИСТАННЯ CI/CD ПРОЦЕСІВ ПІД ЧАС РОЗГОРТАННЯ ІНФРАСТРУКТУРИ НА ОСНОВІ IAC”**

Розділ: «Інформаційні технології»

Виконала: курсант 205 навчальної групи

солд. \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ксендзук О.О

(підпис)

Керівник: викладач кафедри № 21,

м-р\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Балан Д.Д

(підпис)

Київ – 2024

Зміст

ВСТУП 3

РОЗДІЛ 1 Концепція *Continuous* *integration* (Ci) та *Continuous* *Delivery* (CD) 5

1.1 Поняття CI/CD 5

1.2. Безперевна інтеграція CI 6

1.3 Безперервна доставка/розгортання CD (*Delivery*/*Deployment*) 7

1.4 Порівняння різних *DevOps* платформ CI/CD 9

Висновки до розділу 1 11

Розділ 2 Інфраструктура як код (IaC) 12

2.1 Поняття та переваги використання IaC 12

2.2 *Terraform* 14

2.2.1 Провайдери *Terraform* 16

2.2.2 Каталоги та модулі 18

2.3 *Ansible* 19

Висновок до розділу 2 21

Розділ 3 Аналіз моделі використання CI/CD під час розгортання інфраструктури на основі IaC 23

3.1 Аналіз способів інтеграції CI/CD з інструментами інфраструктури як коду, такими як *Ansible*, *Terraform* 23

3.2 Практичні аспекти інтеграції CI/CD з IaC інструментами 24

3.2.1 Репозиторій коду та система контролю версій 25

3.2.2 Структура репозиторію та організація коду 26

3.2.3 Розгортання тимчасових середовищ 27

3.2.4 Універсальні вхідні параметри та секрети 29

3.2.5 Тестові набори та перевірка IaC коду 30

3.2.6 Артефакти конфігурацій 32

3.2.7. Схвалення змін та безпечне розгортання 33

3.2.8. Моніторинг та журналювання 33

3.2.9 Зворотні засоби (*rollback*, *rollforward*) 34

3.2.10 Гібридні середовища та різні хмарні провайдери 36

Висновки до розділу 3 37

Висновки 39

Список використаних джерел 40

# АНОТАЦІЯ

наукової роботи на тему:

„ Модель використання CI/CD процесів під час розгортання інфраструктури на основі IaC”

Актуальність дослідження пов’язана з необхідністю впровадження методів автоматизації для розгортання та управління інфраструктурою у Збройних силах України.

За останні роки інфраструктура в Збройних силах України стала значно складніша та розподілена і тенденція нарощування зберігатиметься та лише зростатиме. Військова діяльність вимагає високої стабільності і надійності інформаційних систем та мереж. Окрім того, під час служби нерідко виникає потреба у швидкому масштабуванні інфраструктури для забезпечення додаткових можливостей. CI/CD та IaC дозволяють автоматизовано масштабувати інфраструктуру з мінімальними затратами.

Процеси CI/CD допомагають автоматизувати тестування якості та забезпечує швидке виявлення та виправлення помилок збірки, що підвищує загальну надійність систем, тим самим забезпечуючи безпеку.

Мета роботи – підвищення ефективності, швидкості та надійності управління та розгортанням інформаційних систем та технологічних рішень, що використовуються в Збройних силах України.

Робота складається з трьох розділів. У ході дослідження було використано загальнонаукові методи, такі як: аналіз, синтез, порівняльний аналіз, імітаційне моделювання, тощо.

Основні положення конкурсної роботи оприлюднені на X науково-практичній конференції Військового інституту телекомунікацій та інформатизації

***Ключові слова:*** *DevOps, безперевна інтеграція, безперевна доставка/розгортання, конвеєр, Git, система контролю версій, Jenkins, GitLab, інфраструктура як код, Ansible, playbook, Terraform, контейнеризація, застосунок.*

# Перелік Умовних позначень

CI/CD – *Continuous Integration / Continuous Delivery, Deployment*

IaC – *Infrastructure as Code* (безперевна інтеграція/безперервна доставка)

DevOps – *Development and Operation*

API – *Application Programming Interface* (прикладний програмний інтерфейс)

SLA – *Service Level Agreement* (угода про рівень послуг)

HCL – *HashiCorp Configuration Language*

AWS – *Amazon Web Services*

RDS – *Relational Database Service*

GCP – *Google Cloud Platform*

SSL – *Secure Sockets Layer*

DNS – *Domain Name System*

SSH – *Secure Shell*

# ВсТУП

**Актуальність роботи.** Враховуючи стрімкий розвиток технологій та необхідність надійного функціонування інформаційних систем, дослідження цієї теми має важливе значення для подальшого розвитку для застосування у Збройних силах України.

У сучасних умовах безпеки та оборони України, ефективне використання передових технологій та методів стає критично важливим для забезпечення обороноздатності. Ця робота спрямована на дослідження практичного вдосконалення процесів розгортання інфраструктури та забезпечення безпеки інформаційних систем. Впровадження передових методів CI/CD та використання підходів IaC забезпечуватиме швидкість, ефективність і безпеку при впровадженні нових технологій та інформаційних систем.

**Метою дослідження** є опис оптимальної моделі використання процесів *Continuous Integration* (CI) та *Continuous Delivery* (CD) під час розгортання інфраструктури на основі *Infrastructure as Code*  (IaC) з метою підвищення ефективності, швидкості та надійності управління та розгортанням інформаційних систем у Збройних силах України

**Об’єктом дослідження** є процеси розгортання інфраструктури на основі *Infrastructure as Code*  (IaC)

**Предметом дослідження** є модель використання *Continuous Integration* (CI) та *Continuous Delivery* (CD) процесів під час розгортання інфраструктури на основі *Infrastructure as Code*  (IaC)

Для досягнення мети в роботі сформульовані та вирішені часткові взаємопов’язані завдання дослідження:

1. Аналіз шляхів підвищення ефективності функціонування.

2. Аналіз методів побудови.

3. Аналіз моделі.

**Практична цінність результатів роботи.**

Впровадження CI/CD процесів під час розгортання інфраструктури на основі IaC полягає у наступних перевагах:

1. Скорочення часу на розгортання та оновлення інфраструктури, підвищення швидкості доставки змін.

2. Підвищення стабільності та надійності інфраструктури за рахунок автоматизованого тестування змін.

3. Зниження ризиків помилок при ручному розгортанні за рахунок автоматизації процесів.

4. Покращення відслідковування змін та можливість повернення до попередніх конфігурацій.

5. Узгодженість конфігурацій в різних середовищах (*dev, staging, prod*).

6. Спрощення процесу управління гібридними та мультихмарними інфраструктурами.

7. Підвищення ефективності командної роботи та співпраці над інфраструктурним кодом.

# РОЗДІЛ 1

# Концепція *Continuous* *integration* (Ci) та *Continuous* *Delivery* (CD)

## 1.1 Поняття CI/CD

CI/CD – це неперервна інтеграція та неперервна доставка (або неперервне розгортання – *Continuous Integration/ Continuous Delivery* або *Deployment*), – це практика розробки певного продукту, яка включає такі ключові елементи:

1. Неперервна інтеграція: часте об’єднання змін коду у спільний репозиторій. Після кожної інтеграції автоматично запускаються тести для забезпечення якості коду та виявлення можливих проблем.

2. Неперервна доставка:  на основі неперервної інтеграції автоматизує процес підготовки коду до розгортання, щоб код завжди був готовий до випуску.

3. Неперервне розгортання: робить ще один крок, автоматично розгортаючи зміни коду у продакшн (тобто випуск) після успішного проходження тестів та перевірок якості.

Ці практики дозволяють командам розробників швидше, надійніше та з меншою кількістю помилок доставляти програмне забезпечення. Вони сприяють оптимізованому робочому процесу, що покращує якість коду та прискорює випуск оновлень. Інструменти CI/CD допомагають розробникам швидко збирати, тестувати та розгортати оновлення додатків через автоматизовані конвеєри. Замість ізольованих ручних процесів, CI/CD зв’язує всі етапи – від коміту до продакшну – в єдиний плавний робочий процес.

*DevOps* зосереджений на швидшій доставці ПЗ за рахунок інтеграції розробки та операцій. CI/CD – це важлива практика *DevOps,* яка автоматизує інтеграцію та доставку змін коду. *Continuous Integration* (CI) та *Continuous Delivery/Deployment* (CD) – це практики, які дозволяють автоматизувати процес збирання, тестування та випуску програмного забезпечення. Вони є ключовими елементами підходу *DevOps* та сприяють частішому та більш надійному релізу програмних продуктів.

Ця автоматизація прискорює цикли випуску, покращує якість за рахунок раннього виявлення дефектів та дозволяє надійні розгортання “одним натисканням”. Впроваджуючи CI/CD, команди *DevOps* можуть часто та надійно випускати оновлення.

## 1.2. Безперевна інтеграція CI

Безперервна інтеграція (*Continuous Integration*, CI) є ключовою практикою в процесі розробки програмного забезпечення, що допомагає підвищити якість коду та прискорити цикл релізів (випусків). Розгляньмо на прикладі класичної ситуації, коли розробники об’єднують зміни коду в спільний репозиторій протягом дня. Кожна зміна автоматично збирається і тестується, щоб швидко виявляти проблеми. Такий приклад насправді актуальний не лише для цивільної сфери, як може здаватися на перший погляд, оскільки для Збройних сил України також розробляються різні інформаційні системи, програмні застосунки та інші рішення, котрі після їх релізу (тобто випуску) також потребують підтримки, оновлень, особливою мірою певні безпекові рішення для цих продуктів.

Тобто *Continuous Integration* (CI) стосується практики регулярного збирання та тестування коду програми після кожного злиття у центральний репозиторій коду. Це допомагає виявляти конфлікти та помилки на ранніх стадіях циклу розробки. CI здійснюється через автоматизовані збірки та запуск тестових наборів, що дозволяє постійно перевіряти цілісність системи.

Отже, інтеграція це та частина роботи команди розробників котра стосується до збірки готового продукту (перед випуском у продакшн). Концепція CI полягає в регулярному (наприклад, після кожного коміту, тобто застосованих змін, до системи контролю версій) збиранні та автоматичному тестуванні всього проекту для раннього виявлення та виправлення помилок інтеграції. Основні кроки в процесі безперервної інтеграції:

1. Збирання коду (*compile/build*) програмний код збирається в деяку виконувану форму (двійкові файли, артефакти тощо).

2. Запуск юніт-тестів - виконуються автоматизовані тести низького рівня, що перевіряють окремі модулі системи.

3. Аналіз якості коду - статичний аналізатор коду перевіряє код на дотримання стандартів, відсутність вразливостей та антипатернів.

4. Збирання артефакту - створюється пакунок або образ, придатний для розгортання.

5. Запуск інтеграційних/функціональних тестів - більш комплексні тести перевіряють взаємодію компонентів системи.

6. Розгортання в тестове середовище - збірка, що пройшла всі перевірки, розгортається для додаткових ручних або автоматизованих тестів.

Усі ці моменти більш ретельно описуються у розділі 3.

Весь цей процес ініціюється автоматично при новому коміті і зазвичай виконується на окремому віртуальному або фізичному сервері CI. Популярні рішення: *Jenkins, GitLab CI/CD, Azure DevOps, CircleCI* та інші.

Основні переваги безперервної інтеграції: раннє виявлення та виправлення дефектів, підвищення якості програмного забезпечення, зменшення ризиків і витрат на інтеграцію, можливість частіших релізів, гнучкість та економія часу за рахунок автоматизації.

CI є невід'ємною практикою сучасних гнучких методологій розробки програмного забезпечення, таких як *Agile, DevOps* тощо.

## 1.3 Безперервна доставка/розгортання CD (*Delivery*/*Deployment*)

Безперервна доставка (*Continuous Delivery*):

CD (*Delivery*) розширює концепцію CI, автоматизуючи процес релізу програмного забезпечення за допомогою безперервного розгортання до тестових або проміжних середовищ. Основні кроки CD (*Delivery*):

1. Виконання процесу CI (збирання, тестування коду).

2. Створення пакунку або образу для розгортання.

3. Розгортання збірки у тестове або проміжне середовище.

4. Запуск додаткових тестів (навантажувальне тестування, тести користувацького інтерфейсу тощо).

5. Потенційне ручне схвалення для розгортання у виробниче середовище.

Ключова відмінність від CI - можливість легко вручну або автоматично перенести успішну збірку виробничого середовища після додаткових перевірок.

Безперервне розгортання (*Continuous Deployment*):

CD (*Deployment*) є менш обмежувальною версією безперервної доставки. Тут кожна збірка, що успішно пройшла через конвеєр CI/CD, автоматично розгортається у виробниче середовище без ручного схвалення. Це забезпечує максимальну швидкість релізів.

Конвеєр (*pipeline*) у контексті безперервної інтеграції та безперервної доставки - це автоматизована послідовність кроків, які виконуються з кодом з моменту внесення змін до моменту розгортання у виробниче середовище.

Конвеєр CI/CD описує весь робочий процес створення та релізу програмного забезпечення і може включати такі основні етапи:

1. Ініціація конвеєра - коміт коду в репозиторій або інша подія, що запускає конвеєр.

2. Збирання коду (*build*) - компіляція або пакування коду в придатну для розгортання форму.

3. Юніт-тестування - запуск автоматизованих модульних тестів.

4. Аналіз якості коду - статичний аналіз, перевірка стандартів, безпеки ін.

5. Створення артефактів - створення пакунків, образів для розгортання.

6. Тестування безпеки - сканування на вразливості, перевірки відповідності політиці безпеки.

7. Інші типи тестування - інтеграційне, регресійне, навантажувальне тощо.

8. Розгортання у тестове середовище - для подальших тестів та перевірок.

9. Затвердження для релізу - ручне або автоматичне схвалення переходу до продакшн.

10. Розгортання у виробниче середовище.

11. Моніторинг - спостереження за продуктивністю та станом розгорнутого застосунку.

Конвеєри зазвичай конфігуруються як код в спеціальних файлах із визначенням усіх етапів, середовищ, інструментів. Популярні інструменти: *Jenkins, GitLab CI/CD, Azure Pipelines, AWS CodePipeline* тощо.

Ключові переваги конвеєрів: автоматизація процесу доставки, виявлення помилок на ранніх стадіях, підвищення швидкості та надійності релізів, відтворюваність процесу, прозорість та аудит процесу, скорочення витрат на розгортання.

Отже, ключові відмінності CD (*Delivery*) від *Continuous Deployment*: при CD потрібне ручне схвалення для розгортання у виробниче середовище, при *Continuous* *Deployment* розгортання у виробниче середовище відбувається автоматично

Обидві практики CD і *Continuous Deployment* мають на меті прискорити цикл релізів, але CD додає додатковий контрольний захід безпеки у вигляді ручного затвердження перед виробничим розгортанням.

Ключові переваги CD (*Delivery*) та *Continuous Deployment*:

- швидший цикл релізів та доставка;

- зниження ризиків через автоматизацію та безперервне тестування;

- зменшення витрат на релізи програмного забезпечення;

- можливість невеликих, але частих випусків замість великих релізів.

Обидві практики є складовими *DevOps* підходу і повинні впроваджуватися разом з відповідними культурними та організаційними змінами.

## 1.4 Порівняння різних *DevOps* платформ CI/CD

Розглянемо деякі з найпопулярніших систем CI/CD:

*Jenkins* – це одна з найстаріших і найпопулярніших систем CI/CD з відкритим кодом, підтримує величезну кількість плагінів для інтеграції з різними інструментами та середовищами. Конфігурування через вебінтерфейс або за допомогою кодування конвеєрів. Переваги: велика спільнота, гнучкість, безкоштовний. Недоліки: не завжди зручний вебінтерфейс, вимагає додаткового налаштування та обслуговування.

*GitLab* CI/CD – позиціонує себе як повноцінна *DevOps* платформа, це вбудована система CI/CD від *GitLab* для інтеграції з їх хмарною або локальною *Git*-системою. Її особливості: конфігурація через YAML-файли в репозиторії, має підтримку контейнерів *Docker* і *Kubernetes*.

Переваги: інтегрована з *GitLab,* простота налаштування, візуалізація конвеєрів. Недоліки: менше гнучкості та розширюваності порівняно з *Jenkins.*

*Azure DevOps* - рішення від Microsoft для CI/CD та управління життєвим циклом додатків, включає *Git*-репозиторії, системи будівництва та релізів, тестові інструменти. Хмарна служба, але також можливий локальний сервер.

Переваги: інтеграція з *Azure* і *Microsoft*, хмарний сервіс, безкоштовний для невеликих команд. Недоліки: не відкрите джерельне ПЗ, деякі платні функції.

AWS *CodePipeline* - сервіс безперервної доставки від AWS для автоматизації релізів, інтегрується з іншими сервісами AWS (*CodeBuild, CodeDeploy, ECR, ECS* тощо), налаштування конвеєрів через веб-консоль або AWS CLI. Переваги: використання переваг AWS екосистеми, масштабованість, автоматичні оновлення. Недоліки: не має вбудованих інструментів для збирання та тестування, платна послуга.

*CircleCI* - хмарна платформа CI/CD з акцентом на швидкість та продуктивність, підтримує *Docker*, інтеграцію з хмарними провайдерами, її конфігурація конвеєрів через YAML-файли. Переваги: проста в налаштуванні, масштабована, гнучка. Недоліки: повністю хмарний сервіс з платними тарифами.

Загалом, кожна система має свої сильні та слабкі сторони. Вибір залежить від потреб проєкту, наявного досвіду команди, вимог до масштабованості, бюджету та інтеграції з іншими інструментами *DevOps.*

## Висновки до розділу 1

CI та CD є ключовими практиками *DevOps*, які дозволяють автоматизувати процес збирання, тестування та доставки програмного забезпечення, забезпечуючи більш часті та надійні релізи.

Безперервна інтеграція зосереджена на регулярному збиранні та тестуванні коду після кожного коміту. Це допомагає раннє виявляти та виправляти помилки та конфлікти на ранніх етапах циклу розробки.

Безперервна доставка розширює концепцію CI, автоматизуючи процес релізу програмного забезпечення шляхом розгортання успішних збірок у тестові або проміжні середовища.

Безперервне розгортання є крайньою формою CD, де кожна успішна збірка автоматично розгортається у виробниче середовище без ручного схвалення.

Ключовими перевагами CI/CD є: прискорення циклу релізів, раннє виявлення дефектів, підвищення якості ПЗ, зниження ризиків та витрат на розгортання.

Процеси CI/CD реалізуються через конвеєри (*pipeline*) - автоматизовані послідовності етапів від початкового коміту до фінального розгортання.

Існує низка популярних інструментів CI/CD, таких як *Jenkins, GitLab CI/CD, Azure DevOps, AWS CodePipeline* тощо. Вибір системи CI/CD залежить від потреб проекту, масштабованості, інтеграції з іншими інструментами, наявного досвіду команди та бюджету. Успішне впровадження CI/CD вимагає не лише технічних змін, а й культурних та організаційних трансформацій згідно з принципами *DevOps*. CI/CD підходи є невід'ємною складовою сучасного гнучкого процесу розробки програмного забезпечення і допомагають прискорити доставку цінності кінцевим користувачам.

Отже, безперервна інтеграція та безперервна доставка/розгортання є потужними практиками *DevOps*, які забезпечують ефективний та якісний цикл релізів програмного забезпечення при належному впровадженні.

# Розділ 2

# Інфраструктура як код (IaC)

## 2.1 Поняття та переваги використання IaC

Інфраструктура як код (*Infrastructure as Code*  - IaC) - це парадигма, в якій вся інфраструктура ІТ-систем (обчислювальні ресурси, мережі, сховища даних тощо) визначається як версійований машинозчитуваний код замість ручного налаштування ресурсів. Ця інфраструктурна конфігурація може бути відтворена в будь-якому середовищі з коду за допомогою автоматизованих інструментів.

Інфраструктура — це ресурси, які потрібні для підтримки коду. Водночас дехто може уявити серверні стійки, комутатори та купу заплутаних кабелів... Але це вчорашній день. Сьогодні 99% проєктів живе в «хмарах». Тобто ресурси — це віртуальні машини, контейнери, *load balancers*.

Отже, усі хмарні ресурси — це інше програмне забезпечення, яке виконується на комп’ютерах нашого хмарного провайдера.

*Infrastructure as Code*  — це спосіб постачання та керування обчислювальними та мережевими ресурсами методом їх опису у вигляді програмного коду, на відміну від налаштовування необхідного обладнання власноруч чи з допомогою інтерактивних інструментів.

*Infrastructure as Code*  є (вже не таким) новим трендом, який розв’язує актуальну проблему автоматизації інфраструктури.

У багатьох могла трапитися подібна ситуація із наступним діалогом:

— Мені треба задеплоїти лоад-балансер...  
— Вибач, у нас завал! Будь ласка, створи тікет у JIRA і повертайся пізніше.

Якби інфраструктура була автоматизована, цього діалогу б не відбулося (як і затримки в роботі), бо лоад-балансер автоматично деплоївся б. Тому автоматизація інфраструктури така популярна. Вона розв’язує не тільки технічні питання, а й організаційні та комунікаційні. Автоматизація полегшує наше життя і перетворює безлад на передбачуваний процес.

Основні принципи IaC:

- уся інфраструктура описується декларативним способом, як код;

- код інфраструктури зберігається у системі контролю версій;

- налаштування інфраструктури за допомогою інструментів автоматизації;

- ідемпотентність - неодноразове застосування визначення завжди призводить до одного й того ж результату.

Ключові переваги впровадження IaC:

1. Відтворюваність - можливість швидко й послідовно відтворити ідентичні середовища з коду.
2. Масштабованість - легко створювати та керувати великою кількістю ресурсів.
3. Стандартизація конфігурацій та врахування найкращих практик.
4. Покращення співпраці та ефективності командної роботи завдяки використанню коду як єдиного джерела істини.
5. Потенційне зниження помилок від людського фактору, автоматизація змін.
6. Швидкість та ефективність впровадження змін в інфраструктуру.
7. Безпечне відкочування змін завдяки використанню систем контролю версій.
8. Легша інтеграція з практиками *DevOps* та безперервної доставки.
9. Економія часу та ресурсів на підтримку інфраструктури.
10. Перенесення політик доступу та аудиту до коду.

Інструменти IaC:

* спеціалізовані скрипти: bash та ін.;
* засоби управління конфігурацією: *Puppet, Ansible* та ін.;
* засоби шаблонізації серверів: *Packer, Vagrant*;
* засоби оркестрації: *Kubernetes* та ін.;
* засоби ініціалізації ресурсів: *Terraform, CloudFormation* та ін.

IaC є невід'ємним компонентом сучасних хмарних середовищ, *DevOps* процесів та розробки масштабованих, надійних додатків. Популярними інструментами для реалізації IaC є *Ansible, Terraform, AWS CloudFormation, Azure Resource Manager* тощо.

Загалом, підхід IaC забезпечує гнучкість, стабільність, можливість повторного використання та більший контроль над ІТ-інфраструктурою через моделі розробки програмного забезпечення.

Проблеми, котрі вирішує IaC:

1) Проблема масштабу. В середньому, кожен мікросервіс потребує від 10 до 12 інфраструктурних ресурсів, таких як *load balancer*, екземпляр RDS, групи безпеки та інші. Якщо у нас є три різних середовища — тестове, підготовче та продуктивне — це вже означає близько 30 різних ресурсів. Проте, коли кількість мікросервісів збільшується до 10, 20 або навіть 100, проблема стає ще більш виразною та масштабною.

2) Проблема передбачуваності. Якщо створювати всі ці ресурси вручну, то питання «Що робити, якщо ми припустимося помилки і наші середовища будуть відрізнятися; до яких багів це може призвести?» перетворюється на «Що робити, коли...» Бо вірогідність припуститися помилки в кількох сотнях ручних операцій наближається до 100%. З урахуванням цих проблем автоматизація інфраструктури стає не просто модним трендом, а необхідністю.

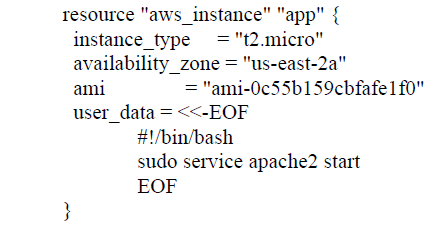
## 2.2 *Terraform*

Розглянемо інструмент *Terraform* для реалізації підходу "Інфраструктура як код". В контексті IaC, *Terraform* це засіб ініцілізації ресурсів. Засоби ініціалізації ресурсів, на відміну від інших інструментів IaC, створюють самі ресурси, тобто віртуальні машини, кластер *Kubernetes*, бази даних, мережі тощо.

За допомогою засобів ініціалізації ресурсів можна створювати: бази даних, кеші, балансувальники навантаження, системи моніторингу, налаштування підмереж і брандмауера, правила маршрутизації, сертифікати SSL і майже будь-який інший аспект вашої інфраструктури.

Наприклад, наступний код розгортає вебсервер за допомогою *Terraform*.

-*ami* – визначає ідентифікатор образу AMI, який потрібно розгорнути на сервері. -user\_data – bash-скрипт, що виконується за час завантаження вебсервера.

Рис. 2.2.1 Приклад *Terraform* коду для розгортання вебсервера

*Terraform* – це інструмент із відкритим вихідним кодом від компанії *HashiCorp*, написаний мовою програмування *Go.* Код на *Go* компілюється в єдиний двійковий файл з назвою *Terraform*.

Цей файл дає змогу розгорнути інфраструктуру прямо з вашого ноутбука (або будь-якого іншого комп'ютера) і для всього цього не потрібно ніякої додаткової інфраструктури. Все завдяки тому, що всередині виконуваний файл *Terraform* робить від вашого імені API-виклики до одного/декількох провайдерів: *AWS, Azure, Google Cloud, DigitalOcean, OpenStack*. Особливості *Terraform*:

1. Підтримка численних провайдерів. *Terraform* підтримує широкий спектр хмарних провайдерів (*AWS, Azure, GCP* та ін.), а також приватні віртуальні та фізичні середовища через власні чи сторонні провайдери.

2. Декларативний синтаксис. Мова конфігурації *Terraform* є декларативною, що означає, що вона описує бажаний кінцевий стан вашої інфраструктури. Конфігурація описується декларативною мовою HCL (*HashiCorp Configuration Language*). Користувач визначає бажаний стан інфраструктури, а *Terraform* визначає необхідні дії для досягнення цього стану (і постійно це контролює).

3. Графова модель залежностей. *Terraform* будує графову модель усіх ресурсів та їхніх залежностей для створення відповідного плану виконання.

4. Ідемпотентність - неодноразове застосування однакової конфігурації буде призводити до того самого результату.

Переваги використання *Terraform*: портативність між різними хмарними провайдерами, зменшення руйнівних помилок завдяки планам виконання, повторне використання конфігурацій як коду, безпечне гранулярне додавання та відкочування змін, можливість тестування та модульної структури конфігурацій, інтеграція з іншими інструментами та практиками *DevOps*.

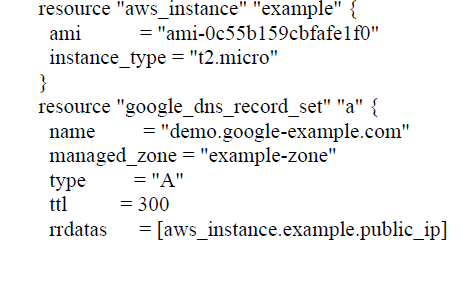
*Terraform* став популярним завдяки простоті синтаксису, гнучкості підтримуваних провайдерів та можливості створювати переносні визначення інфраструктури. Він активно використовується в контексті *DevOps* та безперервної доставки для впровадження практики IaC.

В наступних підрозділах буде описана певні базові технічні деталі про те як працює *Terraform*.

## 2.2.1 Провайдери *Terraform*

Провайдер (*provider*) у контексті *Terraform* - це компонент або плагін, який дозволяє *Terraform* взаємодіяти з певним типом ресурсів або сервісами. Провайдери абстрагують специфічні API різних постачальників хмарних послуг, систем керування конфігураціями тощо за єдиним інтерфейсом *Terraform*. Вони надають набори ресурсів, які можна використовувати у конфігураційних файлах.

Ось приклад конфігурації *Terraform* звідки *Terraform* дізнається, які API-виклики йому потрібно робити:

Рис. 2.2.1.1 Приклад провайдера

Цей фрагмент змушує *Terraform* виконати API-виклики до двох провайдерів: до AWS, щоб розгорнути там сервер, і до *Google Cloud*, щоб створити DNS-запис, який вказує на IP-адресу сервера з AWS.

Що роблять продайдери? Кожен провайдер додає набір типів ресурсів (resource types) та/або джерел даних (data sources), якими може керувати *Terraform*.

*Terraform* повинен оголосити, які провайдери йому потрібні, щоб *Terraform* міг їх встановити і використовувати.

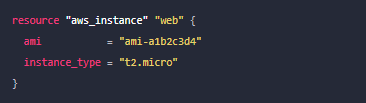
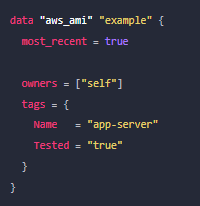
*Resorce type* та *Data sources*:

Рис. 2.2.1.2 *Resorce type* та *Data sources*

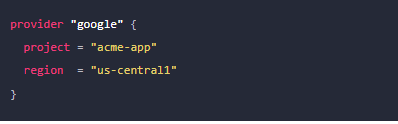
Кожен провайдер має два ідентифікатори:

Рис. 2.2.1.4 Ідентифікатори провайдера

1) Унікальна адреса джерела, яка використовується лише тоді, коли потрібен постачальник.

2) Локальна назва, яка використовується всюди в модулі *Terraform*.

Конфігурації провайдерів налаштовуються в кореневому модулі конфігурації *Terraform*. (Дочірні модулі отримують свої конфігурації провайдера від кореневого модуля.) Конфігурація провайдера створюється за допомогою блоку *provider*:

Рис. 2.2.1.5 Конфігурація провайдера 

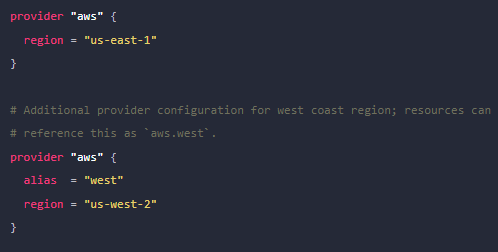
*Alias* - для конфігурації кількох постачальників:

Рис. 2.2.1.6 Конфігурація кількох провайдерів

## 2.2.2 Каталоги та модулі

Модулі - це контейнери для упакування конфігурацій *Terraform* для повторного використання. Вони дозволяють інкапсулювати групу ресурсів та їхні залежності в єдиний пакет коду. Модулі можуть бути локальними або дистанційними. [Модуль](https://developer.hashicorp.com/terraform/language/modules) - це набір файлів .tf та/або .tf.json, що зберігаються разом у каталозі. Кореневий модуль *Terraform* складається з конфігураційних файлів верхнього рівня в каталозі, з якого запускається *Terraform*, та дочірніх модулів.

Переваги використання модулів:

* повторне використання коду та абстракція деталей реалізації;
* організація та структурування великих конфігурацій;
* можливість версіонування та оновлення модулів;
* заохочення модульного та компонентного підходу до розробки інфраструктури.

Модулі можуть бути викликані та параметризовані з основних конфігурацій, забезпечуючи гнучкість і повторне використання. Модулі викликаються з інших модулів за допомогою блоків *module*.

Каталоги (*Registries*) *Terraform* - це центральні сховища для публікації та розповсюдження модулів *Terraform*. Вони забезпечують легкий доступ до офіційних та сертифікованих модулів від *HashiCorp* та спільноти.

Основні каталоги: *Terraform* *Registry* - офіційний каталог від *HashiCorp* та приватні/внутрішні каталоги - власні сховища організацій

Переваги використання каталогів:

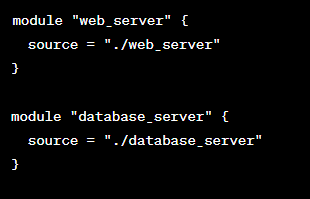
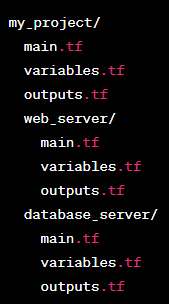
* Спрощений пошук, перегляд та встановлення модулів
* Перевірка якості та безпеки офіційних модулів
* Централізоване управління залежностями модулів
* Можливість використання приватних модулів організацій

Рис. 2.2.2.1 Приклад структури каталогів для конфігурації *Terraform*

Рис. 2.2.2.2 Приклад файлу конфігурації, *my\_project/main.tf*

Комбінування модулів та каталогів сприяє повторному використанню, структуризації та стандартизації кодової бази *Terraform* інфраструктури, забезпечуючи ефективність роботи та відповідність кращим практикам.

## 2.3 *Ansible*

Розглянемо інструмент *Ansible* для реалізації підходу "Інфраструктура як код". *Ansible* - це потужна open-source платформа для автоматизації налаштування віддаленими системами та дозволяє описувати бажану конфігурацію систем у вигляді простих текстових інструкцій, називаних плейбуками (*playbooks*) та контролює її базовий стан.

Базове середовище *Ansible* складається з трьох основних компонентів:

1) Вузол управління (*Control node*) - система, на якій встановлено *Ansible*. Ви запускаєте команди такі як *Ansible* або *Ansible*-*inventory* на вузлі керування.

2) Керований вузол (*Managed node*) - віддалена система або хост, яким керує *Ansible*.

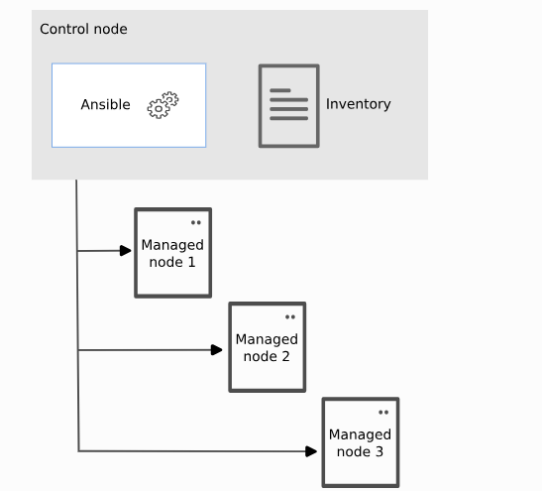
3) Інвентар (*Inventory*) - Список керованих вузлів, які логічно організовані. Ви створюєте інвентар на контрольному вузлі, щоб описати розгортання хосту для *Ansible*.

Рис. 2.3.1 Основні компоненти *Ansible*

Основні принципи роботи *Ansible*:

1. *Ansible* не вимагає встановлення додаткових агентів на керованих вузлах, використовуючи SSH або Windows *Remote Management* для підключення.

2. Ідемпотентність: *Ansible* забезпечує застосування одних і тих самих *playbooks* (файл, котрий написаний на YAML та містить набір інструкцій, що визначають конфігурацію та політики управління для групи хостів або серверів) завжди призведе до однакового результату.

3. Модулі: *Ansible* використовує величезну бібліотеку модулів для виконання різноманітних завдань, від управління пакетами до налаштування хмарних сервісів.

4. Інвентар (файл *Inventory*): Керовані вузли описуються в інвентарних файлах у форматах INI, YAML або через динамічні сценарії.

5. *Playbooks*: Конфігурації задаються у вигляді YAML *playbooks*, які є наборами завдань для вузлів із заданими параметрами.

Ключові переваги *Ansible*: простий у вивченні та використанні завдяки декларативній природі та зрозумілому синтаксису, не потребує встановлення клієнтів-агентів на цільові машини, використовує SSH, конфігурації пишуться на YAML, має портал з готовими конфігураціями [*Ansible* *Galaxy*](https://galaxy.ansible.com/), графічна оболонка [UI *Ansible* *Tower*/AWX](https://docs.ansible.com/ansible-tower/latest/html/userguide/main_menu.html), інтеграція з інструментами оркестрації, контейнеризації та CI/CD. *Ansible* активно використовується для:

* налаштування та оркестрації конфігурації серверів, робочих станцій, контейнерів;
* автоматичного розгортання та оновлення застосунків;
* провізії та управління обчислювальними ресурсами у хмарі;
* налаштування мережевого обладнання - маршрутизаторів, комутаторів;
* забезпечення контролю та відстеження конфігурацій (*configuration management*);
* реалізації практик безперервної інтеграції та доставки.

*Ansible* є потужним IaC інструментом завдяки своїй простоті, універсальності та здатності наблизити інфраструктуру до моделей розробки програмного забезпечення із ідемпотентністю, модульністю та версіями *config*.

## Висновок до розділу 2

Інфраструктура як код (IaC) - це парадигма, в якій вся інфраструктура ІТ-середовища визначається за допомогою кодових інструкцій та машиночитаних файлів конфігурацій замість ручного налаштування обладнання та середовищ. Це підхід, при якому інфраструктура розглядається та керується так само, як і код програмного забезпечення. Основна ідея IaC полягає в тому, щоб забезпечити стандартизацію, узгодженість та автоматизацію процесів розгортання, налаштування та управління інфраструктурними компонентами, такими як віртуальні машини, мережі, сховища даних, балансувальники навантаження та інші ресурси. Використовуючи IaC, адміністратори та *DevOps* інженери можуть визначати та керувати всією інфраструктурою за допомогою коду, який може бути збережений у системі контролю версій, модифікований, протестований та автоматично розгорнутий на різних середовищах. Це забезпечує відтворюваність, масштабованість та стабільність інфраструктури, а також полегшує співпрацю, аудит та внесення змін.

*Ansible* є потужним інструментом для автоматизації та управління конфігураціями різних систем та додатків. Він використовує декларативний підхід, що дозволяє описувати бажаний стан систем, а не послідовність дій для досягнення цього стану. *Ansible* характеризується легкістю у вивченні та використанні, agentless архітектурою та можливістю паралельного виконання завдань на багатьох вузлах.

*Terraform* є інструментом для створення, змін та створення версій інфраструктури безпечним та ефективним способом. Він забезпечує узгоджений *workflow* для керування хмарними провайдерами та послугами за допомогою єдиного інструменту. Ключові переваги *Terraform* включають підтримку багатьох провайдерів, декларативний синтаксис, залежності ресурсів, паралельне створення ресурсів.

*Ansible* та *Terraform* мають різні цілі та області застосування, але можуть успішно використовуватися разом у CI/CD процесах. *Ansible* зосереджений на управлінні конфігураціями та розгортанні додатків, тоді як *Terraform* призначений для керування хмарною інфраструктурою та ресурсами різних провайдерів. Використання підходу Інфраструктури як коду та інструментів, таких як *Ansible* та *Terraform*, забезпечує ефективне, надійне та масштабоване управління інфраструктурою та конфігураціями. Це дозволяє підвищити швидкість розгортання, мінімізувати помилки, забезпечити відтворюваність та полегшити співпрацю між різними командами в рамках *DevOps* практик.

# Розділ 3 Аналіз моделі використання CI/CD під час розгортання інфраструктури на основі IaC

## 3.1 Аналіз способів інтеграції CI/CD з інструментами інфраструктури як коду, такими як *Ansible*, *Terraform*

Розглянемо способи інтеграції практик безперервної інтеграції та безперервної доставки (CI/CD) з інструментами інфраструктури як коду (IaC), такими як *Ansible* та *Terraform*. Інтеграція *Ansible* з CI/CD:

1. Виконання *Ansible* *playbook* у CI процесі: *playbook* можна запускати як один з етапів CI конвеєра, це дозволяє автоматично тестувати та розгортати конфігурації на тимчасових або тестових вузлах

2. Моделювання інфраструктури за допомогою *Ansible* для тестування у CD: *Ansible* дозволяє розгорнути всю інфраструктуру в тимчасовому середовищі, в цьому середовищі можна виконувати розгалужене тестування перед релізом.

3. *Ansible* *Tower* або AWX для оркестрації CD процесів - ці інструменти забезпечують вебінтерфейс для керування *Ansible*, їх можна використовувати як центральний вузол для запуску та синхронізації CD конвеєрів.

4. Інтеграція *Ansible* з CI/CD системами (*Jenkins*, *GitLab* CI/CD, *CircleCI*): через забудовані плагіни або написання спеціальних етапів конвеєрів, це спрощує запуск та керування *Ansible* завданнями в межах CI/CD процесів.

Інтеграція *Terraform* з CI/CD:

1. Виконання планування та застосування змін *Terraform* в CI, команда: *terraform* *plan* генерує план змін після кожного коміту, команда *terraform* *apply* здійснює зміни на етапі CD після перевірок

2. Тестування *Terraform* конфігурацій: юніт-тести (*sentctl*, *Terraform*-*validator* та ін.) для перевірки коректності синтаксису, тести інтеграції для розгортання тимчасових середовищ.

3. Управління stage змінами та робочими просторами: робочі простори для розділення різних середовищ, *Remote state backends* для відстеження стану.

4. Конвеєри із застосуванням практик безпечного розгортання: плани змін вимагають схвалення перед застосуванням, використання інструментів *blue/green* або *canary* *Deployment*

5. Інтеграція з CI/CD інструментами через специфічні провайдери: провайдери для різних платформ: *Kubernetes*, AWS, *Azure* та ін, написання власних етапів *Terraform* у CI/CD конвеєрах

Обидва інструменти, *Ansible* та *Terraform*, добре інтегруються з підходами CI/CD через різноманітні сценарії використання та можливість включення їх у автоматизовані конвеєри. Це забезпечує постійну перевірку та надійне розгортання інфраструктурного коду згідно з принципами *DevOps*.

## 3.2 Практичні аспекти інтеграції CI/CD з IaC інструментами

Розглянемо деякі практичні аспекти інтеграції інструментів інфраструктури як коду (IaC) з процесами безперервної інтеграції та безперервної доставки (CI/CD).

Але перед порядком необхідних дій та кроків реалізації та описом цих технічних аспектів необхідно підготуватися до цієї роботи, а власне встановити *Git*, *Ansible*, *Terraform*. Тож спершу про те що потрібно зробити:

1) Спершу потрібно здійснити установку *git*.

2) Встановлення *Terraform* в залежності від операційної системи, все дуже швидко та просто знаходиться в документації

3) Встановлення *Ansible*.

Увага! *Ansible* має певні вимоги до вузла керування та керованого вузла, а саме: для вашого контрольного вузла (машини, на якій працює *Ansible*) ви можете використовувати майже будь-яку UNIX-подібну машину зінстальованим *Python* 3.8 або новішою, а для керованого вузла (машина, якою керує *Ansible*) не потребує встановлення *Ansible*, але вимагає *Python* 2.7 або *Python* 3.5 - 3.11 для запуску згенерованого *Ansible* коду *Python*. Керованому вузлу також потрібен обліковий запис користувача, який може підключатися через SSH до вузла з інтерактивною оболонкою POSIX. Перевірити яка версія *Python* встановлена можна за допомогою команди: *python*3 *–version*.

## 3.2.1 Репозиторій коду та система контролю версій

Код IaC (*Ansible* *playbook*, *Terraform* конфігурації) має зберігатись в репозиторії контролю версій, це забезпечує відстеження змін, можливість повернення до попередньої версії та співпрацю. Власне для цього нам необхідно зробити наступне:

1) Встановлення *Git*, який є розповсюдженою системою контролю версій. Її потрібно встановити на всіх машинах, де буде редагуватися IaC.

2) Ініціалізація *Git* репозиторію: створіть новий репозиторій за допомогою команди *git* *init*, або ж клонуйте вже наявний репозиторій з віддаленого сховища (*GitLab* або *GitHub*) за допомогою *git* *clone*.

3) Здійснення змін та комітів: після редагування файлів IaC, потрібно додати їх до зони *staging*: *git* *add* . Закомітити зміни із описовим повідомленням: *git* *commit -m* "Описовий коміт"

4) Гілкування та злиття: використовуйте різні гілки для функціоналу та виправлень (*git* *branch* нова\_гілка), після завершення роботи в гілці – виконайте злиття в основну гілку (*git* *merge* нова\_гілка)

5) Віддалені репозиторії: далі необхідно створити обліковий запис у *Git*-хостингу (*GitHub*, *GitLab*, *Bitbucket*), далі додати цей віддалений репозиторій: *git remote add origin* [*https://gitHub.com/repo.git*](https://github.com/repo.git) (як приклад), також регулярно пуште зміни у віддалений репозиторій з використанням команди: *git push -u origin main*

6) Керування доступом та перегляд логів (сповіщень, змін): використовуйте *git* *log* для перегляду історії комітів та відстеження змін.

Додатково можна налаштувати:

* *Git* *hooks* для автоматизації завдань на певних подіях
* Використовувати *branching* *workflow* (*git*-flow, *GitHub* *flow*)
* Інтегрувати *Git* з системами CI/CD для автоматичного тестування/релізу

Використання системи контролю версій *Git* є обов'язковим для ефективної співпраці, відстеження історії та можливості повернення коду IaC в межах практик CI/CD та *DevOps*.

## 3.2.2 Структура репозиторію та організація коду

1) Розділення коду додатку та інфраструктури: далі для ефективного впровадження інструментів CI/CD для побудови інфраструктури на основі IaC, необхідно створити окремі *Git* репозиторії для коду додатку та конфігурацій IaC, наприклад, *repo-app* для коду додатку і *repo-iac* для *Ansible*/*Terraform*. Це забезпечить розділення проблем та незалежність життєвих циклів.

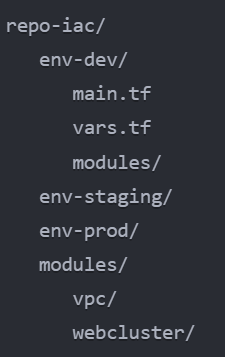
2) Модульний дизайн конфігурацій IaC: в *Ansible* використовуйте ролі для інкапсуляції компонентів інфраструктури, *Terraform* дозволяє створювати модулі для повторного використання. Зберігайте ролі/модулі в окремих підкаталогах репозиторію. Сепарація середовищ: створіть окремі директорії для *dev, staging, prod* та інших середовищ, також дублюйте всю ієрархію конфігурації для кожного середовища, або використовуйте параметри/змінні для розділення конфігурацій.

Рис. 3.2.2.1 Приклад *Ansible* репозиторію та приклад *Terraform* структури Належна структуризація IaC коду допомагає впорядкувати все середовище, забезпечити модульність, можливість реутилізації компонентів та чітке розділення відповідальності між середовищами згідно з практиками *DevOps*.

## 3.2.3 Розгортання тимчасових середовищ

*Terraform* підтримує концепцію робочих просторів, що дозволяє керувати різними екземплярами однієї й тієї ж інфраструктури. Робочі простори забезпечують ізоляцію стану, що дозволяє, наприклад, розгортати різні версії або конфігурації інфраструктури в різних середовищах, таких як *development, staging* та *production*. Ось основні кроки для використання робочих просторів *Terraform* у CI/CD процесі для розгортання тимчасових середовищ:

1. Створення робочих просторів: Перед розгортанням інфраструктури необхідно створити робочий простір для цільового середовища. Це можна зробити за допомогою команди *Terraform* *workspace new* *[NAME]*.

2. Вибір робочого простору: Після створення робочого простору його потрібно вибрати для подальших операцій *Terraform*. Використовуйте команду *Terraform* *workspace select* *[NAME]*.

3. Ізольоване збереження стану: *Terraform* зберігає стан для кожного робочого простору окремо. Це забезпечує ізоляцію інфраструктури середовищ.

4. Конфігурація змінних середовища: Залежно від вимог, ви можете налаштувати змінні середовища для кожного робочого простору. Це дозволить налаштувати поведінку *Terraform* для різних середовищ, наприклад, використовувати різні значення для розмірів ресурсів або налаштувань безпеки.

5. Автоматизація розгортання в CI/CD: В CI/CD процесі можна автоматизувати створення, вибір та застосування робочих просторів для різних середовищ. Наприклад, у процесі безперервної інтеграції можна створювати тимчасовий робочий простір для тестового середовища, застосовувати зміни в інфраструктурі та виконувати тести. Після успішного проходження тестів можна застосувати зміни до наступного середовища (*staging* або *production*).

Після тестування видаліть робочий простір: *Terraform* *workspace select default* && *Terraform* *workspace delete ci-test*.

Для *Ansible*: використовуйте динамічні інвентарі для автоматичного створення списку вузлів. Динамічні інвентарі використовуються в *Ansible* для автоматичного створення списку вузлів на основі різних джерел даних, таких як хмарні провайдери, системи конфігураційного менеджменту або інші інструменти автоматизації. Порядок необхідних кроків використання інвентарів:

**1) Створити скрипт або програму для динамічного інвентаря**

*Ansible* підтримує різні типи динамічних інвентарів, такі як скрипти, виконувані файли або програми на різних мовах програмування. Ці скрипти/програми повинні повертати JSON-дані, що описують хости та їх групи.

Наприклад, ви можете створити скрипт *dynamic*\_*inventory*.*py* на *python*, який буде отримувати список хостів з хмарного провайдера, такого як AWS EC2.

2) **Налаштувати *Ansible* на використання динамічного інвентаря**

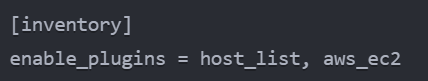
Оновіть файл конфігурації *Ansible* *ansible*.*cfg* або використайте опцію командного рядка, щоб вказати джерело динамічного інвентаря. Наприклад:

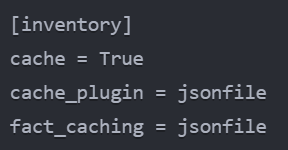
Рис. 3.2.3.1 Приклад використання динамічного інвентаря

Або використайте опцію командного рядка: *Ansible*-*playbook* *-i dynamic\_Inventory.py site.yml*

3) **Виконати *playbook* або *ad-hoc* команду.** Після налаштування динамічного інвентаря ви можете виконувати *playbook* або ***ad-hoc*** команди, як зазвичай. *Ansible* автоматично отримає список хостів від динамічного інвентаря:

*ansible-playbook -i dynamic\_Inventory.py site.yml або Ansible -i dynamic\_Inventory.py all -m ping*

**4) Налаштувати кешування (опціонально).** Динамічний інвентар може бути ресурсоємним процесом, тому *Ansible* дозволяє кешувати результати динамічного інвентаря для підвищення продуктивності. Для цього налаштуйте кешування у файлі конфігурації або за допомогою змінних оточення.

Рис. 3.2.3.1 Приклад налаштування кешування

5) Тестування та налагоджнення

Перевірте вивід скрипту динамічного інвентаря, щоб переконатися, що він повертає належні дані. При налагодженні ви можете використовувати опцію --*list-hosts* для перегляду списку хостів, які повертає динамічний інвентар:

*ansible-Inventory -i dynamic\_Inventory.py --list-hosts*

Використання динамічних інвентарів дозволяє автоматично отримувати актуальний список хостів із зовнішніх джерел даних, що є особливо корисним у хмарних та динамічних середовищах. Це спрощує процес управління конфігураціями та оркестрацією завдяки *Ansible*.

Видалення тимчасових ресурсів: зазвичай етапи CI/CD мають виконувати "прибирання" після тестів. В *Ansible* через окремі завдання або *handlers*. В *Terraform* через *Terraform* *destroy* для відповідного робочого простору.

## 3.2.4 Універсальні вхідні параметри та секрети

Параметризація конфігурацій для переносимості між середовищами: для забезпечення переносимості конфігурацій *Terraform* між різними середовищами використовуються вхідні змінні (*input variables*). Змінні дозволяють параметризувати значення, такі як імена ресурсів, розміри екземплярів, налаштування мережі тощо. Визначення змінних відбувається у файлі `*variables.tf*`, а їх значення можна задавати за допомогою файлів змінних середовища (.*tfvars*) або командного рядка під час застосування конфігурацій. Використання змінних забезпечує гнучкість та можливість легко адаптувати конфігурації до різних вимог середовищ, не змінюючи основний код *Terraform*.

Під час розгортання інфраструктури часто потрібно передавати конфіденційну інформацію, таку як креденшали доступу до хмарних провайдерів, токени доступу або паролі. Зберігання таких секретів безпосередньо у конфігураційних файлах *Terraform* є небезпечною практикою, оскільки це може призвести до витоку конфіденційних даних. Для безпечного зберігання та передачі секретів рекомендується використовувати спеціалізовані інструменти керування секретами, такі як *HashiCorp Vault, AWS Secrets Manager, Azure Key Vault* тощо. Ці інструменти дозволяють централізовано та безпечно зберігати секрети та надавати тимчасові креденшали або токени під час виконання CI/CD процесів.

## 3.2.5 Тестові набори та перевірка IaC коду

Тестування є важливим аспектом у процесі розробки та застосування IaC для забезпечення якості, надійності та відповідності вимог. Розглянемо декілька підходів до тестування IaC коду, зокрема *Terraform*.

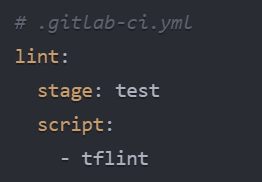
1) Статичний аналіз та юніт-тести. Перед застосуванням змін у *Terraform* конфігураціях, необхідно виконати базову перевірку синтаксису за допомогою команди *Terraform* *validate*. Для більш глибокого статичного аналізу можна використати інструмент *tflint*. Його можна інтегрувати в CI/CD процес, наприклад, в *GitLab* CI/CD:

Рис. 3.2.5.1 Інструмент *tflint*

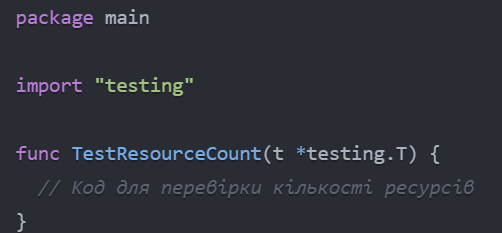
Для юніт-тестування *Terraform* коду можна використати фреймворк *go test.* Для прикладу, створіть файл *main\_test.go* поруч з main.tf:

Рис. 3.2.5.2 Приклад юніт-тесту

Додайте крок виконання юніт-тестів у CI/CD конвеєрі.

2) Інтеграційне тестування:

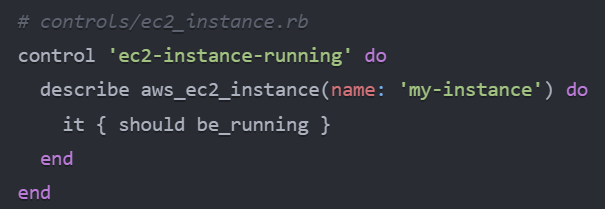
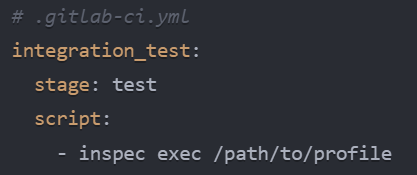
Для інтеграційного тестування розгорнутих ресурсів можна використати інструмент *inspec*. Створіть профіль *InSpec* з тестами для перевірки стану ресурсів, наприклад, перевірка стану EC2 instances в AWS:

Рис. 3.2.5.3 Перевірка стану EC2 instances в AWS

Додайте крок запуску InSpec тестів у CI/CD конвеєрі після успішного застосування *Terraform* змін:

Рис. 3.2.5.4 Запуск InSpec тестів у CI/CD конвеєрі

3) Тестування відповідності політикам

Для перевірки відповідності *Terraform* конфігурацій певним політикам безпеки та кодовим стандартам можна використати *Terraform*-*compliance*.

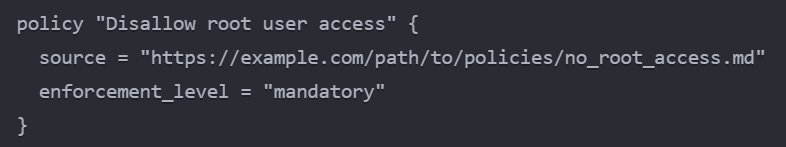
Створіть файл з правилами відповідності:

Рис. 3.2.5.5 Приклад *policies/policy.md*

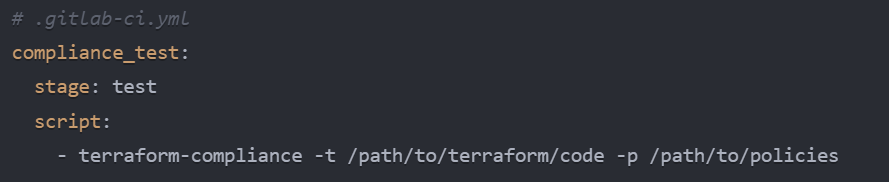
Інтегруйте виконання *Terraform*-*compliance* в CI/CD конвеєрі:

Рис.3.2.5.6 інтегрування виконання *Terraform*-*compliance* в CI/CD конвеєрі

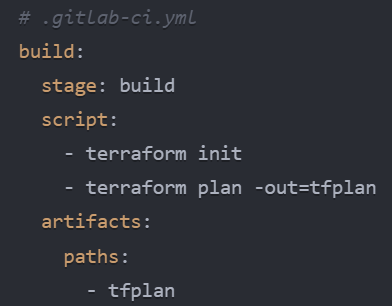
Ці приклади демонструють, як інтегрувати різні види тестування IaC коду в CI/CD процеси. Послідовність дій може включати: статичний аналіз → юніт-тести → застосування *Terraform* змін → інтеграційні тести → тести відповідності політикам. Така практика забезпечує якість, надійність та безпеку IaC коду перед розгортанням у продуктивне середовище.

## 3.2.6 Артефакти конфігурацій

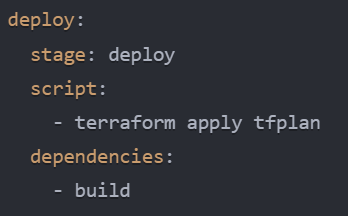
Розглянемо питання артефактів конфігурацій в CI/CD процесах з IaC інструментами, такими як *Terraform*.

1) Збереження та передача збірок конфігурацій між етапами конвеєра: в CI/CD процесах часто потрібно передавати артефакти (наприклад, конфігураційні файли, стан *Terraform*) між різними етапами конвеєра. Для цього використовуються спеціальні сховища артефактів (*artifact repositories*) або тимчасові сховища, інтегровані в CI/CD систему.

Наприклад, в *GitLab* CI/CD можна використовувати вбудовані сховища артефактів. Після створення збірки конфігурацій їх можна зберегти як артефакт:

Рис. 3.2.6.1 Приклад збереження артефакту

Наступний етап конвеєра - артефакт можна завантажити та використати:

Рис. 3.2.6.2 Приклад використання артефакту

2) Використання *artifact repositories* або *immutable infrastructure artifacts.*

Крім тимчасових сховищ артефактів, інтегрованих у CI/CD системи, можна використовувати спеціалізовані сховища артефактів інфраструктури (*artifact repositories*). Ці сховища зберігають незмінні (*immutable*) артефакти інфраструктури, такі як конфігураційні файли *Terraform*, стани, бінарні файли провайдерів тощо.

## 3.2.7. Схвалення змін та безпечне розгортання

Схвалення змін та безпечні стратегії розгортання є важливими аспектами CI/CD процесів, особливо коли йдеться про управління інфраструктурою за допомогою IaC інструментів, таких як *Terraform*. Розглянемо ці моменти:

1. Ручні або автоматизовані схвалення перед застосуванням змін:

Перед застосуванням змін в інфраструктурі, особливо в продуктивному середовищі, рекомендується впровадити етап схвалення змін. Ручне схвалення передбачає, що людина (наприклад, *DevOps* інженер або адміністратор) переглядає та схвалює зміни, перш ніж вони будуть застосовані. Автоматизоване схвалення може бути реалізоване шляхом перевірки певних умов або політик, наприклад, успішне проходження всіх тестів, відповідність правилам безпеки тощо. В *GitLab* CI/CD етап схвалення можна реалізувати з допомогою *manual job*:

Рис. 3.2.7.1 Приклад розгортання з ручним схваленням.

## 3.2.8. Моніторинг та журналювання

Розглянемо важливий аспект моніторингу та журналювання в CI/CD процесах з IaC інструментами:

1. Відстеження подій і статусу розгортання через логи та моніторинг: Під час CI/CD процесів, особливо на етапах розгортання та застосування змін в інфраструктурі, важливо мати можливість відстежувати події, статус та результати виконання. Більшість IaC інструментів, таких як *Terraform*, *Ansible* та ін., генерують вихідні логи, які можна збирати та аналізувати. В CI/CD системах, таких як *GitLab* CI/CD, *Jenkins* або інших, логи кожної задачі (*job*) зберігаються та доступні для перегляду. Крім того, можна інтегрувати централізовану систему збору та аналізу логів, наприклад, *ELK Stack (Elasticsearch, Logstash, Kibana), Graylog або Splunk.* Ці системи дозволяють не лише зберігати та переглядати логи, але й виконувати пошук, фільтрацію, візуалізацію та встановлювати сповіщення на основі певних подій у логах.

2. Інтеграція з системами моніторингу інфраструктури та додатків:

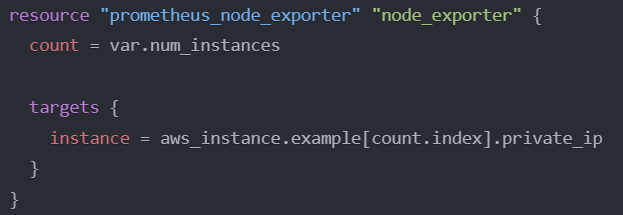
Після успішного розгортання інфраструктури та додатків за допомогою IaC інструментів важливо інтегрувати їх з системами моніторингу. Системи моніторингу, такі як *Prometheus, Datadog, New Relic* або *AWS CloudWatch,* дозволяють відстежувати стан та продуктивність розгорнутих ресурсів, включаючи обчислювальні ресурси, мережі, сховища даних, додатки тощо. Інтеграція з цими системами може бути реалізована різними способами, наприклад, за допомогою агентів моніторингу, які розгортаються разом з інфраструктурою, або шляхом використання вбудованих можливостей моніторингу хмарних провайдерів. Дані моніторингу можуть використовуватись для виявлення та усунення проблем, оптимізації ресурсів, встановлення сповіщень про інциденти та забезпечення відповідності вимогам SLA.

Рис. 3.2.8.1 Приклад інтеграції з *Prometheus* в *Terraform*

Цей код розгортає *Node Exporter* на кожному створеному EC2 instance в AWS, дозволяючи *Prometheus* збирати метрики про стан та їх продуктивність.

Впровадження належного моніторингу та журналювання в CI/CD процеси з IaC інструментами забезпечує прозорість, відстеження подій, своєчасне виявлення проблем та полегшує усунення несправностей у разі виникнення будь-яких проблем під час розгортання або експлуатації інфраструктури.

## 3.2.9 Зворотні засоби (*rollback*, *rollforward*)

Розглянемо ще не менш важливу тему зворотних засобів (*rollback* та *rollforward*) у CI/CD процесах з інструментами IaC, такими як *Terraform*:

1. Зберігання попереднього та поточного станів інфраструктури:

- Більшість IaC інструментів, зокрема *Terraform*, зберігають стан інфраструктури у спеціальних файлах стану (*state files*). Ці файли містять інформацію про створені ресурси, їх налаштування та зв'язки між ними. Під час виконання операцій з інфраструктурою (створення, оновлення, видалення) *Terraform* зберігає поточний стан після застосування змін. Перед застосуванням нових змін *Terraform* також зберігає попередній стан інфраструктури, який можна використовувати для відкату змін у разі потреби.

2. Можливість відкату або перезастосування змін у разі невдачі:

*Rollback*: Якщо під час розгортання нових змін виникла помилка або непередбачена ситуація, *Terraform* забезпечує можливість відкату до попереднього робочого стану інфраструктури. Для виконання відкату потрібно використати команду *terraform* *apply* з попереднім файлом стану: *terraform* *apply -state=previous.tfstate*. Це призведе до відкату всіх змін, внесених під час останнього застосування, і відновить інфраструктуру до попереднього стабільного стану. *Rollforward*: У деяких ситуаціях може знадобитися перезастосування змін, якщо попередня спроба розгортання була невдалою через тимчасові проблеми. Для цього можна використати ту саму команду *terraform* *apply* з файлом плану змін, який було згенеровано під час невдалої спроби розгортання: *Terraform* *apply tfplan*. Це дозволить спробувати застосувати ті самі зміни ще раз, без необхідності їх повторної генерації.

Для забезпечення можливості відкату та *rollforward* необхідно належним чином організувати зберігання файлів стану та планів змін. Це можна зробити, використовуючи віддалені сховища стану (*remote state backends*), такі як *Terraform* Cloud, *AWS S3, Azure Blob Storage* тощо. Також рекомендується зберігати файли стану та плани змін як артефакти в CI/CD конвеєрах.

## 3.2.10 Гібридні середовища та різні хмарні провайдери

Розглянемо питання підтримки гібридних середовищ та різних хмарних провайдерів у CI/CD процесах з інструментами IaC, такими як *Terraform*.

1. CI/CD конвеєри мають підтримувати багатохмарність. Сучасна інфраструктура часто розгортається у різних хмарних середовищах або поєднує хмарні та локальні (*on-premise*) ресурси, утворюючи гібридну архітектуру. CI/CD процеси та IaC інструменти повинні бути здатними працювати з різними хмарними провайдерами, такими як AWS, *Azure*, GCP, а також з локальними платформами, наприклад, *VMware* або OpenStack. *Terraform* має вбудовану підтримку багатьох провайдерів, що дозволяє керувати інфраструктурою в різних середовищах за допомогою єдиного набору конфігурацій. У CI/CD конвеєрах можна визначати різні етапи або задачі для розгортання в різних хмарах або середовищах, використовуючи відповідні конфігурації *Terraform*.

2. Узгодження конфігурацій для різних платформ.

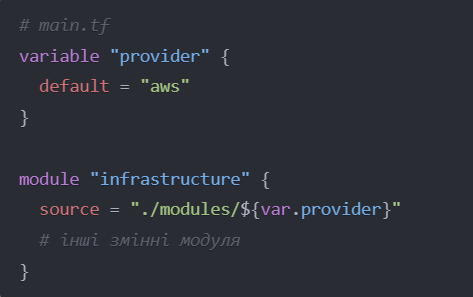
Хоча *Terraform* забезпечує уніфікований підхід до управління інфраструктурою, деякі ресурси та їх налаштування можуть відрізнятися між різними провайдерами або середовищами. Для вирішення цієї проблеми можна використовувати модульну структуру конфігурацій *Terraform* та умовну логіку на основі змінних. Наприклад, можна створити окремі модулі для ресурсів AWS та *Azure*, а в головній конфігурації визначити змінну для вибору провайдера та відповідного модуля. Альтернативно, можна використовувати умовні вирази *count* або *for\_each* для динамічного створення ресурсів залежно від вибраного провайдера або середовища.

Рис. 3.2.10.1 Приклад використання модулів для різних провайдерів

Тоді в директоріях *module*s/*aws* та *module*s/*azure* будуть знаходитись відповідні конфігурації для кожного провайдера. Крім того, у CI/CD конвеєрах можна визначати різні змінні середовища або використовувати файли змінних для різних середовищ розгортання.

Підтримка багатохмарності та узгодження конфігурацій для різних платформ є важливими аспектами для забезпечення гнучкості та масштабованості CI/CD процесів з IaC інструментами в сучасних гібридних ІТ-середовищах. Ретельне планування та врахування цих практичних аспектів забезпечить ефективну інтеграцію IaC інструментів з конвеєрами CI/CD, посилюючи переваги *DevOps* підходу.

## Висновки до розділу 3

У даному розділі було проаналізовано ключові аспекти та практичні підходи до інтеграції CI/CD процесів з інструментами IaC, такими як *Terraform* та *Ansible*, для забезпечення ефективного розгортання та управління інфраструктурою. Основні висновки можна сформулювати таким чином:

Для успішної реалізації CI/CD процесів з IaC необхідно встановити всі необхідні інструменти (*Git*, *Ansible*, *Terraform*) та налагодити інтеграцію із системою контролю версій і репозиторієм для зберігання та відстеження змін в інфраструктурному коді.

Використання робочих просторів (*workspaces*) в *Terraform* дозволяє ефективно розділяти та ізолювати тимчасові середовища для розгортання та тестування змін, забезпечуючи надійність та контрольоване управління інфраструктурою.

Параметризація конфігурацій за допомогою вхідних змінних та безпечне зберігання секретів у спеціалізованих сховищах (*Vault, Secrets Manager* тощо) забезпечує гнучкість, переносимість та безпеку під час розгортання інфраструктури в різних середовищах.

Впровадження статичного аналізу, юніт-тестів, інтеграційного тестування та тестування відповідності політикам за допомогою фреймворків, таких як *tflint*, inspec, goss та *Terraform*-*compliance*, дозволяє підвищити якість та надійність IaC коду перед його розгортанням у продуктивні середовища.

Використання централізованих сховищ артефактів (*artifact repositories*) та незмінних артефактів інфраструктури (*immutable infrastructure artifacts*) забезпечує відтворюваність, стабільність та контроль версій під час розгортання інфраструктури.

Етапи схвалення змін (ручні або автоматизовані) та використання безпечних стратегій розгортання (*blue/green, canary, rolling update*) дозволяють мінімізувати ризики та забезпечити контрольоване впровадження змін в інфраструктурі.

Інтеграція з системами моніторингу та журналювання (*ELK Stack, Prometheus* тощо) забезпечує прозорість, відстеження подій, своєчасне виявлення проблем та полегшує усунення несправностей під час розгортання та експлуатації інфраструктури.

Можливість відкату змін (*rollback*) та перезастосування змін (*rollforward*) за допомогою зберігання попередніх станів інфраструктури та планів змін забезпечує гнучкість, стійкість до помилок та швидке відновлення у разі виникнення проблем.

Підтримка багатохмарності та гібридних середовищ, а також узгодження конфігурацій для різних платформ за допомогою модульної структури коду та умовної логіки забезпечують масштабованість та гнучкість CI/CD процесів з IaC інструментами в сучасних ІТ-середовищах. Впровадження цих підходів та практик у CI/CD процеси з інструментами IaC забезпечує ефективне, надійне та контрольоване розгортання інфраструктури, підвищує якість коду, мінімізує ризики та сприяє швидкому реагуванню на зміни та інциденти.

# Висновки

Отже, впровадження принципів CI/CD та інфраструктури як коду (IaC) в процесах розгортання ІТ-інфраструктури для Збройних Сил України є вкрай важливим для підвищення ефективності, надійності та безпеки.

Основні переваги даного підходу:

1. Скорочення часу на розгортання та оновлення систем за рахунок автоматизації процесів.

2. Підвищення стабільності та передбачуваності розгорнутого середовища завдяки систематичній перевірці змін.

3. Покращення безпеки шляхом швидкого реагування на виявлені загрози та вразливості.

4. Зменшення ризику людських помилок через зведення до мінімуму ручних операцій.

5. Централізоване управління конфігураціями та стандартизація середовищ.

6. Можливість швидкого відкату змін у разі виникнення проблем.

Для успішного впровадження необхідно провести навчання персоналу, інтегрувати відповідні інструменти та налаштувати безперервні конвеєри доставки для автоматизації розгортання. Також слід приділити особливу увагу забезпеченню безпеки на всіх етапах циклу.

Перехід до використання CI/CD та IaC дозволить Збройним Силам України підвищити загальну ефективність ІТ-операцій, скоротити витрати та забезпечити відповідність найсучаснішим практикам у галузі.

# Список використаних джерел

1. Refactoring: Improving the Design of Existing Code. 2-ге вид. Addison-Wesley Professional, 2019. 455 с. Signature Series (Fowler).
2. Gene Kim Patrick Debois John Willis. The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations. IT Revolution Press, 2016. 480 с.
3. Michael Nygard. Release It!: Design and Deploy Production-Ready Software. 2-ге вид. Pragmatic Bookshelf, 2018. 378 с.
4. Craig Berg. DevOps For Beginners: A Complete Guide To DevOps Best Practices (Including How You Can Create World-Class Agility, Reliability, And Security In Technology ... With DevOps). 2020. 117 с.
5. Adam Wiggins. The Twelve-Factors App. [*https://12factor.net/*](https://12factor.net/).
6. Mykyta Protsenko. Infrastructure as Code: Базові принципи vs інструменти, що еволюціонують. *https://dou.ua/lenta/articles/infrastructure-as-code/*.