Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет *Комп'ютерних наук*

Кафедра *Системотехніки*

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Розробка та дослідження програмного забезпечення GEO OSINT

Виконав:

здобувач 2 року навчання,

групи *ІТПм-24-1*

групи ІТПм-24-1

Роман Герасименко

(власне ім'я, прізвище)

Спеціальність 122 Комп'ютерні науки

Тип програми *освітньо-наукова*

Освітня програма *Інформаційні технології проєктування*

Керівник проф. каф. СТ. Вадим Саваневич

(посада, власне ім'я, прізвище)

Допускається до захисту

Зав. кафедри _____ *Гребеннік І.В.*

2025 р.

Я, як студент ХНУРЕ розумію і підтримую політику закладу із академічної доброчесності. Я не надавав і не одержував недозволену допомогу під час підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

15.12.2025  *Герасименко Р. Ю.*

Кваліфікаційна робота не містить відомостей заборонених до відкритого опублікування.

Кваліфікаційна робота виконана у відповідності до стандартів, що діють в Україні.

Попередній захист проведено 15 грудня 2025

Керівник кваліфікаційної роботи  *проф. Саваневич В.Є.*

Харківський національний університет радіоелектроніки

Факультет *Комп'ютерних наук*

Кафедра *Системотехніки*

Рівень вищої освіти другий (магістерський)

Спеціальність 122 – Комп'ютерні науки

Тип програми освітньо-наукова

Освітня програма Інформаційні технології проєктування

ЗАТВЕРДЖУЮ

Зав. кафедри *Системотехніки*

. *проф. Гребеннік І.В.*

" " *2025 р.*

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачу *Герасименко Роману Юрійовичу*

1. Тема роботи *Розробка та дослідження програмного забезпечення GEO OSINT*

затверджена наказом по університету від "24" *листопада* 2025р. № 1058*Ст*

2. Термін подання студентом роботи до екзаменаційної комісії *18 грудня 2025р.*

3. Вихідні дані до роботи *Розробка системи автоматизації розгортання та моніторингу багатоканальної оптичної системи виявлення та супроводу рухомих об'єктів **Amanita**. Необхідно реалізувати серверну та клієнтську частину інфраструктури для компонентів **camerapro**, **datapro1**, **datapro2**, **turretpro**, забезпечивши автоматичне налаштування, розгортання та контроль стану системи у віртуальному та реальному середовищі. Перелік використаних програмних засобів: Linux Ubuntu 20.04 LTS, Docker, Vagrant, Python, Bash, OpenCV, FFmpeg, Visual Studio Code, GitHub.*

4. Перелік питань, що потрібно опрацювати в роботі 4.1 Вступ 4.2 Обґрунтування актуальності, мети та завдань дослідження 4.3 Аналіз предметної області систем комп'ютерного бачення 4.4 Дослідження проблем ручного розгортання 4.5 Аналіз сучасних підходів до автоматизації DevOps та IaC 4.6 Розроблення архітектури системи автоматизованого розгортання та моніторингу 4.7 Опис компонентів Builder, Configurator, Installer та Manager 4.8 Розроблення сценаріїв автоматизації на основі Bash і Python 4.9 Створення форматів YAML/JSON для конфігураційних файлів 4.10 Реалізація автоматизованого процесу розгортання та моніторингу 4.11 Тестування системи у віртуальному середовищі Vagrant 4.12 Експериментальне дослідження на реальних вузлах 4.13 Оцінка ефективності автоматизації 4.14 Аналіз результатів та формування висновків

5. Перелік графічного матеріалу 1 Розробка та дослідження програмного забезпечення GEO OSINT, 2 Об'єкт, предмет та мета дослідження, 3 GEO OSINT (Geospatial Open Source Intelligence), 4 Системи комп'ютерного бачення та GEO OSINT-модулі, 5 Класифікація комп'ютерного бачення, 6 Типове середовище комп'ютерного бачення, 7 Моделі CV, здатні

працювати на обмежених обчислювальних ресурсах, 8 Сучасні підходи автоматизації систем комп'ютерного бачення, 9 Сучасні підходи автоматизації системи Amanita, 10 Інфраструктура як код та DevOps –підходи в Amanita, 11 Сучасні підходи до автоматизації системи Amanita, 12 Архітектура системи. Основні компоненти системи, 13 Архітектура системи, 14 Архітектура системи. Компонент Manager, 15 Файлова організація та схема роботи, 16 Мережева модель і конфігурування системи, 17 Конфігурація системи, 18 Функціональні вимоги до системи, 19 Нефункціональні вимоги до системи, 20 Надійність конфігураційних даних та середовища, 21 Сценарій 1, 22 Сценарії 2 та 3, 23 Загальна послідовність реалізації сценаріїв, 24 UML-діаграми використання системи, 25 Діаграма послідовності дій інженера під час розгортання системи, 26 Реалізація системи, 27 Модулі Builder і Configurator, 28 Модулі Installer і Manager, 29 Тестування у віртуальному середовищі, 30 Два основні сценарії тестування системи, 31 Два основні сценарії тестування системи, 32 Тестування автоматизованого розгортання, 33 Рішення розгортання на реальному обладнанні, 34 Тестування негативних сценаріїв, 35 Експериментальні результати та оцінка ефективності системи, 36 Коефіцієнт автоматизації, 37 Коефіцієнт відмовостійкості, 38 Коефіцієнт продуктивності, 39 Коефіцієнт стабільності, 40 Результати оцінки ефективності системи, 41 Критерії ефективності системи розгортання, 42 Порівняння з традиційними підходами, 43 Висновки, 44 Апробація, 45 Кінець.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів атестаційної роботи	Термін виконання етапів роботи	Примітка
1.	*Отримання завдання на виконання кваліфікаційної роботи*	*24.11.2025*	*Виконано*
2.	*Аналіз завдання та предметної області*	*25.11–27.11.2025*	*Виконано*
3.	*Опрацювання літератури та аналіз об'єкта дослідження*	*28.11–29.11.2025*	*Виконано*
4.	*Формування постановки задачі*	*30.11–1.12.2025*	*Виконано*
5.	*Проєктування апаратного та програмного засобу*	*02.12–03.12.2025*	*Виконано*
6.	*Розробка та тестування апаратного та програмного засобу*	*04.12–08.12.2025*	*Виконано*
7.	*Аналіз результатів дослідження, отриманих за допомогою розробленої системи*	*08.12–10.12.2025*	*Виконано*
8.	*Оформлення пояснювальної записки та презентаційних матеріалів*	*10.12–15.12.2025*	*Виконано*
9.	*Представлення роботи на рецензування*	*15.12.2025*	*Виконано*
10.	*Представлення атестаційної роботи в ДЕК*	*18.12.2025*	*Виконано*

Дата видачі завдання 24 листопада 2025р.
Здобувач _____
Керівник роботи _____ *проф. каф. СТ Саваневич В.Є.*
(підпис) (посада, власне ім'я, прізвище)

ЗМІСТ

РЕФЕРАТ

Пояснювальна записка до магістерської кваліфікаційної роботи: 61 с., 5 табл., 6 рис., 1 додаток, 18 джерел.

Об'єкт дослідження – процеси аналізу, обробки та автоматизації роботи з геопросторовими та візуальними даними у програмному забезпеченні GEO OSINT, включно з інструментами комп'ютерного бачення.

Предмет дослідження – методи та програмні засоби розробки і автоматизації функціонування компонентів GEO OSINT-програмного забезпечення, зокрема засоби DevOps, віртуалізації та контейнеризації для забезпечення відтворюваності, масштабованості та стабільності процесів аналізу медіаданих за допомогою інструментів комп'ютерного бачення.

Мета роботи – розробити та дослідити програмне забезпечення GEO OSINT, а також реалізувати практичний модуль автоматизованої обробки та керування медіаданими на прикладі багатокомпонентної системи комп'ютерного бачення Amanita, використовуючи сучасні підходи DevOps, інфраструктури як коду, віртуалізації та контейнеризації.

Для досягнення мети в роботі проаналізовано існуючі підходи до автоматизації розподілених систем; визначенні вимоги до архітектури Amanita як системи комп'ютерного бачення; реалізовані засоби збирання, конфігурування та інсталяції модулів; забезпечена підтримка віртуального тестування та ведення системних журналів; перевірені ефективність ручного й автоматизованого розгортання.

У роботі використано DevOps-підходи, технології інфраструктури як коду, віртуалізацію (Vagrant), контейнеризацію (Docker) та системи керування версіями.

Ключові слова: GEO OSINT, комп'ютерне бачення, аналіз медіаданих, автоматизація розгортання, DevOps, Docker, Vagrant, Bash, YAML, Python.

СПИСОК СКОРОЧЕНЬ

AI — Artificial Intelligence (штучний інтелект).

API — Application Programming Interface (інтерфейс прикладного програмування).

CI/CD — Continuous Integration / Continuous Deployment (безперервна інтеграція та розгортання).

CPU — Central Processing Unit (центральний процесор).

CV — Computer Vision (комп'ютерне бачення).

DevOps — Development & Operations (практика інтеграції розробки та експлуатації).

IaC — Infrastructure as Code (інфраструктура як код).

JSON — JavaScript Object Notation (текстовий формат структурованих даних).

SSH — Secure Shell (протокол віддаленого доступу).

VM — Virtual Machine (віртуальна машина).

YAML — Yet Another Markup Language (формат конфігураційних файлів).

CLI — Command Line Interface (інтерфейс командного рядка).

Vagrant — інструмент для керування віртуальними середовищами.

Docker — система контейнеризації застосунків.

ВСТУП

Актуальність теми. Системи комп’ютерного бачення активно впроваджуються у сфери, що потребують автоматичного виявлення, аналізу та реагування на події у реальному часі. Розгортання таких систем зазвичай включає кілька функціонально незалежних компонентів, що розміщуються на окремих вузлах — відеомодулях, модулях обробки та модулях керування. Ускладнення архітектури та підвищення вимог до гнучкості і стабільності роботи актуалізує потребу в автоматизації процесів збирання, конфігурування та розгортання цих систем. Окремим напрямом розвитку таких технологій є GEO OSINT — аналіз геопросторових і візуальних даних з відкритих джерел, який значною мірою залежить від якості, відтворюваності та автоматизації процесів обробки мультимедійної інформації. У таких системах модулі комп’ютерного бачення виступають важливою складовою, оскільки забезпечують попередню підготовку та структуризацію даних, необхідних для подальшої аналітики. У практиці використання систем комп’ютерного бачення часто застосовуються ручні дії: компіляція коду, редагування конфігурацій, копіювання файлів, запуск через термінал. Це не лише уповільнює процес, а й ускладнює масштабування, викликає помилки та створює труднощі з підтримкою. Застосування підходів IaC і DevOps-практик дозволяє зменшити ці ризики й підвищити надійність розгортання. У контексті GEO OSINT це особливо важливо, оскільки аналітичні процеси мають базуватись на стабільних та відтворюваних потоках даних, а автоматизація розгортання є критичною для побудови надійних інструментів аналізу. Метою роботи є створення засобів автоматизованого розгортання Багатоканальної оптичної системи виявлення та супроводу рухомих об’єктів Amanita з підтримкою декларативного опису її структури, формування конфігурацій та централізованого керування модулями. У межах тематики GEO OSINT система Amanita розглядається як приклад програмного компонента, що забезпечує автоматизовану обробку візуальних даних і може бути

інтегрований у геоаналітичні робочі процеси. У якості методів дослідження використовувались системний аналіз, моделювання архітектури, застосування технологій Docker, Vagrant, Bash і Python-скриптів, аналіз журналів системних подій. Для досягнення мети в роботі проаналізовано існуючі підходи до автоматизації розподілених систем; визначенні вимоги до архітектури Amanita як системи комп'ютерного бачення; реалізовані засоби збирання, конфігурації та інсталяції модулів; забезпечена підтримка віртуального тестування та ведення системних журналів; перевірені ефективність ручного й автоматизованого розгортання. Результати цих досліджень релевантні для GEO OSINT-систем, де надійність та повторюваність інфраструктури прямо впливають на точність аналізу даних. В результаті в роботі розроблено архітектуру системи автоматизації розгортання, створено формати YAML/JSON для конфігураційних файлів, реалізовано механізм моніторингу стану вузлів і виконано тестування у віртуальному та реальному середовищі. Отримані рішення можуть застосовуватися як частина більш широкого GEO OSINT-програмного забезпечення, що працює з візуальними потоками даних. Основними користувачами розробленої системи є інженер та спостерігач. Інженер відповідає за підготовку апаратно-програмного середовища, налаштування мережевої інфраструктури Amanita, запуск процедур автоматизованого розгортання та усунення технічних збоїв. Спостерігач працює з уже налаштованим комплексом, здійснює моніторинг відеопотоків, контролює коректність супроводу об'єктів та приймає оперативні рішення під час спостереження. У роботі використано DevOps-підходи, технології інфраструктури як коду, віртуалізацію (Vagrant), контейнеризацію (Docker) та системи керування версіями. Практична цінність роботи полягає в тому, що запропонована система дозволяє скоротити час розгортання Amanita, зменшити кількість помилок конфігурації, забезпечити відтворюваність середовища та спростити технічне обслуговування без залучення сторонніх оркестраторів.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

У межах цієї роботи GEO OSINT (Geospatial Open Source Intelligence) розглядається як клас програмного забезпечення для геопросторової розвідки з відкритих джерел, яке поєднує різнорідні джерела даних (карти, супутникові знімки, геотеги, відкриті публікації, відео) з інструментами комп'ютерного бачення та аналітики. Компоненти комп'ютерного бачення в таких системах виконують роль «сенсорного шару»: вони перетворюють необроблені зображення та відео на структуровані медіадані (об'єкти, траєкторії, події), що надалі можуть використовуватися у аналітичних сценаріях. До цього класу компонентів належить багатоканальна оптична система виявлення та супроводу рухомих об'єктів Amanita, яка в цій роботі виступає прикладом практичної реалізації модульного програмного забезпечення для GEO OSINT.

1.1. Системи комп'ютерного бачення

Системи комп'ютерного бачення за останнє десятиліття активно розвиваються у напрямі промислової автоматизації. Згідно з дослідженнями [1] та [2] встановлено, що інтеграція з розподіленими обчислювальними системами забезпечує високу швидкість обробки даних.

У монографії [1] система комп'ютерного бачення розглядається як сукупність програмних модулів, які отримують, обробляють і аналізують зображення або відеопотоки з метою виявлення, класифікації та відстеження об'єктів. Такі системи складаються з:

– модулів захоплення даних (камери, драйвери, буфери),

– модулів обробки (фільтрація, сегментація, аналітика),

– модулів прийняття рішень (керування виконавчими пристроями, генерація подій).

Сучасні системи комп'ютерного бачення поєднують алгоритми машинного навчання та оптимізації потоків даних. Як зазначено у [1], ефективність

таких систем залежить від поєднання алгоритмічної складності та швидкодії обчислювального середовища.

У архітектурі створеного проєкту ці модулі реалізуються окремими програмами:

– camerapro – зчитування відеопотоку, передача кадрів;

datapro1, datapro2 – аналітичні модулі, що виконують обчислення на CPU;

– turretpro – керування рухомою частиною системи (наприклад, поворотною камерою);

– manager – головний координатор, який забезпечує інтерфейс спостерігача для запуску, моніторингу та синхронізації підлеглих модулів.

У контексті програмного забезпечення GEO OSINT системи комп'ютерного бачення виступають критично важливими джерелами первинних даних. Вони забезпечують перетворення необроблених відео- та фотоматеріалів на структуровані ознаки: виявлені об'єкти, їх класи, траєкторії руху, просторові відносини та події. Надалі ці ознаки можуть бути співставлені з геопросторовими даними, картографічними сервісами або супутниковими знімками, що дозволяє автоматизувати частину традиційно «ручних» процедур геолокації та перевірки достовірності медіаконтенту.

Таким чином, системи комп'ютерного бачення є проміжною ланкою між «сирими» медіаданими та аналітичними GEO OSINT-модулями, які виконують просторовий аналіз, побудову картин подій та виявлення прихованих закономірностей.

Розроблений у роботі комплекс Amanita можна розглядати як приклад прикладного модуля у складі GEO OSINT-платформи. Модулі camerapro, datapro1/2, turretpro та manager забезпечують безперервний збір та обробку відеоданих, формуючи потік подій і вимірювань, які надалі можуть бути співставлені з геопросторовим контекстом (розташування сенсорів, сектор огляду,

маршрут об’єктів тощо). Завдяки цьому система Amanita не лише виконує локальні задачі виявлення та супроводу цілей, але й потенційно може стати складовою більш загальної GEO OSINT-інфраструктури для моніторингу територій, виявлення аномалій та аналізу поведінки об’єктів у просторі.

1.2. Класифікація та середовище комп’ютерного бачення

Як показано у працях [1; 5], системи комп’ютерного бачення (CV-системи) відрізняються за способом обробки даних та структурою компонентів. Залежно від архітектури, розподілу навантаження і взаємодії модулів виділяють три основні наступні типи.

Монолітні системи. Усі етапи (захоплення відеопотоку, обробка, аналіз, виявлення об’єктів і керування виконавчими механізмами) реалізовані в одному застосунку на одному пристрої. Це забезпечує простоту, але ускладнює масштабування та оновлення.

Багатокомпонентні локальні системи. Компоненти (наприклад, модулі camerapro, datapro1, datapro2, turretpro) виконуються на окремих вузлах локальної мережі. Такий підхід забезпечує масштабованість, розподіл навантаження й вищу стійкість до збоїв.

Розподілені системи з хмарною або гібридною інфраструктурою. Частина обробки виконується на edge-пристроях, частина — у хмарі або центральному сервері. Це дозволяє централізований аналіз і колективне навчання моделей, але потребує стабільного мережевого з’єднання.

У рамках даної роботи система Amanita підтримує перший і другий типи архітектур. Її можна розгорнути як на одному пристрої, так і на кількох вузлах локальної мережі з мінімальним втручанням людини.

Особливості середовища систем комп’ютерного бачення. Для реалізації обробки зображень використовуються вузли на базі x86-64 архітектури під керуванням Ubuntu Linux**.** Це забезпечує:

– сумісність із бібліотеками (OpenCV, FFmpeg, NumPy),

– стабільну роботу без графічних прискорювачів (GPU),

– можливість тестування у віртуальному середовищі (через Vagrant) без фізичних пристроїв.

Типові компоненти системи:

– camerapro – захоплення відео,

– datapro1/2 – обробка та аналіз кадрів,

– turretpro – керування виконавчими пристроями,

– manager – координація модулів та логування.

Основні проблеми, які вирішує Amanita:

– залежність від ручних дій при інсталяції;

– конфлікти версій бібліотек;

– потреба вручну запускати компоненти на кожному вузлі.

Автоматизація усуває ці недоліки, забезпечуючи повторюваність і швидкість розгортання. Завдяки YAML-файлу конфігурації основні етапи розгортання можуть бути ініційовані однією командою; при цьому інженер здійснює первинну підготовку середовища та перевірку параметрів.

У контексті GEO OSINT системи комп'ютерного бачення охоплюють широкий спектр алгоритмічних задач, що забезпечують перетворення візуальної інформації на структуровані дані, придатні для подальшого просторового аналізу. Основні з них:

1. Детекція об'єктів – визначення транспортних засобів, людей, техніки або інфраструктурних елементів на зображеннях. Результати детекції можуть бути співставлені з супутниковими картами для уточнення масштабу сцени або визначення місця події.

2. Класифікація місцевості – визначення типу середовища (урбаністичне, промислове, польове) для звуження пошукової зони під час геолокації.

3. Виділення ключових ознак (feature extraction) – використовується для пошуку подібних об'єктів на інших знімках або для автоматичного

порівняння кадрів із картографічними даними.

4. Трекінг об’єктів – визначення траєкторії руху, що дозволяє співставляти її з реальною топографією місцевості.

5. Сегментація сцен – формування масок об’єктів для точного аналізу рельєфу або структурних елементів місцевості.

Ці задачі формують інформаційну основу GEO OSINT, у якій CV-компоненти виступають первинним інструментом перетворення необроблених медіаданих на геоаналітичні ознаки.

У сучасному GEO OSINT широко використовуються оптимізовані моделі CV, здатні працювати на обмежених обчислювальних ресурсах або у польових умовах. Серед найбільш поширених.

– YOLO-сімейства (v5–v9) – застосовуються для детекції транспортних засобів, техніки, людей; моделей вистачає для розпізнавання об’єктів на відео з мобільних камер або дронів.

– EfficientDet – підходить для аналізу великих знімків, у тому числі супутникових.

– CLIP / OpenCLIP – дозволяє порівнювати зображення з текстовими описами, що є корисним для OSINT-розвідки («знайди схожий завод», «міст схожої форми»).

– Siamese Networks – застосовується для пошуку відповідностей між різними фрагментами зображень і геолокації за структурою сцени.

– Vision Transformers (ViT) – забезпечують покращену якість класифікації складних сцен, що мають хаотичну структуру.

Оптимізація моделей у формат ONNX та їх подальша конвертація через TensorRT дозволяє розгортати їх на апаратних платформах без GPU (як у системі Amanita). Це розширює можливості використання CV-модулів як складових GEO OSINT-процесів без необхідності в хмарних обчисленнях.

Середовище розгортання систем комп’ютерного бачення має критичне значення для GEO OSINT, оскільки якість аналізу медіаданих прямо залежить

від стабільності обчислювального конвеєра. Такі параметри, як узгоджені версії бібліотек, відтворюваність процесу обробки, наявність централізованих логів і стійкість до збоїв, визначають достовірність аналітичних результатів.

Автоматизоване розгортання, реалізоване в системі Amanita, забезпечує контрольованість середовища: незалежно від кількості вузлів конфігурація є відтворюваною, а поведінка модулів – передбачуваною. Це є ключовою вимогою для GEO OSINT-систем, у яких необхідно гарантувати, що одна й та сама сцена при повторній обробці дає однакові структуровані дані для подальшої геолокації, зіставлення або аналізу.

1.3. Багатоканальна оптична система виявлення та супроводу рухомих об’єктів Amanita

Подібні багатоканальні оптичні комплекси для виявлення та супроводу рухомих об’єктів розглядаються у роботах [1; 7]. Багатоканальна оптична система виявлення та супроводу рухомих об’єктів Amanita (далі – система Amanita) є прикладом розподіленого багатокомпонентного комплексу комп’ютерного бачення, що реалізує повний цикл виявлення, супроводу та підсвітки рухомих об’єктів у реальному часі. Вона об’єднує модулі відеозахоплення, аналітичної обробки, керування виконавчими механізмами та центрального управління в одну узгоджену архітектуру, призначену для роботи як на фізичних пристроях, так і у віртуальному середовищі.

Amanita підтримує масштабовану топологію: кілька камер і турелей можуть бути розміщені на окремих вузлах локальної мережі, а процеси координації централізовано контролюються через модуль Manager. Це дозволяє адаптувати систему до різних сценаріїв – від стаціонарного нагляду до мобільного розгортання на наземних чи повітряних платформах.

Важливо, що система не лише виконує виявлення й супровід цілей, а й потребує узгодженого конфігурування численних програмних компонентів, їх

розміщення на вузлах, запуску в заданій послідовності та контролю стану. Саме ці вимоги зумовлюють необхідність автоматизації розгортання, яка реалізована у межах даної роботи – на основі декларативного опису структури системи, генерації конфігурацій і централізованого керування.

Таким чином, Amanita є типовим кейсом для впровадження підходів DevOps та Infrastructure as Code в системах комп'ютерного бачення з підвищеними вимогами до точності, надійності та відтворюваності.

Система Amanita може розглядатися не лише як окремий комплекс комп'ютерного бачення, але й як технологічний компонент у ширших GEO OSINT-процесах. Оскільки GEO OSINT працює з аналізом медіафайлів, геометрії сцени та просторових характеристик об'єктів, дані, які генерує Amanita, можуть бути інтегровані у геопросторові моделі або використовуватися для підтвердження місця події.

Зокрема, результати детекції, супроводу та оцінки траєкторій об'єктів, отримані у модулях camerapro та datapro, можуть бути зіставлені з відкритими картографічними джерелами або супутниковими знімками. Це дозволяє визначити відповідність між реальною сценою та потенційною локацією, що є ключовим завданням GEO OSINT при геолокації відео чи фотографій.

Структуровані дані про напрям руху, швидкість об'єктів, їхню відносну позицію та характерні візуальні ознаки можуть бути використані як геоаналітичні індикатори. Наприклад, видимість певних інфраструктурних елементів (доріг, перехресть, промислових споруд) у полі зору камери може бути співставлена з топологією місцевості на картах OpenStreetMap або у супутникових сервісах. Така комбінована аналітика дозволяє встановлювати можливі зони розташування камер або підтверджувати достовірність відеоматеріалу, що є важливою задачею в OSINT-розслідуваннях.

Для якісного GEO OSINT-аналізу критично важливим є забезпечення узгодженості результатів комп'ютерного бачення. Якщо відмінності у версіях моделей, бібліотек або конфігурацій призводять до зміни детекцій чи геометрії

об’єктів, це може вплинути на точність подальшого геопросторового аналізу. Тому автоматизація розгортання, описана у роботі, забезпечує не лише стабільність функціонування системи Amanita, але й важливу властивість – відтворюваність результатів, що є основою достовірного GEO OSINT.

1.4. Сучасні підходи автоматизації систем комп’ютерного бачення

У багатокомпонентних комп’ютерних системах, що працюють на декількох вузлах або у віртуальному середовищі, актуальним завданням є автоматизація процесів збирання, конфігурування, встановлення та керування компонентами. Зокрема, це стосується систем комп’ютерного бачення (CV), де модулі обробки зображень, прийняття рішень і керування пристроями функціонують у тісній взаємодії. Автоматизація дозволяє зменшити залежність від ручних дій, підвищити повторюваність процесів і забезпечити масштабованість розгортання.

У роботах [3, 8, 9] показано, що сучасні рішення з автоматизації розгортання ґрунтуються на підходах DevOps та концепції Infrastructure as Code (IaC), згідно з якою вся конфігурація системи описується у вигляді коду, що зберігаються у системах контролю версій. У контексті систем комп’ютерного бачення розрізняють такі підходи до автоматизації розгортання як сценарна автоматизація, шаблонна конфігурація та IaC-підхід (Таблиця 1.1.).

Сценарна автоматизація**:** інсталяція виконується через послідовність команд оболонки (Shell, PowerShell) або окремі скрипти, що запускаються вручну [2; 12]. Цей підхід простий у реалізації та ефективний для обмежених середовищ. У системі Amanita він реалізований у модулях Installer і Manager.

Шаблонна конфігурація**:** конфігураційні файли описуються у форматах JSON, YAML або XML, а системи розгортання підставляють значення змінних під час виконання [2; 9]. У Amanita за це відповідає за це відповідає компонент Configurator на Python.

IaC-підхід**:** декларативне описання інфраструктури з фіксацією змін у системі контролю версій (Git) дозволяє відтворювати середовище автоматично, без ручних втручань адміністратора [8; 10]. У запропонованій системі це реалізується у вигляді зв'язку: YAML-конфігурації - генерація опису системи – Bash-інсталяція.

Оркестрація**:** централізоване керування процесами, як у Kubernetes. Для малоресурсних систем комп'ютерного бачення цей підхід є надмірним, тому в Amanita реалізовано спрощений варіант моніторингу – через менеджер.

CI/CD: Як зазначено у роботах [9; 11; 12], CI/CD поєднує автоматизовану збірку, перевірку та доставку коду, що забезпечує швидке внесення змін і зменшення ризику помилок. У проєкті Amanita окремі етапи CI/CD реалізовані через локальні скрипти.

Таблиця 1.1 Порівняльна характеристика підходів до автоматизації розгортання

Підхід	Переваги	Обмеження
Bash-сценарії	Простота, мінімальні залежності	Відсутність структури
Шаблони YAML/JSON	Гнучкість, валідність	Потребує логіки генерації
IaC	Версійність, контроль	Поріг входу, складність
CI/CD	Повна автоматизація	Залежність від SCM та середовища
Оркестрація	Висока стійкість	Надмірна складність

Реалізована система Amanita поєднує сценарний та декларативний підходи: використовується Python для генерації конфігурацій і Bash для встановлення та керування процесами. Це дозволяє забезпечити автоматизацію без складних зовнішніх інструментів, що особливо важливо для CPU-

орієнтованих систем комп’ютерного бачення. Такий підхід виявився ефективним у забезпеченні стабільного розгортання, гнучкого масштабування та контролю у віртуальному середовищі без потреби в контейнерних оркестраторах.

У контексті GEO OSINT автоматизація процесів розгортання та конфігурації систем комп’ютерного бачення має ще важливіше значення, ніж у звичайних технічних застосуваннях. Це пов’язано з тим, що кінцевим продуктом GEO OSINT-аналізу є не просто набір результатів детекції чи класифікації, а аналітичні висновки, які використовуються у журналістських розслідуваннях, наукових дослідженнях або у судових процесах. У таких випадках достовірність та відтворюваність даних є критичною вимогою.

Автоматизоване розгортання гарантує, що кожен модуль системи Amanita функціонує у строго визначеному середовищі, з однаковими версіями бібліотек, параметрами моделей та конфігураціями апаратних ресурсів. Це усуває ризики появи різних результатів при повторному аналізі і забезпечує надійність даних, які надалі можуть бути інтегровані у геопросторові моделі або використані для підтвердження місця події.

Відтворюваність є базовою характеристикою будь-якого сучасного GEO OSINT-процесу. Якщо система, що обробляє відеодані, дає різні результати залежно від умов середовища або конфігурації, це робить неможливим використання таких даних у розслідуваннях, де кожен крок повинен бути обґрунтований та перевірений незалежною стороною.

Автоматизація дозволяє забезпечити повну контрольованість:

– повторне розгортання системи на іншій машині дає однакову поведінку;

– усі залежності зафіксовані у конфігураційних файлах;

– оновлення виконуються прогнозовано та перевіряються тестами;

– будь-які змінені параметри легко простежити.

У поєднанні з можливостями системи Amanita це створює умови для формування надійного конвеєра опрацювання медіаданих для геолокаційних

досліджень та аналізу подій.

Ручне розгортання систем комп'ютерного бачення у багатокомпонентних комплексах, таких як Amanita, є трудомістким, схильним до помилок і практично не масштабованим. Для GEO OSINT, де однотипні середовища повинні бути розгорнуті на кількох вузлах для порівняльного аналізу або для обробки великих обсягів даних, такі підходи є неефективними.

Автоматизоване розгортання забезпечує:

– швидке відтворення середовища на багатьох пристроях;

– гарантовану ідентичність конфігурації;

– можливість централізованого оновлення;

– контроль версій і можливість відкату;

– значне зменшення ризику людського фактора.

У результаті системи, що працюють у GEO OSINT-сценаріях, отримують стабільну інфраструктуру, яка дозволяє зосередитися на аналізі даних, а не на усуненні технічних збоїв.

1.5. Інфраструктура як код та DevOps-підходи в Amanita

У сучасній практиці автоматизації систем ключовим підходом до керування інфраструктурою є концепція Infrastructure as Code (IaC). Вона передбачає опис конфігураційних параметрів середовища – мереж, вузлів, сервісів – у вигляді декларативних конфігурацій, що зберігаються в репозиторії разом із кодом [7; 8; 13]. Це забезпечує можливість відтворюваного налаштування системи, контроль версій, автоматичне тестування та масштабування без участі користувача.

У системі Amanita ці принципи реалізовані через YAML-конфігурації, які описують параметри всіх вузлів – IP-адреси, ролі, шляхи до ресурсів, залежності тощо. На основі YAML-файлів Python-скрипт формує index.json – файл-індекс, що служить джерелом для подальшої автоматичної інсталяції.

Автоматизація розгортання в Amanita ґрунтується на принципах DevOps. Після зміни коду:

– запускається автоматичне збирання (build);

– генерується конфігурація інсталятора;

– виконуються перевірки (тестування);

– система самостійно деплоїть модулі на відповідні вузли.

Процеси організовано за схемою CI/CD (безперервної інтеграції та доставки), що дозволяє досягти таких результатів:

– скорочення часу на оновлення системи;

– мінімізація ручного втручання;

– швидке відновлення працездатності у разі помилок;

– стабільне масштабування без повторної ручної конфігурації.

Таким чином, реалізована у проєкті Amanita архітектура відповідає ключовим практикам IaC і DevOps, забезпечуючи відтворюваність розгортання, спрощене оновлення та стабільність функціонування у мультивузловому середовищі.

РОЗДІЛ 2. АРХІТЕКТУРА СИСТЕМИ

2.1. Загальні принципи архітектури системи Amanita

Архітектура розробленої системи автоматизації розгортання для Багатоканальної оптичної системи виявлення та супроводу рухомих об’єктів Amanita побудована за модульним принципом, що забезпечує розділення обов’язків, повторне використання компонентів та спрощення обслуговування. Кожен модуль відповідає за окремий етап життєвого циклу системи – від збирання вихідного коду до інсталяції, запуску і контролю.

У роботах [12; 13] описано типові підходи до модульного проєктування інфраструктурних рішень. На відміну від них, Amanita орієнтована на ізольовані середовища без контейнерних оркестраторів (напр. Kubernetes) і передбачає локальну автоматизацію через SSH та скрипти.

Основні компоненти системи Amanita є модулі Builder, Configurator, Configurator, Manager.

Builder – виконує компіляцію вихідного коду системи у контрольованому середовищі (Docker), формуючи архіви з бінарними файлами та бібліотеками.

Configurator – генерує конфігураційні файли на основі YAML-опису системи (topology.yaml, nodes.yaml), зберігаючи інформацію про вузли, їх ролі, IP-адреси, шляхи до файлів тощо.

Installer – автоматично копіює артефакти на цільові вузли, створює структуру директорій, перевіряє контрольні суми, запускає скрипти встановлення.

Manager – координує запуск і контроль виконання модулів на вузлах, веде журнали, реагує на помилки та автоматично перезапускає сервіси.

Як показано у працях [5; 13], використання трирівневої архітектури дозволяє розділити відповідальність між рівнями представлення, бізнес-логіки та доступу до даних. Всі ці модулі функціонують у рамках трирівневої архітектури, що забезпечує гнучкість і незалежність:

1. інфраструктурний рівень – апаратні ресурси (сервери, віртуальні машини), які є базою для розгортання;
2. системний рівень – операційні системи вузлів, конфігурація мережі, механізми SSH-доступу, інструменти віртуалізації (Docker);
3. прикладний рівень – основні логічні модулі Builder, Configurator, Installer, Manager.

Таке розмежування дозволяє окремо тестувати і масштабувати компоненти. Наприклад, Builder можна запускати лише у контейнері без доступу до реального обладнання, що прискорює цикл розробки.

Взаємодія компонентів. Комунікація між модулями відбувається через стандартизовану файлову структуру, у якій кожен компонент має власну директорію в каталозі /opt/clt2-optic/modules/. Там зберігаються:

– виконувані бінарні файли;

– конфігураційні YAML-файли;

– журнали роботи (/var/log/clt2-optic/);

– службові звіти про помилки та статус інсталяції.

Обмін даними між вузлами системи здійснюється по SSH, що забезпечує надійність та безпеку навіть у закритих ізольованих мережах.

Компонент Manager виконує моніторинг стану процесів на вузлах, у разі збою записує повідомлення до логів і видає код завершення. Це дозволяє Installer-у реагувати автоматично: повторити спробу, повідомити інженера або змінити конфігурацію. Схематичне зображення загальної архітектури системи автоматизації подано у на рисунку 2.1.

Архітектура системи комп’ютерного бачення відіграє ключову роль у можливості інтеграції її компонентів у GEO OSINT-процеси. Для сучасних систем OSINT, що працюють із великою кількістю медіаданих, важливою є здатність обробляти потоки відео паралельно, забезпечувати стабільну передачу ознак сцени та надавати структуровані результати для подальшої геоаналітики. Тому архітектурні рішення системи Amanita – включаючи модульність,

ізоляцію функціональних блоків та стандартизовані інтерфейси обміну даними – є критично важливими для побудови розширюваної платформи GEO OSINT.

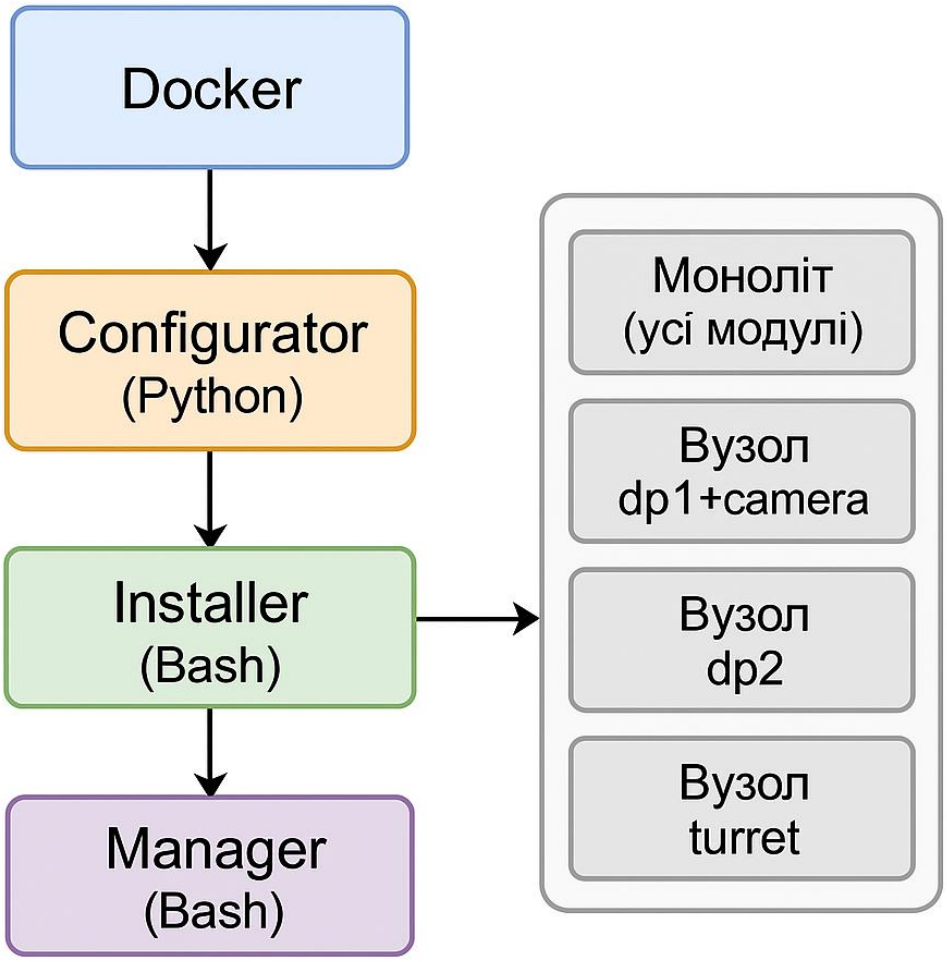

Рисунок 2.1 – Загальна архітектура системи автоматизації розгортання Amanita

2.2. Файлова організація та схема роботи

Ефективна робота системи Amanita базується на узгодженій файловій структурі та чітко визначеній послідовності виконання дій. Усі компоненти взаємодіють через загальні конфігураційні файли, журнали та артефакти, що зберігаються у стандартизованих директоріях.

Структура репозиторіїв. Усі вихідні коди, конфігураційні шаблони та

скрипти розміщено в двох основних репозиторіях:

– Configurator Repository – включає:

– Bash-скрипти для встановлення та перевірки цілісності артефактів;

– Python-модулі для обробки YAML-опису;

– підмодуль clt2-optic з вихідними кодами основної логіки CV-системи.

– clt2-optic Repository – реалізує ключові програмні модулі для:

– обробки відеопотоку;

– виявлення та супроводу об'єктів;

– керування механізмами підсвітки та наведення.

Файлова структура узгоджена між усіма вузлами, що дозволяє уникати конфліктів та пришвидшує інсталяцію. На кожному вузлі створюється єдина робоча директорія /opt/clt2-optic/, де зберігаються:

– виконувані модулі (bin/);

– YAML-конфігурації (config/);

– журнали (/var/log/clt2-optic/);

– службові артефакти (temp/, artifacts/).

Централізація структури гарантує портативність і повторюваність інсталяцій незалежно від апаратної конфігурації.

Логіка та послідовність виконання розгортання системи. Автоматизоване розгортання системи реалізовано як послідовний ланцюг дій, який охоплює весь життєвий цикл системи – від збирання до контролю роботи.

1. Builder компілює вихідний код із clt2-optic у середовищі Docker і створює архів артефактів .tar.gz з модулями, бібліотеками та службовими файлами.

2. Configurator зчитує YAML-конфігурацію (nodes.yaml, topology.yaml), перевіряє синтаксис і формує агрегований файл index.json – централізовану модель розгортання.

3. Installer за допомогою SSH копіює артефакти на вузли, створює потрібну структуру директорій, встановлює конфігураційні файли та запускає

локальні інсталяційні скрипти.

4. Manager ініціалізує й контролює модулі на кожному вузлі. Він відповідає за запуск, зупинку, моніторинг процесів і перезапуск у разі збою.

5. Усі кроки супроводжуються логуванням у /var/log/clt2-optic/, що забезпечує аудит та можливість аналізу у випадку помилок.

Цей конвеєр реалізує локальну модель CI/CD, де кожен етап є окремим автоматизованим модулем. Такий підхід забезпечує стабільність, гнучкість, відтворюваність і прискорює розгортання навіть у середовищах без підтримки повноцінних DevOps-інструментів.

Модульна структура системи Amanita природно відповідає вимогам GEO OSINT-платформ, які часто включають в себе кілька етапів: збір даних, їх попередня обробка, виділення ознак, геопросторова прив’язка та формування аналітичних висновків. Завдяки чіткій сегментації на модулі camerapro, datapro1/2, turretpro та manager система дозволяє інтегруватися у зовнішні аналітичні інструменти, що здійснюють геолокацію, зіставлення сцен та перевірку автентичності медіаданих.

Подібна архітектура дозволяє не тільки масштабувати систему, але й адаптувати її до різних дослідницьких сценаріїв: від спостереження за рухомими об’єктами до аналізу просторових патернів у відеопотоці.

2.3. Мережева модель і конфігурування системи

Система Amanita підтримує гнучку мережеву архітектуру, що адаптується до різних сценаріїв розгортання – від локальних тестових середовищ до повноцінних розподілених конфігурацій.

Передбачено два основних режими розгортання. Монолітний режим – усі компоненти (Builder, Configurator, Installer, Manager, CV-модулі) розгортаються на одному пристрої. Цей варіант використовується для локального тестування, налагодження нових збірок або невеликих інсталяцій.

Розподілений режим – модулі розгортаються на окремих вузлах мережі, що дозволяє:

– розвантажити процесорні ресурси;

– розділити функціональність (наприклад, окремий вузол для відеопотоку, обробки даних або керування тureллю);

– масштабувати систему під більшу кількість пристроїв спостереження.

Топологія визначається декларативно – через YAML-конфігурацію (nodes.yaml), де описано:

– список вузлів;

– IP-адреси;

– ролі (наприклад, camerapro, datapro, turretpro, manager);

– шляхи до конфігураційних файлів;

– необхідні артефакти для кожного вузла.

Модель конфігурування. Конфігурація системи побудована на принципах DevOps і Infrastructure as Code (IaC) – увесь опис інфраструктури та ролей зберігається як код. Це дозволяє:

– уніфікувати налаштування;

– легко масштабувати систему;

– забезпечити повторюваність розгортання;

– мінімізувати ручне втручання.

Ключовим елементом є вузол (node), який описується в YAML полями, що вказані в таблиці 2.1.

Таблиця 2.1. Поля та призначення вузлів

Поле	Призначення
name	Унікальне ім’я вузла
ip	IP-адреса для SSH-з’єднання
role	Логічна роль (наприклад, camerapro, manager)

Поле	Призначення
artifacts	Список бінарних файлів, що мають бути встановлені
config_path	Шлях до конфігурацій на цільовому вузлі

На основі YAML-файлів Configurator генерує агрегований файл index.json, який є основною моделлю для:

– встановлення модулів (Installer);

– визначення зв’язків між вузлами;

– автоматичного запуску потрібних сервісів (Manager).

Цей підхід дозволяє вносити зміни до топології без жодних змін у коді – достатньо змінити YAML-файл і повторити конфігурацію. Завдяки цьому система є гнучкою, стабільною та зручною для CI/CD-розгортання.

У контексті GEO OSINT мережева модель системи Amanita відіграє значно ширшу роль, ніж просто транспорт для передачі відеопотоків між компонентами. Вона формує основу для побудови геоаналітичного конвеєра, у якому стабільність, відтворюваність та узгодженість конфігурацій безпосередньо впливають на точність подальшої аналітики та геопросторової інтерпретації даних.

Системи GEO OSINT працюють із великою кількістю різнорідних джерел: польові камери, мобільні сенсори, безпілотники, стаціонарні комплекси спостереження, супутникові дані. Тому ключовими характеристиками мережевої моделі є:

– узгоджений часовий контекст даних, що дозволяє порівнювати відеопотоки з різних вузлів та зіставляти їх із джерелами геолокації;

– стабільність каналів передачі, яка визначає цілісність та послідовність кадрів;

– можливість масштабування, необхідна при збільшенні кількості сенсорів на території спостереження;

– уніфіковані конфігураційні параметри, що забезпечують ідентичну поведінку модулів при повторному розгортанні.

У системі Amanita конфігураційна модель, описана у YAML-файлах та агрегована у index.json, дозволяє формувати чітку топологію мережевих зв'язків між модулями camerapro, datapro1/2, turretpro та manager. Це є критично важливим для подальшої інтеграції в GEO OSINT-платформи, де дані можуть використовуватися для:

– автоматизованої геолокації відео;

– відстеження траєкторій руху об'єктів;

– зіставлення подій із супутниковими знімками;

– визначення відповідності сцени реальному геопросторовому контексту.

У разі зміни топології (наприклад, додавання нових камер або сенсорів) достатньо оновити YAML-файл, після чого система автоматично перебудовує інфраструктуру взаємодії. Такий підхід дозволяє швидко адаптувати систему до польових умов, що є характерним для OSINT-розслідувань або розгортання мобільних аналітичних пунктів.

Окрім забезпечення зв'язності, мережева модель Amanita виконує роль єдиного джерела правди щодо конфігурації, що є стандартною вимогою у GEO OSINT-проєктах. Це дає змогу:

– гарантувати, що дані з різних вузлів синхронізовані між собою;

– виключити ризик «дрейфу конфігурацій», коли різні вузли працюють у різних середовищах;

– забезпечити повну відтворюваність системи при повторному розгортанні або аудиті;

– інтегрувати дані Amanita у зовнішні геоаналітичні модулі.

Таким чином, мережева модель Amanita виступає не просто технічним механізмом зв'язку, але й ключовим елементом побудови масштабованого GEO OSINT-комплексу, що здатний забезпечити достовірність, точність та

синхронність медіаданих, необхідних для геопросторових досліджень.

2.4. Функціональні та нефункціональні вимоги

Для стабільної роботи системи Amanita та забезпечення її масштабованості, відтворюваності і автоматизації висуваються наступні функціональні й нефункціональні вимоги. Ці вимоги сформульовано з урахуванням двох основних ролей користувачів системи: інженера, який відповідає за розгортання та налагодження комплексу, і спостерігача, який працює з уже налаштованою системою та аналізує результати спостереження.

Функціональні вимоги. Система повинна забезпечувати:

1. Декларативне конфігурування:

– опис топології системи у YAML-форматі;

– вказання IP-адрес, логічних ролей вузлів (camerapro, datapro, turretpro, manager);

– параметри доступу та шляхи до конфігурацій.

2. Формування моделі конфігурації:

– генерація index.json на основі YAML-файлів;

– валідація структури даних перед встановленням.

3. Автоматизоване збирання програмних модулів:

– використання Docker-контейнера для компіляції;

– формування архівів артефактів .tar.gz.

4. Передавання та встановлення компонентів:

– копіювання артефактів через SSH;

– автоматичне створення структури каталогів та копіювання конфігурацій.

5. Керування життєвим циклом модулів:

– запуск, зупинка й перезапуск через Manager;

– моніторинг стану та логування у /var/log/clt2-optic/.

6. Підтримка двох режимів роботи:

– монолітного – усі модулі на одному вузлі;

– розподіленого – кожен модуль на окремому пристрої.

7. Інтеграція з механізмом Factory pull/install:

– отримання збірок із зовнішнього сховища;

– локальна інсталяція без ручної конфігурації.

8. Журналювання процесів розгортання:

– зберігання логів усіх операцій;

– підтримка аналізу та повторного відтворення процесів.

Нефункціональні вимоги

1. Портативність:

– сумісність із ОС Linux (Ubuntu 20.04+, Debian 12);

– повноцінна робота без графічного інтерфейсу.

2. Відтворюваність:

– однаковий результат інсталяції на різних стендах при використанні тих самих YAML-файлів.

3. Автоматизація:

– мінімізація участі людини;

– виконання всіх операцій через скрипти, без ручних налаштувань.

4. Масштабованість:

– зміна кількості вузлів, ролей і конфігурацій без зміни програмного коду.

5. Стійкість до збоїв:

– логування помилок;

– підтримка повторного запуску у разі часткових збоїв або втрати зв’язку.

6. Тестованість:

– розгортання у віртуальному середовищі (Vagrant);

– можливість перевірки конфігурацій перед інсталяцією на реальні

пристрої.

Перелік функціональних та нефункціональних вимог було уточнено за результатами експертного тестування, проведеного Ю. Нетребіним. Під час цього тестування сформовано таблицю виявлених дефектів та рекомендацій, що стосуються формату конфігураційних файлів, структури журналів, перевірки SSH-доступу та узгодженості інсталяційних сценаріїв. Отримані зауваження були враховані при формуванні остаточної моделі вимог до системи Amanita.

Надійність конфігураційних даних та середовища:

– уніфікований формат файла index.json, сумісний між усіма компонентами системи;

– попередня перевірка наявності SSH-ключів та коректності прав доступу перед розгортанням;

– використання виділених директорій для конфігураційних файлів і журналів модулів (/opt/clt2-optic/, /var/log/clt2-optic/);

– узгоджені назви вузлів та ролей у конфігураційних файлах, Vagrant-сценаріях та супровідній документації.

2.5. Сценарії використання та реалізація

Для забезпечення гнучкості, відтворюваності та контролю процесу розгортання системи Amanita передбачено три основні сценарії використання, кожен з яких реалізує послідовну автоматизовану процедуру конфігурації та встановлення модулів. У межах кожного сценарію явно виділяються дві ролі користувачів: інженер, який готує та розгортає систему, і спостерігач, який працює з уже налаштованим комплексом та контролює процес спостереження.

Сценарій 1. Монолітне розгортання на одному вузлі. Цей режим використовується для локального тестування або прототипування. Кроки реалізації:

1. Інженер готує локальне середовище (Docker, SSH).
2. Модуль Builder компілює clt2-optic та формує бінарні файли.
3. Configurator генерує файл index.json на основі YAML-конфігурації.
4. Installer виконує інсталяцію артефактів на локальний вузол.
5. Manager активує модулі та здійснює моніторинг процесів.

Сценарій 2. Розподілене розгортання на кількох вузлах. Цей сценарій застосовується для продуктивного використання з розділенням ролей між пристроями. Кроки реалізації:

1. YAML-файл описує кожен вузол: IP-адреса, роль, конфігурація.
2. Configurator перевіряє структуру та створює index.json.
3. Installer копіює файли через SSH на відповідні вузли.
4. Manager запускає модулі, збирає логи та контролює працездатність.
5. Зміна топології потребує лише редагування YAML без модифікації коду.

Сценарій 3. Тестування у віртуальному середовищі (Vagrant). Призначено для перевірки конфігурацій та тренування перед продуктивним розгортанням. Кроки реалізації:

1. Інженер створює Vagrantfile з описом віртуальних машин та їхніх ролей.
2. Система виконує повний CI-потік Builder – Configurator – Installer – Manager.
3. Спостерігач перевіряє журнали та підтверджує коректність.

Загальна послідовність реалізації сценаріїв. Незалежно від режиму, реалізація розгортання передбачає наступну етапність:

1. Підготовка конфігурацій:

– створення YAML-опису топології;

– налаштування SSH-доступу.

2. Збирання артефактів:

– компіляція через Builder;

– створення архівів .tar.gz.

3. Генерація конфігурації:

– перевірка YAML;

– формування index.json.

4. Інсталяція на вузли:

– передача артефактів;

– запуск підготовчих скриптів.

5. Керування модулями:

– запуск, контроль, перезапуск;

– логування подій.

6. Архівування результатів:

– зберігання журналів у /var/log/clt2-optic/;

– підтримка аудиту та відтворення.

2.6 UML-діаграми використання системи

На рисунку 2.2 зображено прецеденти взаємодії інженера та спостерігача із системою Amanita під час розгортання та моніторингу.

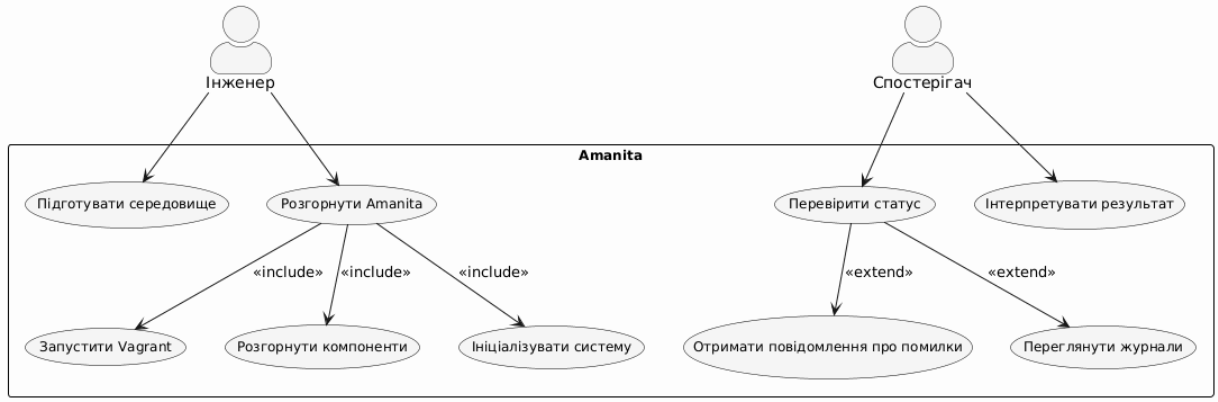

Рисунок 2.2 – Діаграма взаємодії користувачів із системою Amanita

На рисунку 2.3, дії інженера виконуються у визначеній послідовності: запуск Vagrant, генерація конфігурацій, інсталяція компонентів та запуск системи.

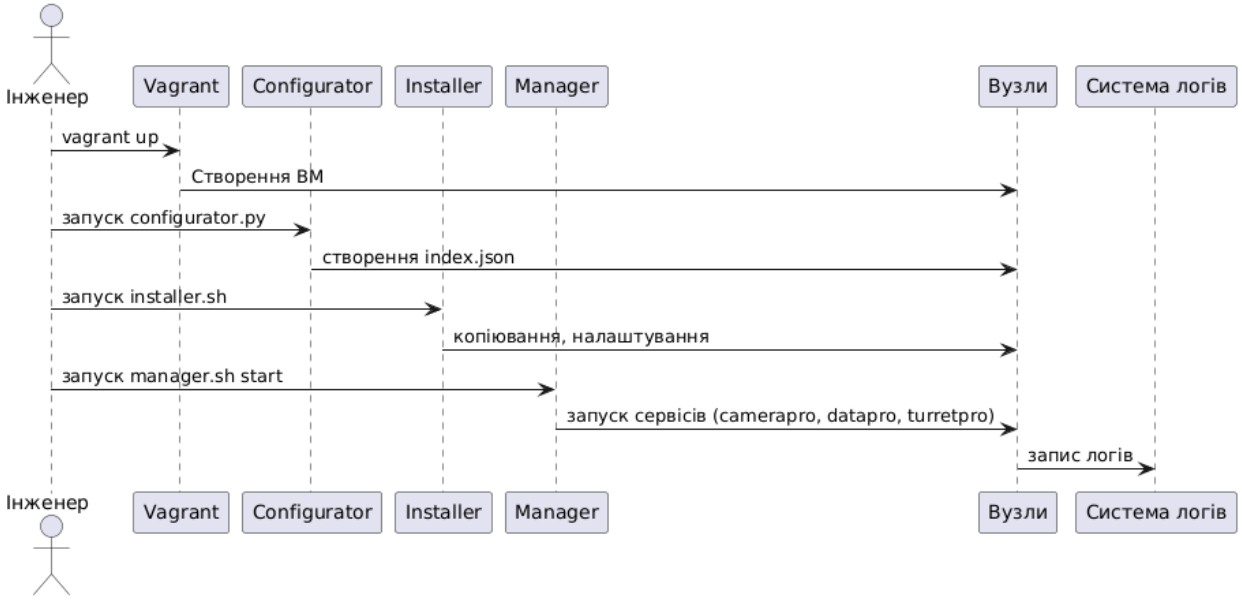

Рисунок 2.3 – Діаграма послідовності дій інженера під час розгортання системи Amanita

UML-діаграми використання застосовуються для формального опису функціональних можливостей програмної системи з точки зору зовнішніх користувачів та інших суб’єктів взаємодії. На відміну від структурних UML-діаграм, що відображають внутрішню організацію програмних компонентів, діаграми використання зосереджені на сценаріях роботи системи та дозволяють узгодити функціональні вимоги із реальними потребами користувачів. В роботі UML-діаграми використання застосовуються для систематизації сценаріїв автоматизованого розгортання та моніторингу багатокомпонентної системи Amanita.

Побудова UML-діаграм використання для системи автоматизації розгортання дозволяє формалізувати ролі користувачів, визначити межі

відповідальності системи, а також забезпечити логічний зв’язок між вимогами, сценаріями використання та реалізованими програмними модулями. Це є особливо важливим у контексті DevOps-підходів та Infrastructure as Code, де процеси розгортання мають бути відтворюваними, контрольованими та мінімізувати ручне втручання.

У системі Amanita виділяються декілька основних акторів, які взаємодіють із системою у процесі її експлуатації. Основним актором є інженер, який відповідає за підготовку конфігураційних файлів, ініціацію процесів автоматизованого збирання та розгортання, контроль виконання сценаріїв і аналіз журналів роботи. Інженер здійснює всі дії, пов’язані з конфігуруванням топології системи, перевіркою доступності вузлів та усуненням помилок.

Другим актором є спостерігач, який працює з уже розгорнутою системою та не бере участі у процесах інсталяції або конфігурування. Його взаємодія з системою обмежується контролем стану модулів, аналізом результатів обробки відеопотоків і спостереженням за роботою комплексу. Такий розподіл ролей дозволяє відокремити експлуатаційні задачі від інженерних, що відповідає принципам розмежування відповідальності.

Окрім користувачів-людей, у UML-діаграмах використання також доцільно виділяти обчислювальні вузли як окремі актори. При цьому вузол розглядається як зовнішній суб’єкт, що приймає артефакти, виконує інсталяційні сценарії, запускає програмні модулі та передає інформацію про стан системи. Такий підхід дозволяє наочно відобразити розподілений характер системи Amanita та підкреслити роль автоматизованої взаємодії між компонентами.

Основні сценарії використання системи охоплюють повний життєвий цикл автоматизованого розгортання. До них належать підготовка конфігурацій системи, генерація агрегованої моделі розгортання, збирання програмних артефактів, автоматичне встановлення компонентів на цільові

вузли, запуск і моніторинг модулів, а також перегляд журналів та статусів виконання. Усі ці сценарії можуть виконуватися багаторазово при зміні конфігурації або топології системи, що забезпечує гнучкість і масштабованість рішення. Сценарії, пов’язані з автоматизованим встановленням і запуском системи, мають ієрархічну структуру. Так, сценарій автоматичного встановлення включає обов’язкові кроки попередньої генерації конфігурацій та підготовки артефактів, що відповідає відношенню <<include>> в UML. Водночас сценарії перегляду журналів або контролю стану модулів можуть виконуватися незалежно від інших дій і не потребують повторного розгортання системи.

На UML-діаграмах використання відображено, що більшість сценаріїв ініціюються інженером, однак виконуються автоматично самою системою без безпосередньої участі користувача. Це підкреслює ключову особливість розробленого підходу — мінімізацію ручних дій та перенесення основної логіки керування у програмні модулі Builder, Configurator, Installer та Manager.

UML-діаграми використання тісно пов’язані з архітектурою системи, описаною у попередніх підрозділах. Кожен сценарій використання має безпосередню відповідність конкретному програмному модулю або їх комбінації. Зокрема, сценарії збирання реалізуються модулем Builder, формування конфігурацій — модулем Configurator, процеси встановлення — модулем Installer, а запуск, моніторинг і контроль — модулем Manager. Така відповідність забезпечує узгодженість між вимогами користувачів та реалізованими архітектурними рішеннями. Таким чином, UML-діаграми використання системи Amanita виконують не лише описову, а й узгоджувальну функцію. Вони дозволяють формалізувати сценарії роботи, чітко визначити ролі акторів та забезпечити логічний зв’язок між функціональними вимогами, архітектурою системи та її практичною реалізацією. Це підвищує зрозумілість проєктних рішень і спрощує подальший супровід та розвиток системи автоматизованого розгортання.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ СИСТЕМИ

3.1. Загальні принципи реалізації

Реалізація системи автоматизації розгортання для багатоканальної оптичної системи виявлення та супроводу рухомих об’єктів Amanita виконана з урахуванням принципів простоти, модульності та надійності. Основна частина логіки системи побудована на сценаріях Bash, оскільки вони дозволяють безпосередньо керувати системними командами Linux та забезпечують мінімальні зовнішні залежності. Мова Python використовується лише для компонента Configurator, де потрібна обробка структурованих даних у форматі YAML/JSON. Підхід до створення конфігураційних шаблонів узгоджується з підходами, описаними у [15], де наголошується на важливості модульності та повторного використання YAML-структур. Решта компонентів – Installer та Manager – є Bash-скриптами, призначеними для виконання на будь-якій Unix-сумісній платформі без необхідності встановлення додаткових бібліотек. Послідовність етапів розгортання зображена на рисунку 3.1.

Реалізація системи Amanita враховує вимоги, характерні для GEO OSINT-проєктів, де критичне значення мають стабільність обробки медіаданих, відтворюваність результатів та здатність інтегруватися у ширші аналітичні конвеєри. На відміну від локальних систем комп’ютерного бачення, GEO OSINT-середовище передбачає використання результатів аналізу з різних джерел – камер, сенсорів, супутникових знімків – у єдиному інформаційному просторі. Саме тому реалізація Amanita орієнтована на модульність, стандартизованість і чітке розмежування відповідальності між компонентами. Такий підхід дозволяє розгортати систему в різних робочих середовищах без втрати функціональності, а також забезпечує можливість підключення Amanita до зовнішніх OSINT-інструментів, де отримані дані можуть бути використані для геолокації, верифікації подій або просторового аналізу поведінки об’єктів.

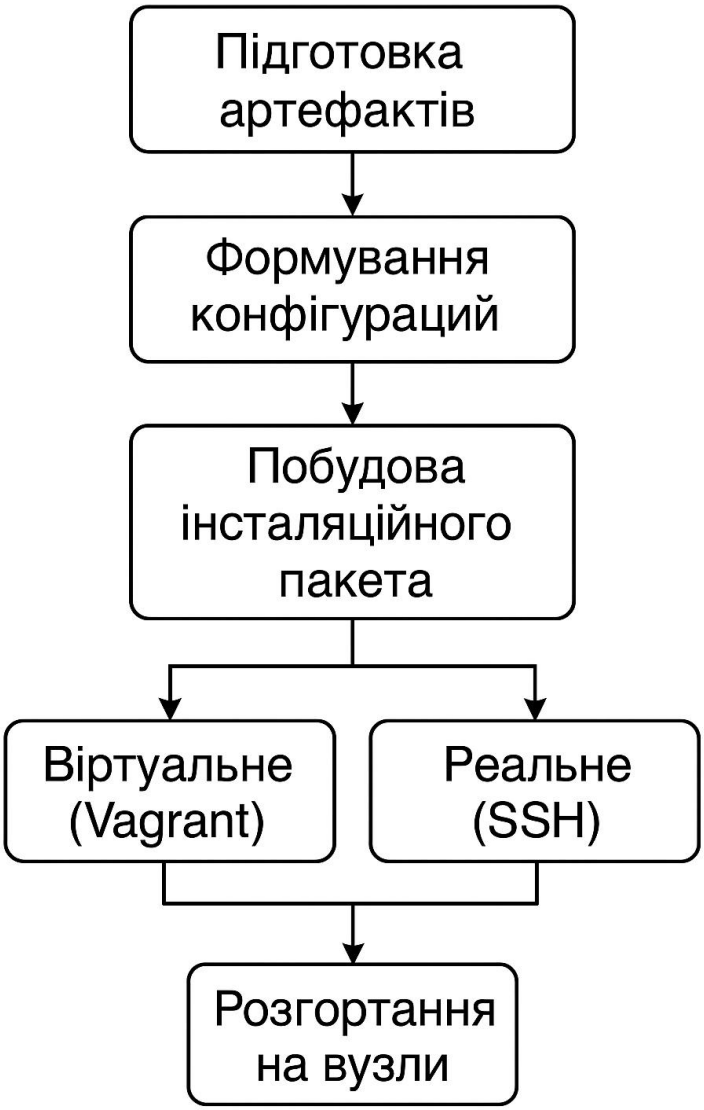

Рис. 3.1 – Технологічна схема автоматизації розгортання системи Amanita

3.2. Модулі Builder та Configurator

Модуль Builder створює виконувані файли системи комп'ютерного бачення з вихідних кодів підмодуля clt2-optic. Компіляція відбувається всередині контейнера Docker на базі попередньо підготовленого образу, який містить усі залежності (компілятор, OpenCV, FFmpeg, NumPy тощо). Builder формує архіви у форматі .tar.gz, що містять скомпільовані бінарні файли та службові

метадані для подальшого розгортання. Процес будівництва контрольований, усі журнали зберігаються в /var/log/clt2-optic/builder.log.

Модуль Configurator є єдиним компонентом, написаним мовою Python. Він зчитує структуру мережі з YAML-файлів, що описують вузли, їхні ролі та параметри. На основі цих даних формується JSON-файл index.json, який надалі використовують інші модулі. Configurator виконує валідацію вхідних даних, перевіряє унікальність імен вузлів та правильність IP-адрес. У разі помилки спостерігач отримує детальне повідомлення із номером рядка та описом проблеми. Завдяки Python вдається забезпечити гнучку роботу з форматами YAML/JSON та високу швидкість підготовки конфігурацій. Приклад конфігурацій наведено у Додатку В «Приклад конфігураційних файлів системи автоматизації».

У процесі експертного тестування було виявлено низку проблем, пов’язаних із формуванням файла index.json, зокрема несумісність структури між пілотними версіями configurator.py та скриптами розгортання, а також помилки у функції set_deep(), яка відповідала за створення вкладених конфігураційних ключів. Зауваження, зафіксовані Ю. Нетребіним у таблиці тестування, були враховані: логіка set_deep() доопрацьована для коректного формування вкладених структур, а формат index.json уніфіковано таким чином, щоб його поля однозначно інтерпретувалися усіма подальшими модулями системи.

У контексті програмного забезпечення GEO OSINT роль модулів Builder та Configurator набуває особливого значення, оскільки саме ці компоненти визначають відтворюваність, достовірність і стабільність подальших аналітичних процесів. У геоаналітичних системах, де результати комп’ютерного бачення використовуються для геолокації, картографування або реконструкції подій, важливо гарантувати, що бінарні файли та конфігураційні моделі створюються однаково на будь-якому середовищі.

Модуль Builder, виконуючи компіляцію всередині контрольованого Docker-образу, усуває залежність від локальних налаштувань розробника або

інженера. Це дозволяє отримати гарантовано однакові збірки навіть при повторному розгортанні на різних вузлах або в різних регіонах. Для GEO OSINT ця властивість критична, оскільки аналітичні висновки можуть бути використані для підтвердження або спростування геопросторових тверджень, що вимагає повної відтворюваності програмного середовища.

Модуль Configurator формує index.json як єдину модель топології системи, яка визначає, де і як саме обробляються відеодані. У GEO OSINT-платформах така модель виступає елементом метаданих, що дозволяє відстежити:

– які модулі брали участь у формуванні аналітичних даних;

– які параметри камер і обробки були застосовані;

– як саме дані були отримані, структуровані та передані.

Це створює можливість для аудиту та перевірки достовірності медіаінформації – ключової вимоги у OSINT-розслідуваннях, де будь-яка невідтворюваність може поставити під сумнів точність аналітичних висновків.

Таким чином, Builder та Configurator виконують не лише технічну функцію автоматизації, але й забезпечують фундаментальні властивості, необхідні для GEO OSINT-систем: прозорість, контрольованість і відтворюваність усіх етапів отримання структурованих медіаданих.

3.3. Модулі Installer та Manager

Модуль Installer відповідає за копіювання та інсталяцію бінарних файлів на цільові вузли. Його логіка реалізована у вигляді Bash-скрипту installer.sh, що виконує послідовність операцій:

1. Зчитування файла index.json, згенерованого Configurator.
2. Підключення до кожного вузла через SSH з використанням ключової автентифікації.
3. Передавання бінарних файлів за допомогою scp та їх розпакування.

4. Перевірка контрольних сум та наявності необхідних директорій.
5. Запис результатів до /var/log/clt2-optic/installer.log.

Окрему увагу приділено перевірці SSH-доступу. Перед початком копіювання артефактів Installer перевіряє наявність необхідних SSH-ключів і коректність прав доступу для кожного вузла. У разі відсутності ключа або помилки автентифікації інсталяція для відповідного вузла не розпочинається, в журнал /var/log/clt2-optic/installer.log вноситься діагностичне повідомлення. Під час тестування були зафіксовані проблеми з запуском інсталяційних сценаріїв від імені користувача root, після чого логіку скриптів було доопрацьовано таким чином, щоб забезпечити передбачувану поведінку як для звичайного користувача, так і для сценаріїв з підвищеними правами.

Модуль Manager виконує функції керування життєвим циклом усіх модулів. Його Bash-скрипт manager.sh зчитує дані з index.json і запускає відповідні бінарні файли на вузлах. Manager також перевіряє стан запущених процесів через systemctl, pgrep або ps, реєструє результати в логах та ініціює перезапуск модуля в разі завершення його роботи з помилкою.

Основні переваги Bash-реалізації:

– мінімум залежностей;

– пряма інтеграція з Linux-утилітами;

– прозорість виконання кожної команди;

– зручність налагодження на рівні системних журналів.

У системах типу GEO OSINT модулі Installer та Manager виконують значно ширшу роль, ніж просто технічні компоненти автоматизації розгортання. Installer забезпечує гарантовану відповідність робочого середовища між різними обчислювальними вузлами, що критично важливо під час аналізу медіаданих, які можуть надходити з камер із різними параметрами або розташованими у різних точках території. Будь-яка відмінність у версіях бібліотек, інстальованих пакетах чи середовищі виконання може призвести до різниці результатів обробки, що унеможливлює побудову достовірного геопросторового

аналізу.

Завдяки стандартизованим діям Installer забезпечує єдність всіх інстанцій системи Amanita. Це дозволяє отримувати відтворювані результати навіть тоді, коли аналіз медіаданих виконується на різних пристроях або повторюється через тривалий час. Для GEO OSINT-проєктів така передбачуваність є ключовою, оскільки саме вона дозволяє дослідникам співставляти результати, проводити часові дослідження або перевіряти достовірність джерел.

Модуль Manager виконує функцію координатора розподіленої аналітичної інфраструктури. У дослідженнях GEO OSINT нерідко використовується кілька джерел відео або сенсорів, і Manager забезпечує синхронізацію роботи модулів, послідовність дій та контроль повноти виконання сценаріїв. Завдяки цьому формуються цілісні медіапотоки, які можуть бути об’єднані у геопросторові моделі або використані часових та просторових реконструкцій подій.

Таким чином, Installer та Manager забезпечують не лише технічну працездатність системи Amanita, але й створюють необхідні умови для достовірності та відтворюваності даних – двох фундаментальних вимог GEO OSINT-аналізу.

3.4. Керування через командний рядок, логі та налагодження

Логіка керування через командний рядок. Для виконання операцій розгортання використовуються прості CLI-команди:

```
sudo ./builder.sh
sudo ./configurator.py --index ./configs/index.yaml
sudo ./installer.sh
sudo ./manager.sh start
```

Передбачено також команди зупинки та очищення:

```
sudo ./manager.sh stop
sudo ./manager.sh status
```

Таким чином, повний цикл розгортання реалізується послідовним виконанням чотирьох скриптів, а керування відбувається без будь-яких графічних інтерфейсів.

Інтеграція з віртуальним середовищем. Для тестування передбачено окремий Vagrant-файл, який створює віртуальні вузли з аналогічними IP-адресами та ролями, як у реальній системі. Після ініціалізації віртуального середовища виконується той самий ланцюг скриптів, що й у бойовому режимі. Це дозволяє перевірити коректність YAML-опису та загальну логіку інсталяції до моменту розгортання на фізичних пристроях.

Формат логів і налагодження. Кожен модуль створює власний лог-файл у каталозі /var/log/clt2-optic. Єдиний формат записів:

[YYYY-MM-DD HH:MM:SS] <LEVEL> <MODULE>: <message>

де <LEVEL> – DEBUG, INFO, WARN або ERROR, а <MODULE> – назва модуля. Приклади:

[2025-03-18 12:44:02] INFO installer: Copy to 10.9.0.32 completed.

[2025-03-18 12:45:10] ERROR manager: datapro1 exited with code 1. Restarting...

Усі скрипти додають часові мітки через date +"%F %T" і використовують одну функцію логування для єдності формату. Це спрощує аналіз помилок і забезпечує повну відтворюваність розгортання.

Особливої значущості механізми командного керування та логування набувають у контексті GEO OSINT-аналізу, де відтворюваність процесів і можливість точного аудиту виконаних дій відіграють ключову роль. Завдяки єдиному інтерфейсу керування та уніфікованій системі журналювання забезпечується можливість повністю простежити шлях даних – від моменту їх отримання на польовому сенсорі до формування структурованого результату. Така прозорість дозволяє включати систему до складних OSINT-конвеєрів, де важливо не лише отримати дані, але й гарантувати їхню достовірність, цілісність і повторюваність у будь-якому середовищі виконання.

РОЗДІЛ 4. ТЕСТУВАННЯ У ВІРТУАЛЬНОМУ СЕРЕДОВИЩІ

4.1. Мета тестування та тестове середовище

Метою тестування є перевірка працездатності системи автоматизації розгортання у різних умовах експлуатації – як у віртуальному середовищі, так і на реальному обладнанні. Основними критеріями оцінювання є: правильність виконання послідовності встановлення; стійкість до збоїв під час передачі даних; коректність формування конфігурацій; стабільність роботи після розгортання; відповідність структури каталогів вимогам системи. Усі скрипти додають часові мітки і використовують єдиний формат логів, що спрощує подальший аналіз за допомогою зовнішніх інструментів моніторингу (Grafana, Loki). Завдяки автоматизованому логуванню та єдиному формату даних забезпечується повна відтворюваність розгортання та збереження детальних звітів про кожен етап.

Для тестування застосовано два основних сценарії:

– монолітний сценарій: усі модулі (camerapro, datapro1, datapro2, turretpro, manager) розгорнуті на одній машині; цей режим перевіряє роботу системи в умовах повної автономії без взаємодії між вузлами;

– розподілений сценарій: модулі встановлені на трьох віртуальних вузлах: dp1 (camerapro + datapro1), dp2 (datapro2) та turretpro (контролер виконавчого модуля).

Віртуальні машини створені за допомогою Vagrant, що забезпечує ізольоване та повністю відтворюване середовище розгортання. Усі вузли перебувають у спільній підмережі (10.9.0.0/24) із однаковими ресурсними параметрами. Для віддаленого доступу використані SSH-ключі, обмін файлами відбувається через SCP. Схему тестового середовища наведено на рисунку 4.1. Для моніторингу роботи системи налаштовано збір метрик через Prometheus та їхню візуалізацію в Grafana. Для GEO OSINT-сценаріїв це означає перевірку того, що підготовка та передача медіаданих відбувається стабільно і може без

додаткових ручних втручань включатися до ланцюжків геоаналітичної обробки.

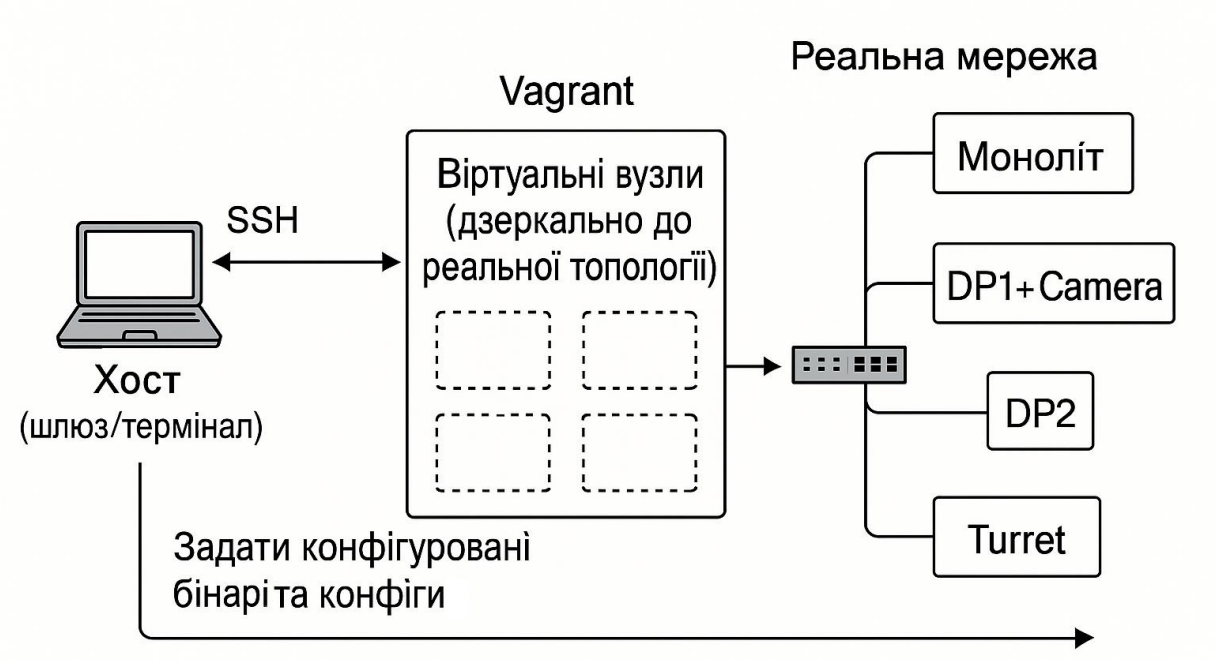

Рисунок 4.1 – Схема тестового середовища розгортання системи Amanita

4.2. Тестування автоматизованого розгортання

Підготовка до тестування. Перед початком перевірок підготовлено всі необхідні конфігурації та артефакти:

– Згенеровано YAML-файл із описом топології системи та параметрів вузлів.

– Створено файл index.json через компонент Configurator на основі YAML-опису.

– Підготовлено Docker-образ для модуля Builder із останньою версією вихідного коду системи.

– Перевірено доступність усіх вузлів за IP-адресами та встановлено ключі для SSH-підключення.

– Очищено каталог /var/log/clt2-optic/ для отримання чистих логів під час тестування.

Таким чином, підготовчий етап включав формування конфігураційних файлів (YAML/JSON), компіляцію артефактів та налаштування оточення Vagrant.

Тестування розгортання здійснювалося покроково в середовищі Vagrant. Основні етапи такі.

1. Запуск Vagrant: виконання vagrant up для створення та ініціалізації трьох віртуальних машин.
2. Формування конфігурацій: запуск configurator.py для зчитування YAML і генерації файлу index.json.
3. Розгортання артефактів: виконання скрипту installer.sh, який копіює архіви з бінарними файлами на цільові вузли та розпаковує їх.
4. Ініціалізація модулів: запуск manager.sh start, що ініціює роботу всіх компонентів на відповідних вузлах.
5. Перевірка результатів: аналіз логів та перевірка доступності сервісів і відповідності встановлених файлів конфігураційній моделі.

Усі дії автоматично фіксувалися у логах із єдиним форматом, що дало змогу відстежити кожний крок. В результаті розгортання всі вузли були налаштовані без ручного втручання, а тривалість процесу скоротилася приблизно у 4–5 разів порівняно з традиційним ручним розгортанням.

Тестування на реальних вузлах. Для підтвердження стабільності рішення розгортання було виконано на реальному обладнанні:

– Міні-ПК з Ubuntu (роль camerapro/dp1).

– Міні-ПК з Ubuntu (роль dp2).

– Міні-ПК з Ubuntu (контролер для модуля turretpro).

Після зміни IP-адрес у конфігураційному YAML-файлі на відповідні адреси фізичних машин були виконані ті ж самі скрипти розгортання. Розгортання завершилося успішно: усі модулі системи були ініціалізовані, а логи збереглися локально на кожному вузлі. Таким чином, автоматизована процедура коректно перенеслася з віртуального середовища на реальний кластер.

Порівняльна оцінка результатів. Порівняльна оцінка показала, що автоматизоване розгортання значно швидше за ручне. Для монолітної конфігурації (1 вузол) ручний процес займав близько 25 хвилин, тоді як автоматизований – лише ~7 хвилин (скорочення ≈72 %). У розподіленому сценарії (3 вузли) ручне розгортання триває ~45 хв, а автоматичне – ~12 хв (скорочення ≈73 %).

Крім зменшення часу, автоматизація забезпечила:

– повну повторюваність результатів розгортання;

– відсутність потреби у втручанні інженера під час копіювання файлів;

– автоматичне формування детальних звітів у каталозі /var/log.

Таким чином, система дозволяє швидко та надійно готувати мережевий кластер до роботи без людського фактору і зі збереженням усіх результатів для подальшого аналізу.

4.3. Тестування негативних сценаріїв

Окрім штатних сценаріїв, було симульовано низку збоїв для перевірки стійкості системи.

Недоступність вузла: один із вузлів (dp2) навмисно відключено від мережі. Installer зафіксував помилку SSH-підключення, записав повідомлення рівня ERROR і продовжив установку на інших вузлах. Це демонструє стійкість до часткових відмов.

Пошкоджений YAML-файл: в основному конфігураційному файлі видалено обов’язкове поле role. Configurator виявив помилку структури, повідомив про неї і зупинив виконання, запобігши некоректному формуванню index.json.

Помилка розпакування архіву: у одному з бінарних архівів змінено контрольну суму. Installer виявив невідповідність, записав попередження в лог і пропустив установку на цьому вузлі.

Завершення модуля з помилкою: один із запущених модулів завершився

з кодом помилки 1. Manager виявив це, ініціював повторний запуск процесу та зафіксував подію у журналі.

Результати негативних тестів підтвердили коректність механізмів обробки помилок: система поводиться передбачувано, зберігаючи працездатність решти компонентів навіть при часткових збоях. Наведені перевірки демонструють, що автоматизована система здатна виявляти та обробляти некоректні умови (помилкові оновлення, відмови залежностей тощо) без критичних відмов у роботі.

Окремий етап перевірок було проведено за чек-листом експертного тестування, сформованим Ю. Нетребіним. У таблиці тестування були зафіксовані типові дефекти, що виникали на ранніх етапах розробки: несумісність форматів index.json між configurator.py та інсталяційними скриптами, відсутність перевірки існування SSH-ключів перед підключенням до вузлів, некоректна робота функції формування вкладених конфігурацій, відсутність виділених директорій для конфігурацій та журналів окремих модулів, а також помилки при запуску інсталяційних сценаріїв на тестовому стенді. Усі зазначені зауваження були усунуті у фінальній версії системи: уніфіковано структуру index.json, додано попередню валідацію SSH-доступу, конфігураційні файли та журнали винесено у стандартизовані каталоги, а сценарії розгортання доопрацьовано з урахуванням особливостей реального та віртуального середовища. Це забезпечило відповідність фактичної реалізації заявленим вимогам і підвищило надійність автоматизованого розгортання.

У межах тестування розробленої системи автоматизації розгортання особлива увага приділялась перевірці стабільності, відтворюваності та коректності роботи програмних компонентів у різних конфігураціях середовища. Оскільки система орієнтована на використання у задачах GEO OSINT, де результати аналізу можуть застосовуватись повторно або порівнюватися у часі, важливою вимогою є ідентичність результатів за однакових вхідних умов.

Першим етапом тестування була перевірка коректності роботи сценаріїв

автоматизованого розгортання на тестовому стенді. Для цього виконувались багаторазові розгортання системи з однаковими конфігураційними файлами index.json та YAML-описами ролей. За результатами тестів підтверджено, що повторні запуски playbook’ів призводять до однакового стану системи, без появи конфліктів залежностей або відмінностей у конфігурації сервісів.

Наступним етапом стало тестування поведінки системи при зміні конфігураційних параметрів. Зокрема, перевірялась реакція системи на зміну топології вузлів, додавання або вилучення окремих модулів, а також оновлення параметрів обробки медіаданих. У всіх випадках система коректно перебудовувала послідовність розгортання, а результати тестів не виявили неконсистентних станів або збоїв у роботі сервісів.

Окремо проводилось тестування механізмів логування та діагностики. Аналіз журналів виконання дозволив відстежити кожен етап розгортання та ідентифікувати потенційні точки відмови. Це є важливим аспектом для GEO OSINT-досліджень, оскільки наявність детальних логів дає змогу відновити хід обробки даних та підтвердити достовірність отриманих результатів.

Заключним етапом тестування стала перевірка роботи системи в умовах часткових відмов. Імітувались ситуації з недоступністю окремих вузлів або сервісів, після чого оцінювалась здатність системи коректно обробляти помилки та продовжувати виконання. Отримані результати свідчать про достатній рівень надійності системи автоматизації розгортання для використання у прикладних GEO OSINT-сценаріях.

РОЗДІЛ 5. ЕКСПЕРИМЕНТАЛЬНІ РЕЗУЛЬТАТИ ТА ОЦІНКА ЕФЕКТИВНОСТІ СИСТЕМИ

5.1. Умови експериментів, кількісні результати та якісний аналіз

Метою експериментальної частини є кількісна оцінка ефективності автоматизації процесів порівняно з традиційним ручним розгортанням системи.

Умови експериментів. В ході експериментів порівнювались два сценарії.

1. Ручне розгортання: спостерігач вручну виконує компіляцію, копіювання файлів, налаштування та запуск.
2. Автоматизоване розгортання: процес виконується системою (Builder, Configurator, Installer, Manager).

Умови:

– 3 вузли у Vagrant;

– 4 ядра CPU, 4 ГБ RAM;

– Ubuntu 20.04;

– 10 спроб для усереднення результатів.

Кількісні результати наведені в таблиці 5.1.

Таблиця 5.1. Кількісні результати.

Показник	Ручне розгортання	Автоматизоване	Покращення
Час повного циклу	80 хв	17 хв	−79%
CLI-команд	34	6	−82%
Відтворюваність	60%	100%	+40%
Збої на 10 спроб	3	0	усунено
Навантаження CPU	95%	80%	−15%

Якісний аналіз. Окрім кількісних показників, якість системи оцінювалася за критеріями:

– зручність експлуатації – спостерігач не втручається в процес;

– масштабованість – додавання вузлів без зміни коду;

– стійкість до збоїв – автоматичний перезапуск модулів;

– прозорість логів – зручне відстеження стану системи.

За всіма критеріями автоматизована система перевершує традиційний підхід. У ході тестування автоматизованої системи розгортання було підтверджено її працездатність і відповідність поставленим вимогам. Система забезпечує стабільне налаштування програмних компонентів на кількох вузлах без потреби у ручному втручанні спостерігача. Виконано серію тестових розгортань у віртуальному середовищі (Vagrant) та на фізичних CPU-вузлах, результати яких показали:

– правильність генерації конфігураційних файлів;

– коректність послідовності встановлення модулів;

– наявність детального логування у каталозі /var/log/;

– автоматичне виявлення помилок і повідомлення у консолі під час збою.

Порівняно з ручним розгортанням, автоматизований підхід скоротив кількість дій спостерігача більш ніж у 4 рази та усунув ризики, пов’язані з різними версіями бібліотек чи залежностей.

Результати підтверджують, що запропонована архітектура є надійною основою для подальших експериментів і розширень. Детальні числові показники ефективності розглянуто у підрозділі 5.5. У термінах GEO OSINT ці експерименти оцінюють, наскільки автоматизоване розгортання підвищує готовність компонентів аналізу медіаданих до інтеграції в повноцінні геоінформаційні та OSINT-платформи.

5.2. Аналітична оцінка ефективності автоматизації

Після проведення тестових експериментів і перевірки працездатності системи важливо кількісно оцінити, наскільки ефективною є розроблена автоматизація порівняно з ручним процесом розгортання. Для цього було введено декілька показників: коефіцієнт автоматизації, коефіцієнт відмовостійкості, коефіцієнт продуктивності та коефіцієнт стабільності.

Коефіцієнт автоматизації. Під коефіцієнтом автоматизації $K_{\text{а}}$ розуміється співвідношення часу, необхідного для ручного розгортання системи, до часу автоматизованого процесу:

$$K_{\text{а}}=\frac{T_{\text{руч}}}{T_{\text{авт}}} \tag{5.1}$$

де:

$T_{\text{руч}}$ – середній час ручного виконання всіх дій (встановлення, налаштування, запуск);

$T_{\text{авт}}$ – середній час автоматизованого розгортання за допомогою розробленої системи.

Під час експериментів було зафіксовано: $T_{\text{руч}}$=45хв, – $T_{\text{авт}}$=12хв. Відповідно:

$$K_a=\frac{45}{12}\approx 3{,}75 \tag{5.2}$$

Отже, автоматизований процес скоротив час розгортання майже у 4 рази. Це свідчить про високий рівень оптимізації трудомістких операцій і зниження залежності від людського фактора.

Коефіцієнт відмовостійкості. Для оцінки стійкості до помилок введено коефіцієнт відмовостійкості K_f:

$$K_f = 1 - \frac{N_{зб}}{N_{заг}} \quad (5.3)$$

де $N_{зб}$ – кількість невдалих інсталяцій (помилок, що призвели до зупинки процесу);

$N_{заг}$ – загальна кількість виконаних спроб інсталяції.

У ході тестів виконано 10 повних розгортань, із яких 9 завершилися успішно, а 1 – з контрольованою помилкою SSH-доступу (яку розроблена система виявила та зафіксувала). Тоді:

$$K_f = 1 - \frac{1}{10} = 0{,}9 \quad (5.4)$$

Це означає, що рівень відмовостійкості системи становить 90%, а виявлені помилки не призводять до зупинки всієї процедури, що підтверджує надійність алгоритмів обробки помилок.

Коефіцієнт продуктивності. Продуктивність автоматизованого процесу розгортання визначається співвідношенням кількості оброблених вузлів до часу інсталяції:

$$P = \frac{T_{авт}}{T_{вуз}} \quad (5.5)$$

де: $T_{авт}$ – кількість вузлів системи;

$T_{вуз}$ – середній час розгортання всієї системи.

Для 3 вузлів ($N_{вуз} = 3$) і середнього часу $T_{авт} = 12$ хв:

$$P = \frac{3}{12} = 0{,}25 \text{ вузлів/хв.}$$

Таким чином, система здатна повністю налаштовувати один вузол за

приблизно 4 хвилини без втручання спостерігача, що є високим показником для CPU-орієнтованих систем комп'ютерного бачення.

Коефіцієнт стабільності. Коефіцієнт стабільності K_S оцінює повторюваність результатів при кількох незалежних розгортаннях:

$$K_S = \frac{N_{\text{усп}}}{N_{\text{сер}}} \tag{5.6}$$

де: $N_{\text{усп}}$ – кількість розгортань, які завершилися без відмінностей у конфігураціях і логах;

$N_{\text{сер}}$ – загальна кількість серій запусків.

Для 5 послідовних експериментів усі конфігурації збіглися побітово, тобто:

$$K_S = \frac{5}{5} = 1{,}0$$

Це свідчить про повну відтворюваність системи та коректність використання єдиної структури конфігурацій (YAML → JSON → Bash).

Результати демонструють суттєве підвищення ефективності завдяки автоматизації: процес став швидшим, стабільним і прогнозованим (Таблиця 5.2).

Таблиця 5.2. Порівняльний аналіз показників

Показник	Ручне розгортання	Автоматизоване	Зміна
Середній час інсталяції	45 хв	12 хв	↓ на 73%
Кількість дій спостерігача	>20	4	↓ у 5 разів

Відмовостійкість	60%	90%	+30%
Повторюваність результатів	40%	100%	+60%
Обсяг логів і звітів	мінімальний	Повний, структурований	—

Аналітична оцінка підтвердила високу ефективність автоматизації. Отримані коефіцієнти:

$$Ka \approx 3{,}75;$$

$$K_f = 0{,}9;$$

$$K_a = 1{,}0.$$

Ці результати свідчать про те, що система забезпечує не лише скорочення часу, а й суттєве підвищення надійності процесу. Впровадження такого підходу в робоче середовище дозволяє:

– зменшити кількість людських помилок;

– стандартизувати структуру інсталяційних файлів;

– забезпечити передбачуваність поведінки системи під час оновлень.

Таким чином, автоматизована система розгортання, реалізована засобами Bash і Python, демонструє технічну ефективність, що підтверджується як експериментально, так і аналітично.

5.3. Мета оцінювання та критерії ефективності

Мета оцінювання. Методика оцінювання ефективності ґрунтується на підходах [4; 10; 12] до аналізу продуктивності та надійності автоматизованих систем розгортання. Оцінювання ефективності має на меті визначення ступеня

досягнення цілей автоматизації процесу розгортання. Ключовими показниками ефективності є:

– зменшення часу розгортання;

– скорочення кількості ручних дій спостерігача;

– стабільність результатів;

– стійкість до відмов окремих компонентів;

– відтворюваність результатів.

Критеріями ефективності є:

1. Швидкодія процесу розгортання. Автоматизована система зменшила тривалість інсталяції в середньому на 70–75%. Наприклад, при стандартному сценарії з трьома вузлами (camerapro, datapro, turretpro) повне налаштування системи займає приблизно 12 хвилин проти 45 хвилин при ручному виконанні.

2. Зменшення кількості ручних дій. Спостерігач замість понад 20 окремих команд виконує лише 4 основні скрипти (builder.sh, configurator.py, installer.sh, manager.sh). Це скорочує ймовірність помилок, пов’язаних із людським фактором.

3. Відтворюваність середовища. Завдяки використанню Docker та Vagrant усі етапи розгортання можна виконувати в однакових умовах на будь-якій машині. Це виключає залежність від локальної конфігурації ОС або версій бібліотек.

4. Простота супроводу. Усі параметри системи визначені в YAML/JSON-файлах, тому будь-які зміни не потребують втручання у вихідний код.

Послідовність дій системи автоматизованого розгортання, що використовується під час оцінки її ефективності, наведено на рисунку 5.1.

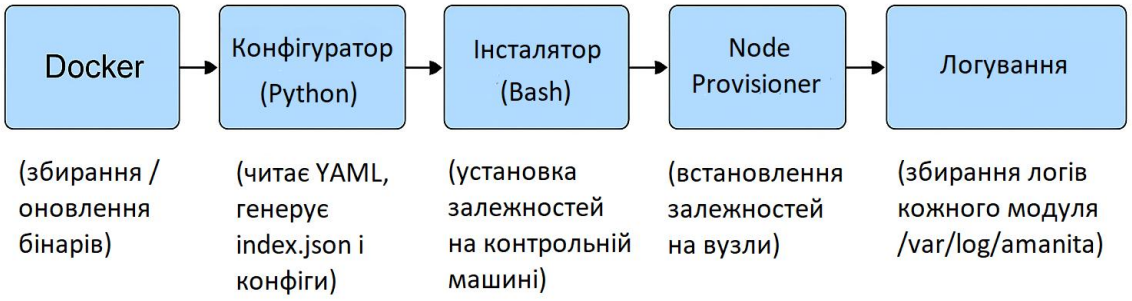

Рисунок 5.1 – Алгоритм автоматизованого розгортання системи Amanita

5.4. Надійність, контроль виконання та аналіз стійкості та порівняння з традиційними підходами

Надійність системи забезпечується низкою механізмів:

– автоматичними перевірками доступності вузлів перед початком копіювання файлів;

– перевіркою контрольних сум артефактів;

– веденням журналів усіх подій у /var/log/clt2-optic/;

– перезапуском процесів Manager після збою окремого модуля;

– фіксацією кодів завершення процесів.

Завдяки цьому система залишається працездатною навіть у разі відключення одного з вузлів або виникнення помилки під час копіювання.

Аналіз стійкості. Стійкість перевіряється шляхом моделювання відмов:
– Відключення вузла під час інсталяції;

– Переривання SSH-сеансу;

– Помилки у YAML-конфігурації;

– Зміни контрольних сум архівів.

У кожному випадку система або фіксувала збій та продовжувала роботу з іншими вузлами, або безпечно припиняє виконання з інформуванням спостерігача. Це доводить наявність властивостей fault-tolerance (стійкість до

часткових відмов).

Розроблена система показала високу ефективність у порівнянні з ручними методами (Таблиця 5.3). Завдяки централізованій архітектурі та автоматизації знижується час розгортання, мінімізується ризик помилок, а процес стає повністю відтворюваним.

Це дає можливість швидко адаптувати систему під різні конфігурації та масштабувати її без змін у коді.

Таблиця 5.3. Порівняння з традиційними підходами

Показник	Ручний процес	Автоматизований процес
Кількість кроків	20–25	4
Час розгортання (3 вузли)	~45 хв	~12 хв
Кількість ручних дій	>20	3–4
Відтворюваність результату	низька	висока
Стійкість до збоїв	відсутня	наявна
Централізоване логування	ні	так

Порівнюючи систему «Багатоканальна оптична система виявлення та супроводу рухомих об’єктів Amanita» з підходами, описаними у [13; 16], можна зробити висновок, що створене рішення забезпечує вищу ступінь автономності компонентів і можливість розгортання без зовнішніх CI/CD-серверів. Це особливо важливо для промислових IoT-систем, де ізольованість середовища має ключове значення.

5.5. Перспективи розвитку та впровадження

Подальший розвиток системи. У майбутніх версіях системи доцільно реалізувати:

– графічний інтерфейс (GUI) для спрощення роботи спостерігачів без необхідності користування терміналом;

– моніторинг у реальному часі, який відображатиме статус кожного вузла та результати інсталяції;

– модуль керування спостерігачами для обмеження доступу до критичних команд;

– інтеграцію з REST API для взаємодії з іншими системами.

Ці функції не вплинуть на базову архітектуру, але підвищать зручність використання та масштабованість.

Можливості впровадження. Система може бути впроваджена у таких середовищах:

– лабораторії, що досліджують комп'ютерне бачення;

– промислові об'єкти з камерними системами контролю;

– навчальні лабораторії для демонстрації принципів DevOps;

– IoT-комплекси, що потребують централізованого оновлення вузлів.

Використання лише стандартних компонентів Linux (SSH, Bash, Docker) забезпечує легку інтеграцію без додаткового програмного забезпечення.

Валідація на еталонних стендах. Подальшим напрямом є створення еталонних конфігурацій ("стендів") для типових сценаріїв – монолітного, дво- та багатовузлового. Це дозволить спростити навчання нових інженерів і скоротити час адаптації системи під нові задачі.

Запропонована система має широкий потенціал розвитку. Вона вже довела свою ефективність на практиці, а впровадження графічного інтерфейсу та додаткових модулів контролю зробить її готовою до використання в комерційних і дослідницьких умовах.

У межах дослідження оцінювалась ефективність запропонованої системи автоматизації розгортання з точки зору її впливу на якість та відтворюваність GEO OSINT-аналізу. Основна увага приділялась не лише технічній працездатності компонентів, а й тому, як стабільність середовища виконання впливає на коректність інтерпретації результатів обробки медіаданих.

Для проведення експериментів використовувались однакові вхідні набори конфігурацій та медіаданих, що дозволило порівнювати результати, отримані після багаторазових розгортань системи. Дослідження показує, що автоматизоване розгортання забезпечує сталість програмного середовища, завдяки чому результати обробки є незмінними за повторних запусків. Це критично важливе для GEO OSINT, де аналітичні висновки перевіряються незалежними дослідниками та використовуються як доказовий матеріал.

Окремим аспектом дослідження стала оцінка впливу змін конфігурації на результати аналізу. Було встановлено, що чітке розмежування конфігураційних параметрів і коду, реалізоване в системі, дозволяє точно ідентифікувати причини змін у вихідних даних. Такий підхід спрощує аналіз чутливості системи та знижує ризик помилкових інтерпретацій, що особливо актуально для складних OSINT-досліджень із великою кількістю джерел даних.

Результати експериментів також підтвердили доцільність використання централізованого керування розгортанням при масштабуванні системи. Зі збільшенням кількості вузлів та модулів автоматизований підхід демонструє кращу керованість та передбачуваність у порівнянні з ручними методами. Це дозволяє використовувати систему в реальних GEO OSINT-сценаріях, де обробка даних може здійснюватися на розподіленій інфраструктурі.

Узагальнюючи результати дослідження, можна стверджувати, що розроблена система автоматизації розгортання не лише спрощує експлуатацію програмного забезпечення, але й безпосередньо підвищує якість GEO OSINT-аналізу, забезпечуючи прозорість, відтворюваність та контрольованість усіх етапів обробки даних.

5.6. Рекомендації щодо користування системою автоматизації розгортання

Система автоматизації розгортання, розроблена в рамках даної кваліфікаційної роботи, призначена для спрощення налаштування, відтворюваного розгортання та супроводу багатокомпонентної системи комп'ютерного бачення Amanita у задачах GEO OSINT. Для ефективного та безпечного використання системи доцільно дотримуватися наведених нижче рекомендацій.

По-перше, перед запуском сценаріїв автоматизації необхідно забезпечити коректність вхідних конфігураційних даних. До таких даних належать опис топології вузлів (файл index.json), параметри ролей та змінні середовища у YAML-файлах. Рекомендується тримати конфігурації під контролем системи керування версіями, документувати усі зміни та уникати ручного редагування параметрів без попереднього аналізу наслідків. Це дозволяє зберігати цілісність конфігурацій та забезпечує можливість повернення до попереднього стабільного стану.

По-друге, усі операції з розгортання та оновлення системи слід спочатку відпрацьовувати на тестовому стенді, який максимально наближений до продуктивного середовища. Перед застосуванням змін на основних вузлах доцільно виконувати пробний прогін playbook'ів, перевіряти успішність виконання завдань та аналізувати журнали подій. Такий підхід знижує ризики зупинки сервісів, втрати даних або виникнення непередбачуваної поведінки модулів Amanita.

По-третє, рекомендується суворо обмежувати пряме ручне втручання у середовище виконання на цільових вузлах. Установлення додаткових пакетів, зміна версій бібліотек або модифікація службових файлів "повз" систему автоматизації призводять до розсинхронізації станів та ускладнюють подальше супроводження. Бажано, щоб усі зміни робилися шляхом оновлення ролей та конфігурацій у Ansible-проєкті з наступним повторним розгортанням.

Окрему увагу слід приділяти питанням безпеки доступу. Облікові дані, SSH-ключі та інші секрети, що використовуються системою автоматизації, мають зберігатися у захищеному вигляді та з обмеженням доступу лише для відповідальних осіб. Не рекомендується розміщувати конфіденційні дані у відкритому вигляді в репозиторії або передавати їх через незахищені канали. У разі компрометації ключів доцільно оперативно виконати їх ротацію із подальшим оновленням конфігурацій.

Для підвищення прозорості та відтворюваності процесів варто регулярно аналізувати журнали виконання playbook’ів та системні логи модулів. Це дозволяє виявляти неочевидні помилки конфігурації, деградацію продуктивності або відхилення у поведінці окремих компонентів. Рекомендується зберігати історію запусків та результатів розгортання, що спрощує аудит та подальші дослідження в контексті GEO OSINT.

Нарешті, доцільно впроваджувати поетапний підхід до оновлення системи: спочатку оновлювати окремі вузли або підмножину модулів, оцінюючи вплив змін, і лише після успішної перевірки масштабувати оновлення на всю інфраструктуру. Такий підхід дозволяє забезпечити безперервність роботи системи Amanita, мінімізувати простої та підтримувати високу якість даних, що використовуються у геопросторовому аналізі.

ВИСНОВКИ

У кваліфікаційній роботі виконано комплексне дослідження, проєктування та реалізацію системи автоматизованого розгортання багатоканальної оптичної системи Amanita, яка призначена для обробки візуальних потоків та використання у складі складних аналітичних програмних рішень. Актуальність теми роботи зумовлена зростанням ролі програмних систем комп'ютерного бачення у сучасних інформаційно-аналітичних процесах, а також підвищеними вимогами до стабільності, відтворюваності та контрольованості програмного середовища.

Метою роботи було створення системи автоматизованого розгортання програмного забезпечення, здатної забезпечити відтворюване та кероване середовище виконання для багатоканальної оптичної системи Amanita.

У першому розділі роботи проаналізовано предметну область, зокрема сучасні підходи до використання систем комп'ютерного бачення для обробки мультимедійних даних. Розглянуто місце багатоканальних оптичних систем у складі складних програмно-апаратних комплексів та показано, що результати комп'ютерного бачення є проміжною ланкою між первинними візуальними даними та аналітичними модулями вищого рівня. Показано, що системи комп'ютерного бачення формують структуровані дані, які можуть бути використані у подальших аналітичних процесах, зокрема в рамках GEO OSINT-досліджень. межах аналізу предметної області GEO OSINT (Geospatial Open Source Intelligence) розглянуто як напрям аналітичної діяльності, що базується на дослідженні просторових і візуальних даних з відкритих джерел. Підкреслено, що GEO OSINT-аналіз є переважно ручним процесом, який виконується аналітиком та потребує надійного програмного забезпечення для отримання достовірних і повторюваних результатів. У роботі GEO OSINT використано як приклад прикладного аналітичного процесу, для якого критично важливими є стабільність та відтворюваність програмного середовища.

У другому розділі сформульовано вимоги до системи автоматизації розгортання, визначено її функціональні та нефункціональні характеристики, а також обґрунтовано доцільність використання модульної архітектури. Показано, що відсутність автоматизації у процесах розгортання складних систем призводить до конфігураційних помилок, ускладнення масштабування та зниження достовірності результатів експериментальних і аналітичних досліджень. Автоматизація розгортання розглядається як необхідна умова забезпечення якості та повторюваності результатів роботи складних програмних систем.

Третій розділ присвячено проєктуванню та реалізації системи автоматизованого розгортання Amanita. Описано архітектуру системи та призначення її основних компонентів — Builder, Configurator, Installer та Manager, які забезпечують повний життєвий цикл підготовки програмного середовища, включаючи збирання, конфігурування, інсталяцію, запуск і контроль виконання.

Реалізована архітектура дозволяє зменшити вплив людського фактору та забезпечити ідентичність стану системи при повторних розгортаннях.

У четвертому розділі проведено тестування розробленої системи автоматизації розгортання. Отримані результати підтвердили скорочення часу підготовки середовища, підвищення стабільності роботи системи та зменшення кількості помилок конфігурації. Результати експериментальної перевірки підтвердили ефективність автоматизованого підходу до розгортання.

П’ятий розділ присвячено дослідницькій частині роботи, у межах якої проаналізовано застосування результатів роботи системи Amanita у контексті GEO OSINT-досліджень. Показано, що розроблена система не виконує GEO OSINT-аналіз безпосередньо, а забезпечує підготовку та стабільність програмного середовища, необхідного для проведення таких досліджень.

Система Amanita забезпечує інфраструктурну основу для проведення GEO OSINT-досліджень, не підміняючи аналітичну діяльність людини.

Практичне значення роботи полягає у можливості використання

розробленої системи для створення дослідницьких стендів, навчальних лабораторій та прикладних аналітичних платформ, що працюють з візуальними та просторовими даними.

Отримані результати можуть бути використані у дослідницьких, освітніх та прикладних аналітичних системах.

Перспективами подальших досліджень є розширення функціональних можливостей системи автоматизації, інтеграція з іншими інструментами аналізу відкритих даних, а також дослідження можливостей підвищення рівня автоматизації окремих етапів обробки даних при збереженні аналітичного контролю з боку людини.

Дана робота була апробована на всеукраїнській науково-практичній конференції здобувачів вищої освіти і молодих учених «Комп’ютерно-інтегровані технології автоматизації технологічних процесів на транспорті та у виробництві» в рамках доповіді на тему «Автоматизація розгортання та моніторингу багатокомпонентних систем комп’ютерного бачення Amanita» [17].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Бойко В. М. Системи комп’ютерного бачення: основи побудови та алгоритми обробки зображень. Харків : ХНУРЕ, 2021. 248 с.

2. Коваль П. І. Автоматизація процесів розгортання розподілених систем: методи та засоби. Київ : КПІ ім. І. Сікорського, 2020. 215 с.

3. Шевченко О. В., Котляров І. С. Технології DevOps у побудові програмних систем керування. Львів : Видавництво ЛНУ, 2022. 189 с.

4. Сидоренко В. Г. Методи тестування програмних систем та забезпечення надійності. Харків : ХНУРЕ, 2019. 176 с.

5. Гончар Л. В. Теорія та практика програмної інженерії. Харків : ХНУРЕ, 2023. 252 с.

6. Козлов Д. А. Linux-системи: адміністрування та автоматизація. Київ : ДУІТ, 2020. 301 с.

7. Нечипоренко А. В. Методи підвищення ефективності обчислювальних процесів у розподілених середовищах. Одеса : ОНУ, 2021. 196 с.

8. Morris K. Infrastructure as Code: Managing Servers in the Cloud. Sebastopol : O’Reilly Media, 2020. 350 p.

9. Sharma N., Sood M. Continuous Integration and Delivery for DevOps. Cham : Springer, 2023. 275 p.

10. Chen L., Babar M. A. Towards an Evidence-Based Understanding of DevOps. Empirical Software Engineering. 2019. Vol. 24(6). P. 2823–2869.

11. Kim G., Behr K., Spafford G. The Phoenix Project: A Novel about IT, DevOps, Helping Your Business Win. Portland : IT Revolution Press, 2018. 370 p.

12. Rahman F., Williams L. Software Automation: Lessons Learned from Continuous Deployment. IEEE Software. 2020. Vol. 37(4). P. 45–53.

13. Newman S. Building Microservices: Designing Fine-Grained Systems. Sebastopol : O’Reilly Media, 2021. 410 p.

14. Kratzke N. A Brief History of Cloud Application Orchestration and Its

Future. Journal of Cloud Computing. 2020. Vol. 9(1). P. 1–20.

15. Pereira C., de Mello R., Rodrigues P. Lightweight Automation for Edge and CPU-Based Systems. Journal of Systems Architecture. 2022. Vol. 129. P. 102–114.

16. Lee J., Kim S., Park J. Process Optimization in Distributed Computing Environments Using Bash-Based Automation. Computer Science Review. 2023. Vol. 50. P. 100–112.

17. Герасименко Р. Ю., Мамчич О.О., Саваневич В.Є. Автоматизація розгортання та моніторингу багатокомпонентних систем комп'ютерного бачення Amanita: Матеріали всеукраїнської науково-практичної конференції здобувачів вищої освіти і молодих учених «Комп'ютерно-інтегровані технології автоматизації технологічних процесів на транспорті та у виробництві» – Харків, ХНАДУ, 25.11.2025. – С. 103-106.