Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота бакалавра

на тему Автоматизація розгортання безсерверних застосунків із використанням IaC

Виконав: студент групи К-21-3
Спеціальність 122 «Комп'ютерні науки»
Булгаков Д. С.
(прізвище та ініціали)

Керівник
к.т.н., доцент Мормуль М. Ф.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рецензент Університет митної справи та фінансів
(місце роботи)

(посада)

(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2025

АНОТАЦІЯ

Булгаков Д.С. Автоматизація розгортання безсерверних застосунків із використанням IaC.

Дипломна робота на здобуття освітнього ступеня бакалавр за спеціальністю 122 “Комп’ютерні науки”. – Університет митної справи та фінансів, Дніпро, 2025.

Кваліфікаційна робота присвячена дослідженню та практичній реалізації процесу автоматизації розгортання безсерверних застосунків із використанням підходу «Інфраструктура як код» (IaC).

Об’єктом дослідження є процес управління інфраструктурою сучасних програмних систем у хмарних середовищах.

Предмет дослідження – розробка методики автоматизованого розгортання serverless-застосунків із використанням сучасних IaC-інструментів.

Метою роботи є створення універсального підходу до автоматизації розгортання безсерверних застосунків, який забезпечує гнучкість, масштабованість та стабільність інформаційних систем за мінімальних витрат часу на підтримку інфраструктури. У роботі проведено аналіз теоретичних основ безсерверної архітектури, розглянуто ключові інструменти IaC (Terraform, Pulumi, AWS CloudFormation, Ansible) та обґрунтовано вибір оптимальних технологій для впровадження у сучасних хмарних середовищах. Особливу увагу приділено платформі Pulumi, що дозволяє використовувати знайомі мови програмування для опису інфраструктури, спрощуючи автоматизацію процесів CI/CD.

Практична частина роботи реалізована на прикладі створення системи «Agro Monitor», яка використовує serverless-архітектуру та автоматизоване розгортання через Pulumi в Azure. Детально розглянуто побудову багатошарової архітектури застосунку, взаємодію компонентів через асинхронні API, організацію сховищ, роботу з супутниковими даними, а також питання безпеки та оптимізації витрат. Запропоноване рішення дозволяє розробникам швидко і

безпомилково розгортати нові середовища, масштабувати ресурси під поточне навантаження й ефективно управляти життєвим циклом застосунку.

Практичне значення роботи полягає у формуванні підходів до автоматизації розгортання інфраструктури, що можуть бути застосовані у різних галузях для прискорення впровадження хмарних рішень, підвищення їх надійності та зменшення експлуатаційних витрат.

Ключові слова: безсерверна архітектура, інфраструктура як код, автоматизація розгортання, IaC, Pulumi, Azure, Terraform, DevOps, Agro Monitor, serverless, хмарні технології.

ABSTRACT

Bulgakov D.S. Automation of Serverless Application Deployment Using IaC.

Bachelor's Thesis in partial fulfillment of the requirements for the degree of Bachelor in specialty 122 "Computer Science". – University of Customs and Finance, Dnipro, 2025.

This qualification thesis is devoted to the research and practical implementation of automating the deployment of serverless applications using the Infrastructure as Code (IaC) approach.

The object of the study is the process of managing the infrastructure of modern software systems in cloud environments.

The subject of the study is the development of a methodology for automated deployment of serverless applications using modern IaC tools.

The main goal of this work is to develop a universal approach to automating the deployment of serverless applications, ensuring flexibility, scalability, and stability of information systems with minimal maintenance overhead. The thesis analyzes the theoretical foundations of serverless architecture, reviews key IaC tools (Terraform, Pulumi, AWS CloudFormation, Ansible), and justifies the selection of optimal technologies for deployment in modern cloud environments. Special attention is paid to the Pulumi platform, which allows using familiar programming languages to describe infrastructure, thus simplifying the automation of CI/CD processes.

The practical part of the thesis is implemented through the development of the "Agro Monitor" system, which leverages serverless architecture and automated deployment via Pulumi on Azure. The work provides a detailed overview of the application's multilayer architecture, component interaction via asynchronous APIs, data storage organization, satellite data processing, as well as security and cost optimization issues. The proposed solution enables developers to quickly and reliably deploy new environments, scale resources based on current workloads, and efficiently manage the application lifecycle.

The practical value of this thesis lies in the formulation of approaches for infrastructure deployment automation, which can be applied across various domains to accelerate the adoption of cloud solutions, enhance their reliability, and reduce operational costs.

Keywords: serverless architecture, infrastructure as code, deployment automation, IaC, Pulumi, Azure, Terraform, DevOps, Agro Monitor, serverless, cloud technologies.

ЗМІСТ

ВСТУП

Актуальність дослідження. Сучасний етап розвитку інформаційних технологій характеризується високими темпами впровадження інноваційних підходів до організації та супроводу програмного забезпечення. Відкритість глобального ринку, зростання конкуренції між IT-компаніями, а також постійне підвищення вимог до швидкості розробки й стабільності IT-продуктів зумовлюють необхідність удосконалення процесів розгортання, масштабування та обслуговування інформаційних систем.

Одним із ключових трендів у цій галузі є поширення хмарних технологій, які дозволяють компаніям швидко адаптувати інфраструктуру під змінні бізнес-вимоги, оптимізувати витрати та підвищити надійність сервісів. Особливої популярності останнім часом набули безсерверні архітектури (serverless architecture), які суттєво змінюють підходи до побудови та експлуатації сучасних застосунків.

Переваги безсерверної моделі – зменшення витрат на підтримку інфраструктури, автоматичне масштабування, оплата лише за використані ресурси – дають змогу організаціям фокусуватися на розробці бізнес-логіки, залишаючи питання адміністрування хмарним провайдерам. Проте, впровадження serverless-архітектур супроводжується низкою нових викликів: зростає складність взаємодії між окремими компонентами, ускладнюється контроль версій і життєвого циклу ресурсів, виникає потреба у швидкому, відтворюваному та безпечному розгортанні великої кількості елементів.

У цих умовах інфраструктура як код (Infrastructure as Code, IaC) стає незамінним інструментом для автоматизації розгортання і підтримки інформаційних систем. IaC дозволяє визначати конфігурацію інфраструктури за допомогою зрозумілих текстових описів, що забезпечує стандартизацію, прозорість, гнучкість і відтворюваність процесів. Завдяки широкому спектру сучасних інструментів IaC (Terraform, Pulumi, AWS CloudFormation, Ansible тощо) DevOps-команди можуть автоматизувати всі етапи життєвого циклу

застосунків, скорочуючи час на оновлення й мінімізуючи ризики людського фактора.

Інноваційність цієї дипломної роботи полягає у комплексному підході до дослідження й практичного впровадження автоматизації розгортання serverless-застосунків з використанням сучасних IaC-інструментів. Особливий акцент зроблено на платформі Pulumi, яка поєднує переваги класичних інструментів IaC з можливістю використання знайомих мов програмування для опису інфраструктури, що значно спрощує інтеграцію у CI/CD-процеси та підвищує ефективність роботи DevOps-команд.

Результати дослідження мають значну практичну цінність, оскільки дозволяють впроваджувати гнучкі, масштабовані та надійні рішення для розгортання IT-інфраструктури як у малих стартапах, так і у великих підприємствах.

Метою роботи є розробка універсального підходу до автоматизованого розгортання serverless-застосунків у хмарному середовищі з урахуванням вимог сучасної інженерії програмного забезпечення.

Для досягнення цієї мети у дипломній роботі вирішуються такі основні завдання:

- дослідити теоретичні засади безсерверної архітектури та концепції IaC;
- проаналізувати сучасні інструменти та сервіси для реалізації IaC у хмарних платформах;
- розробити архітектуру serverless-застосунку та побудувати процес його автоматизованого розгортання на базі реального проєкту (“Agro Monitor”);
- впровадити практичне рішення із використанням Pulumi для Azure та оцінити переваги автоматизації на всіх етапах життєвого циклу системи.

Об’єкт дослідження – процес управління інфраструктурою та розгортання serverless-застосунків у хмарних середовищах.

Предмет дослідження – методика автоматизації розгортання serverless-застосунків із використанням IaC-інструментів.

Методи дослідження. У роботі використано комплексний підхід, що включає аналіз наукових джерел, системне моделювання архітектур, порівняльний аналіз інструментів і технологій IaC, проєктування програмної архітектури, розробку та експериментальне впровадження автоматизованих процесів розгортання. У межах дослідження увага приділяється розробці інтегрованого рішення, яке поєднує зручність використання, масштабованість і дотримання сучасних DevOps практик.

Практичне значення роботи полягає у формуванні стандартних підходів до автоматизації розгортання інформаційних систем, що можуть бути застосовані для оптимізації IT-процесів у різних галузях.

Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. Обсяг роботи становить 50 сторінок основного тексту, 6 рисунків та 2 таблиці.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ЗАСАДИ АВТОМАТИЗАЦІЇ РОЗГОРТАННЯ БЕЗСЕРВЕРНИХ ЗАСТОСУНКІВ

1.1 Безсерверна архітектура

Безсерверна архітектура, або serverless, є одним із сучасних підходів до розробки програмного забезпечення, який усуває необхідність управління фізичними серверами чи віртуальними машинами. У цій моделі обчислювальні ресурси автоматично керуються хмарними провайдерами, такими як Amazon Web Services (AWS Lambda), Microsoft Azure Functions або Google Cloud Functions [1], [2].

Термін "serverless" не означає відсутність серверів, а вказує на те, що розробникам не потрібно займатися адмініструванням серверної інфраструктури. Сервери, на яких виконується код, управляються провайдерами хмарних послуг, такими як AWS, Microsoft Azure або Google Cloud [1], [3].

Основна ідея полягає у виконанні коду в подієво-орієнтованому середовищі: функції запускаються у відповідь на події, наприклад, HTTP-запити, зміни у файлах чи базах даних [2].

Однією з головних переваг цієї архітектури є відсутність необхідності управління серверами. Команди розробників звільняються від завдань налаштування, масштабування та моніторингу інфраструктури, оскільки цим займається хмарний провайдер. Автоматичне масштабування дозволяє системі динамічно адаптуватися до змін у навантаженні, що критично важливо для сучасних веб-додатків і мобільних сервісів [3].

Ще одним вагомим аргументом на користь без серверної архітектури є модель оплати за використання: замовник сплачує лише за час, коли функція реально виконується, а не за постійно працюючі сервери. Це знижує витрати на обчислювальні ресурси, особливо для додатків зі змінним або нерегулярним навантаженням [4].

Однією з ключових технологій, що реалізує безсерверну модель, є FaaS (Functions as a Service). Це підхід, у якому код організований у вигляді функцій, що виконуються окремо одна від одної. Кожна функція має чітко визначене завдання і запускається в контексті певної події [3], [5].

До основних платформ FaaS належать:

- AWS Lambda - пропонує виконання функцій у хмарі з інтеграцією з іншими сервісами AWS, такими як DynamoDB, S3, API Gateway тощо [6];.
- Azure Functions - забезпечує гнучкість та підтримку кількох мов програмування, включаючи C#, JavaScript и Python [7];
- Google Cloud Functions - надає можливість створення функцій, інтегрованих із сервісами Google Cloud, такими як Pub/Sub і Firestore [8].

FaaS дозволяє створювати високопродуктивні, масштабовані та економічно вигідні застосунки.

Однак безсерверна модель має свої виклики. Зокрема, існує проблема "холодного старту" функцій: при довгому простої запуск функції може зайняти більше часу через необхідність ініціалізації інфраструктури. Також функції зазвичай мають обмеження на тривалість виконання, що може стати бар'єром для складних або тривалих завдань. Ще один важливий аспект – залежність від постачальника послуг (vendor lock-in), оскільки функції часто розробляються із використанням специфічних API певного хмарного провайдера.

1.2 Інфраструктура як код (IaC) - ключовий інструмент автоматизації

Інфраструктура як код (IaC) – це підхід, який дозволяє управляти IT-інфраструктурою за допомогою коду, описуючи її у вигляді текстових файлів або скриптів. Ця методологія дозволяє описати усю необхідну інфраструктуру, включаючи обчислювальні ресурси, мережі та бази даних, у вигляді декларативних та імперативних конфігураційних файлів.

IaC використовує два основних підходи для опису інфраструктури. При декларативному підході розробник описує кінцевий стан інфраструктури, який

система повинна досягти. Інструменти IaC автоматично визначають кроки, необхідні для переходу від поточного стану до заданого. Декларативний підхід забезпечує ідемпотентність, тобто повторне виконання одного й того ж коду не змінює вже налаштовану інфраструктуру (наприклад Terraform, AWS CloudFormation. Імперативний підхід передбачає покрокове описання того, як досягти бажаного стану інфраструктури. У цьому випадку розробник вказує послідовність дій, які потрібно виконати. Імперативний підхід забезпечує більший контроль, але складніший у підтримці (наприклад Ansible, Pulumi.

Інфраструктура як код (IaC) має низку важливих переваг, які роблять її потужним інструментом для автоматизації процесів управління IT-ресурсами. Головною перевагою є можливість автоматизувати розгортання інфраструктури, усуваючи потребу в ручному налаштуванні. Завдяки цьому ресурси можна створювати швидко і з мінімальними зусиллями.

IaC забезпечує ідемпотентність, що означає, що повторне виконання одного й того ж коду завжди призводить до однакового результату. Це гарантує стабільність середовища і зменшує ризик непередбачуваних змін. Крім того, автоматизація розгортання значно скорочує час на створення інфраструктури, дозволяючи командам швидше впроваджувати нові рішення.

Ще однією перевагою є масштабованість. IaC дає змогу легко адаптувати інфраструктуру як для невеликих тестових середовищ, так і для великих систем. Це особливо важливо для організацій, які працюють у динамічному середовищі і потребують швидкої адаптації до змін.

IaC тісно інтегрується з сучасними DevOps-практиками, зокрема процесами CI/CD, що забезпечує автоматичне оновлення середовищ і впровадження змін без перерв у роботі системи. Однією з ключових можливостей IaC є підтримка інтеграції з системами контролю версій, такими як Git. Це дозволяє зберігати всі зміни в конфігураціях інфраструктури у репозиторіях, що полегшує їх відстеження, перевірку й аналіз історії змін. Завдяки цьому кожна версія інфраструктури може бути відновлена, а процеси впровадження змін стають більш керованими та прозорими.

Усі зміни в інфраструктурі описуються у вигляді коду, що робить їх прозорими та легкими для відстеження, перевірки й документування. Це спрощує управління інфраструктурою і підвищує її надійність.

На сучасному етапі розвитку IaC займає центральне місце у практиці DevOps та автоматизації хмарних рішень. Суттєвим трендом є поява GitOps – моделі управління, при якій стан інфраструктури автоматично синхронізується із git-репозиторієм, а будь-які зміни проходять через pull request, рев'ю і автоматичне тестування.

IaC є ключовим фактором для досягнення "Immutable Infrastructure" – підходу, за якого інфраструктурні компоненти не змінюються вручну, а завжди створюються заново з перевіреного коду. Це підвищує надійність системи, спрощує аудит, прискорює disaster recovery та забезпечує відтворюваність середовищ.

У практиці сучасних команд IaC використовується не лише для створення хмарних серверів чи мереж, а й для автоматизації цілого стека: баз даних, балансувальників навантаження, API Gateway, Kubernetes-кластерів, безсерверних функцій, сховищ, ключових vault-сховищ тощо.

Особливу роль набувають інструменти для автоматичної перевірки безпеки IaC-коду (наприклад, Checkov, tfsec, kube-score), що дозволяє виявляти критичні налаштування та попереджати про вразливості ще до розгортання середовища. Це формує базу для впровадження підходу DevSecOps, де безпека інтегрована у всі етапи життєвого циклу.

Варто також враховувати ризики, пов'язані з "Infrastructure Drift" – розбіжністю між бажаним і фактичним станом інфраструктури. Для контролю цієї проблеми провідні IaC-платформи мають можливість автоматичного порівняння станів, аудит логу змін, а також функціонал policy-as-code (наприклад, Open Policy Agent).

Вибір між декларативними та імперативними IaC-підходами визначає рівень абстракції, контроль над ресурсами та складність підтримки. Наприклад, Terraform дозволяє описати інфраструктуру у вигляді "desired state", що

ідеально підходить для стандартних хмарних середовищ, а Pulumi, використовуючи знайомі мови програмування, дає можливість вбудовувати складну логіку, цикли, умови, імпорт бібліотек, що особливо актуально для складних динамічних проєктів.

IaC реалізується через спеціалізовані інструменти, які забезпечують створення, управління та оновлення інфраструктури:

- Terraform - один із найпопулярніших інструментів для мультихмарного управління інфраструктурою. Використовує декларативний підхід і дозволяє створювати крос-хмарні конфігурації. Terraform підтримує широкий спектр провайдерів, таких як AWS, Azure, Google Cloud, Kubernetes тощо [9];
- AWS CloudFormation призначений для управління ресурсами в екосистемі AWS. Використовує декларативні шаблони, які описують всі ресурси інфраструктури у форматі JSON або YAML [10];
- Pulumi забезпечує імперативний підхід і дозволяє використовувати мови програмування (JavaScript, Python, Go) для опису інфраструктури. Підтримує мультихмарні середовища [11];
- Ansible - інструмент для автоматизації інфраструктури, який використовує YAML для опису завдань. Найкраще підходить для конфігурації серверів і управління операційними системами [12].

Всі ці інструменти активно підтримують сучасні методики DevOps і сприяють стандартизації підходів до управління інфраструктурою у хмарних та гібридних середовищах.

IaC також є основою для впровадження DevOps практик, таких як CI/CD (Continuous Integration/Continuous Delivery - безперервна інтеграція та безперервне розгортання), що спрощує оновлення інфраструктури та прискорює вихід продукту на ринок [4].

1.3 Інтеграція IaC із безсерверними технологіями

Інфраструктура як код (IaC) є критично важливим інструментом для впровадження та управління безсерверними застосунками, завдяки своїй

здатності стандартизувати та автоматизувати процеси налаштування. Використання IaC дозволяє описати всі ресурси, необхідні для безсерверної архітектури, у вигляді коду, включаючи обчислювальні функції, ресурси API Gateway, бази даних і тригери подій. Це значно спрощує масштабування застосунків, оскільки нові середовища можна розгорнути за лічені хвилини, використовуючи ті ж самі конфігурації [9].

IaC дозволяє створювати шаблони для розгортання складних систем без ризику ручних помилок. Наприклад, інтеграція функцій AWS Lambda із системами зберігання даних, такими як S3 або DynamoDB, може бути виконана через Terraform або Pulumi. Завдяки цьому процес розгортання стає повторюваним і повністю автоматизованим, незалежно від складності архітектури [10].

Ще однією важливою перевагою IaC є інтеграція з системами безперервної інтеграції та доставки (CI/CD). Використовуючи IaC, інфраструктура може бути включена у весь цикл розробки, починаючи з етапу тестування і закінчуючи продуктивним розгортанням. Це забезпечує швидке та передбачуване впровадження змін. Наприклад, розробники можуть створювати нові функції та миттєво розгортати їх у тестовому середовищі для перевірки, а після цього автоматично виводити їх на продуктивні сервери [12].

IaC також робить можливим впровадження DevSecOps підходу, оскільки конфігурації інфраструктури можуть бути автоматично перевірені на відповідність безпековим стандартам. Спеціалізовані інструменти, як-от Checkov або tfsec, дозволяють перевіряти IaC-код на наявність потенційних вразливостей, таких як відкриті порти або некоректні політики доступу. Таким чином, безсерверні застосунки стають не лише більш ефективними, але й безпечними [14], [15].

IaC також сприяє крос-функціональній співпраці, оскільки інфраструктурний код може бути спільно розроблений, перевірений і підтримуваний як командами розробників, так і адміністраторами. Це сприяє прискоренню розробки та зменшенню витрат на підтримку.

Ще один важливий аспект – залежність від постачальника послуг (vendor lock-in), оскільки функції часто розробляються із використанням специфічних API певного хмарного провайдера. У сучасних дослідженнях та практиці впровадження serverless-архітектур простежується низка додаткових тенденцій і технологічних викликів, що заслуговують окремого розгляду.

Serverless-архітектура на сьогодні охоплює не лише FaaS, а й так званий BaaS (Backend as a Service), що дозволяє розробникам використовувати готові сервіси для автентифікації, зберігання даних, аналітики тощо. При цьому поєднання FaaS і BaaS забезпечує створення повноцінних масштабованих систем із мінімальними зусиллями для підтримки інфраструктури.

Сучасні тренди serverless включають активний розвиток edge-сервісів, коли функції виконуються ближче до джерела даних (наприклад, на edge-локаціях), що дозволяє значно скоротити затримки (latency) і покращити продуктивність для IoT та mobile-застосунків. Одним із прикладів таких рішень є Cloudflare Workers або AWS Lambda@Edge.

Порівняльний аналіз архітектур показує, що serverless дедалі частіше застосовується не лише у web-додатках, а й у галузях агротехнологій, фінансів, e-commerce, де потрібна гнучка реакція на пікові навантаження, а вартість інфраструктури відіграє ключову роль. У системах аграрного моніторингу, наприклад, serverless дозволяє обробляти дані супутників або сенсорів подієво, заощаджуючи ресурси й забезпечуючи автоматичну реакцію на зміни стану полів.

Значна увага в сучасних дослідженнях приділяється проблематиці cold start. Запропоновано різні підходи до оптимізації запуску функцій: використання pre-warmed containers, застосування lightweight runtime (Vercel, Cloudflare), а також нові моделі розрахунку вартості з урахуванням холодних і гарячих стартів.

Варто також звернути увагу на питання спостережуваності та моніторингу serverless-додатків. Сучасні хмарні провайдери надають розвинені засоби для трасування, логування й аналітики (AWS X-Ray, Azure Monitor,

Google Stackdriver), що дозволяє ефективно виявляти й усувати аномалії у роботі застосунку.

Отже, у цьому розділі розглянуто ключові аспекти безсерверної архітектури та інфраструктури як коду (IaC), які формують основу для автоматизації розгортання сучасних застосунків. Безсерверний підхід спрощує управління інфраструктурою, дозволяючи автоматично масштабувати ресурси та знижувати витрати, тоді як IaC забезпечує стандартизацію, стабільність і інтеграцію з DevOps-процесами.

Підсумовуючи, безсерверна архітектура трансформується із вузькоспеціалізованого підходу у комплексну платформу для реалізації інноваційних цифрових продуктів, дозволяючи гнучко керувати ресурсами, скорочувати time-to-market та оптимізувати операційні витрати в різних галузях.

Поєднання цих технологій відкриває широкі можливості для створення ефективних, гнучких і безпечних програмних рішень, які відповідають сучасним вимогам бізнесу та технологій.

1.4 Висновки до першого розділу

У результаті комплексного аналізу теоретичних основ, що лежать в основі сучасних підходів до проєктування й експлуатації інформаційних систем у хмарних середовищах, було розглянуто ключові поняття безсерверної архітектури (serverless architecture), її принципи роботи, переваги та сфери застосування. Встановлено, що serverless-модель дозволяє зосередитися на бізнес-логіці застосунків і значно скоротити операційні витрати завдяки автоматичному масштабуванню ресурсів, гнучкій оплаті за фактичне споживання та мінімізації потреб у ручному адмініструванні.

Особливу увагу приділено аналізу основних викликів, з якими стикаються організації під час впровадження serverless-підходу, зокрема: зростання складності управління розподіленими компонентами, забезпечення безпеки, контролю доступу, моніторингу та відстеження життєвого циклу ресурсів. Для вирішення цих проблем розглянуто концепцію «інфраструктура як код» (IaC), яка на сучасному етапі є невід'ємною частиною ефективного DevOps-процесу.

Деталізовано переваги IaC – стандартизація та формалізація налаштувань інфраструктури, автоматизація розгортання, легкість відтворення середовищ, інтеграція з системами контролю версій, скорочення часу на оновлення та зменшення кількості людських помилок. Проаналізовано типові сценарії застосування IaC у поєднанні з serverless-платформами, виділено основні інструменти (Terraform, Pulumi, AWS CloudFormation, Ansible) та оцінено їхній потенціал щодо побудови масштабованих, надійних і безпечних систем.

Розгляд технологій хмарних платформ (AWS, Azure, Google Cloud) дав змогу порівняти функціональні можливості провідних сервісів serverless, їхню інтеграцію з інструментами IaC, політику безпеки та особливості тарифікації. З'ясовано, що саме комбінація сучасних платформ і розвинених засобів автоматизації відкриває нові можливості для цифровізації бізнесу, підвищення продуктивності та оптимізації витрат.

Отримані теоретичні знання й аналітичні висновки є фундаментом для подальшого обґрунтування вибору архітектурних і технологічних рішень у наступних розділах.

РОЗДІЛ 2. АНАЛІЗ ІНСТРУМЕНТІВ І ТЕХНОЛОГІЙ ДЛЯ АВТОМАТИЗАЦІЇ БЕЗСЕРВЕРНИХ АРХІТЕКТУР

2.1 Порівняння платформ для безсерверних рішень

Безсерверні платформи та інструменти інфраструктури як коду (IaC) відіграють ключову роль у сучасній розробці й автоматизації програмних рішень. Серед основних платформ для безсерверних обчислень особливо вирізняються AWS Lambda, Azure Functions та Google Cloud Functions.

AWS Lambda

AWS Lambda - одна з найперших платформ для безсерверних обчислень, відома своєю високою інтеграцією в екосистему Amazon Web Services (AWS). Цей сервіс дозволяє виконувати функції у відповідь на події, як-от HTTP-запити, оновлення баз даних у DynamoDB (повністю керована, високопродуктивна хмарна база даних NoSQL, яка забезпечує швидкий доступ до даних із низькою затримкою) або завантаження файлів до S3 (сервіс об'єктного зберігання, який дозволяє зберігати й отримувати дані будь-якого обсягу та формату з будь-якого місця).

AWS Lambda забезпечує автоматичне масштабування, яке дозволяє одночасно виконувати до 1000 запитів за секунду, з можливістю збільшення цього ліміту. Це ідеальний вибір для задач, що потребують масштабованості без необхідності управління серверною інфраструктурою. Хоча час "холодного старту" для стандартних середовищ може становити близько 100–200 мс, використання контейнерів значно розширює можливості платформи. AWS Lambda підтримує мови JavaScript, Python, Go, Java, Ruby, C# і Docker-контейнери, що забезпечує гнучкість для розробників [6].

AWS Lambda знаходить широке застосування у різноманітних сценаріях. Одним із поширених прикладів є автоматична обробка даних, наприклад, створення мініатюр для зображень, завантажених у S3. Завдяки цьому функції

можуть швидко обробляти файли без необхідності підтримки постійно активної інфраструктури.

Інший напрямок використання – це розробка API для мобільних та веб-додатків. AWS Lambda інтегрується з API Gateway, дозволяючи створювати масштабовані серверні рішення, які можуть обробляти тисячі запитів одночасно.

Крім того, Lambda часто застосовується для обробки журналів і логів за допомогою сервісів Kinesis або SQS. Це дозволяє обробляти великі обсяги даних у реальному часі, наприклад, для аналітики клієнтської активності або моніторингу систем.

Як приклад практичного використання можна навести досвід компанії Coca-Cola, яка успішно впровадила AWS Lambda для автоматизації роботи своїх торгових автоматів. Завдяки цій платформі вдалося забезпечити стабільну обробку транзакцій навіть у періоди максимальних навантаження. Модель "pay-per-use", за якою клієнти оплачують лише час виконання функцій, дозволила компанії скоротити витрати на інфраструктуру на 40% [6].

Azure Functions

Azure Functions – це потужний інструмент від Microsoft, який виділяється гнучкістю та інтеграцією з корпоративними продуктами. Його головною перевагою є підтримка різних тригерів, зокрема HTTP-запитів, змін у базах даних (CosmosDB - це глобально розподілена, повністю керована база даних від Microsoft Azure, створена для роботи з високопродуктивними та масштабованими додатками) та черг повідомлень (Service Bus).

Azure Functions працюють за двома тарифними планами:

- Consumption Plan - оплата лише за використання ресурсів;
- App Service Plan - передбачає фіксовані витрати для стабільних навантажень.

Платформа підтримує основні мови програмування, як-от JavaScript, Python, C#, Java, а також пропонує інтеграцію з Active Directory, Power BI та Logic Apps, що робить її ідеальною для автоматизації бізнес-процесів [7].

Azure Functions широко використовується для вирішення задач, що вимагають автоматизації та швидкої обробки подій. Одним із поширених сценаріїв є моніторинг змін у базах даних, таких як CosmosDB. Функції дозволяють створювати автоматичні резервні копії або генерувати звіти на основі оновлених даних без потреби втручання користувача.

Ще однією важливою сферою є управління IoT-пристроями. Azure Functions обробляє запити з пристроїв, інтегрує їх із хмарними аналітичними системами та забезпечує зручну взаємодію із системами моніторингу.

Платформа також активно використовується для створення серверних компонентів для бізнес-додатків, зокрема чат-ботів, які обробляють запити клієнтів або виконують автоматичні бізнес-процеси.

Міжнародний банк HSBC впровадив Azure Functions для автоматизації обробки платіжних операцій. Це дозволило скоротити час виконання транзакцій із кількох годин до хвилин, значно підвищивши ефективність роботи фінансових систем. Використання Azure Functions також забезпечило гнучкість у масштабуванні у періоди пікової активності [7].

Google Cloud Functions

Google Cloud Functions – це сервіс, який забезпечує інтеграцію з екосистемою Google Cloud Platform (GCP) та відмінно підходить для роботи з потоковими даними та аналітикою.

Функції виконуються з мінімальною затримкою (середній холодний старт – менше 50 мс) і підтримують тісну інтеграцію з такими сервісами, як Pub/Sub (для асинхронних повідомлень), Firestore (бази даних) і BigQuery (аналітика даних). Крім того, Google Cloud Functions відповідає стандартам ISO/IEC 27001, що робить її привабливою для фінансових і медичних додатків [8].

Google Cloud Functions використовується у багатьох сценаріях, зокрема в задачах, пов'язаних із аналітикою даних та інтеграцією додатків. Одним із прикладів є автоматизація обробки повідомлень через Pub/Sub – сервіс асинхронного обміну повідомленнями в Google Cloud. Це дозволяє обробляти

великі обсяги даних у режимі реального часу, що корисно для моніторингу подій або виконання бізнес-логіки на основі потоків даних.

Ще один напрямок – виконання аналітичних запитів у BigQuery, який є високопродуктивною хмарною базою даних для аналітики. BigQuery дозволяє швидко обробляти величезні обсяги структурованих даних, наприклад, для аналізу бізнес-процесів чи побудови звітів. Використовуючи Cloud Functions, запити до BigQuery можна автоматизувати, запускаючи їх у відповідь на певні події.

Cloud Functions також підтримує створення веб-хуків – механізму для автоматичного обміну даними між додатками через HTTP-запити. Веб-хуки використовуються для інтеграції з CRM-системами або сторонніми сервісами, що дозволяє автоматизувати взаємодію та зменшити залежність від ручних процесів.

Прикладом використання Google Cloud Functions є впровадження цієї платформи компанією Spotify для обробки дій користувачів у режимі реального часу. Завдяки веб-хукам та інтеграції з аналітичними інструментами сервіс миттєво створює персоналізовані рекомендації треків, що забезпечує ефективність і високу продуктивність навіть при значних обсягах даних [8].

OpenFaaS

OpenFaaS – це open-source платформа, яка виступає альтернативою комерційним хмарним рішенням і дозволяє запускати функції як сервіси у власних локальних середовищах або на Kubernetes (платформа, що дозволяє автоматизувати розгортання, масштабування та управління контейнеризованими додатками). Вона забезпечує гнучкість завдяки використанню Docker-контейнерів, що дозволяє розробникам легко створювати, розгортати та управляти функціями.

Однією з ключових переваг OpenFaaS є підтримка інтеграції з Prometheus (система моніторингу та збору метрик, яка дозволяє відстежувати стан інфраструктури, функцій та інших компонентів у реальному часі) для збору метрик і Grafana (інструмент для візуалізації даних, зібраних Prometheus, у

вигляді графіків, таблиць і дашбордів) для візуалізації, що забезпечує зручний моніторинг виконання функцій та інфраструктури. Ця особливість робить платформу ідеальною для компаній, які прагнуть повного контролю над своїми системами.

OpenFaaS знаходить широке застосування у сценаріях, де потрібна автономність, локальне управління та оптимізація витрат.

Одним із напрямків використання є реалізація локальних корпоративних процесів. Платформа дозволяє автоматизувати внутрішні робочі потоки, наприклад, обробку запитів співробітників чи управління документами. Завдяки відсутності залежності від зовнішніх хмарних сервісів компанії можуть забезпечити контроль над чутливими даними та знизити ризик витоків.

Ще однією важливою сферою є виконання функцій на периферійних пристроях (Edge Computing). OpenFaaS дозволяє розміщувати обчислення ближче до джерела даних, що значно знижує затримки у передачі інформації. Це особливо корисно для обробки даних із IoT-пристроїв, таких як датчики, камери чи контролери, де потрібна швидкість і локальна автономність.

Крім того, OpenFaaS використовується для створення власних хмарних середовищ, які не залежать від великих хмарних провайдерів. Це рішення ідеально підходить для компаній із високими вимогами до безпеки, конфіденційності або для організацій із обмеженим бюджетом, які хочуть уникнути високих витрат на комерційні хмарні сервіси.

Компанія Vision AI застосувала OpenFaaS для обробки потокового відео з камер спостереження. Це рішення дозволило проводити обробку даних у реальному часі без використання комерційних хмарних платформ. Завдяки OpenFaaS вдалося значно зменшити затримки у передачі даних та знизити витрати на інфраструктуру, забезпечуючи при цьому високу продуктивність і автономність роботи [5].

AWS Lambda, Azure Functions і Google Cloud Functions – це провідні платформи для реалізації безсерверних рішень у хмарному середовищі. Вони пропонують унікальні можливості для автоматизації, аналітики та інтеграції з

іншими сервісами. OpenFaaS, як open-source рішення, пропонує незалежність від хмарних провайдерів і забезпечує контроль над інфраструктурою, що робить її ідеальним вибором для локальних і периферійних обчислень.

Кожна з платформ має свої сильні сторони, і вибір залежить від специфіки завдань, необхідного рівня інтеграції, продуктивності та вимог до безпеки.

2.2 Інструменти IaC

Інфраструктура як код (IaC) – це ключовий підхід до автоматизації управління IT-інфраструктурою. Він дозволяє описувати інфраструктуру за допомогою коду, що забезпечує повторюваність, прозорість і контроль над процесами розгортання. Для реалізації IaC існує багато інструментів, серед яких найбільш популярними є Terraform, AWS CloudFormation, Pulumi та Ansible. Кожен із них має свої особливості та сфери застосування.

Terraform

Terraform – це мультихмарний інструмент інфраструктури як коду (IaC), розроблений компанією HashiCorp, який використовується для автоматизації управління інфраструктурою в хмарних і локальних середовищах. Інструмент підтримує роботу з багатьма провайдерами, включаючи AWS, Azure, Google Cloud і Kubernetes, що дозволяє створювати рішення, здатні працювати одночасно в кількох хмарних середовищах.

Головною особливістю Terraform є декларативний підхід до опису інфраструктури. Усі ресурси визначаються в текстових файлах, які можна зберігати в системах контролю версій, таких як Git. Це забезпечує прозорість і повторюваність налаштувань, спрощує аудит змін і дозволяє командам спільно працювати над конфігураціями.

Terraform використовує файл стану для відстеження поточного стану інфраструктури, що дозволяє виконувати оновлення без ризику порушення існуючих налаштувань. Це забезпечує ідемпотентність – повторне виконання одного і того ж коду не спричиняє зайвих змін у вже налаштованих ресурсах.

Інструмент також підтримує створення модулів, які можна повторно використовувати, що є особливо корисним для великих середовищ із подібними конфігураціями.

Terraform широко використовується для автоматизації задач, таких як створення інфраструктури для serverless-архітектур. Наприклад, за допомогою цього інструмента можна швидко налаштувати ресурси AWS, зокрема AWS Lambda, API Gateway та бази даних DynamoDB, мінімізуючи ручну роботу [9].

AWS CloudFormation

AWS CloudFormation – це інструмент автоматизації від Amazon Web Services, який дозволяє створювати, оновлювати та управляти ресурсами у межах екосистеми AWS за допомогою опису конфігурацій у вигляді шаблонів. Завдяки підтримці форматів YAML і JSON, CloudFormation забезпечує простоту структурування та документування архітектури.

Цей інструмент надає розробникам можливість автоматизувати процес створення інфраструктури, автоматично обробляючи залежності між компонентами. Наприклад, при створенні стека CloudFormation спершу налаштує мережеві ресурси (VPC), потім створить бази даних (RDS) і лише після цього налаштує serverless-функції (AWS Lambda), забезпечуючи узгодженість. У разі помилок під час налаштування система автоматично скасує зміни, повертаючи інфраструктуру до попереднього стану.

CloudFormation глибоко інтегрується з усіма основними сервісами AWS, включаючи S3, DynamoDB, API Gateway, що робить його універсальним рішенням для автоматизації як простих, так і складних архітектур. Інструмент дозволяє створювати повноцінні середовища за допомогою одного шаблону, спрощуючи управління навіть у великих проєктах [10].

Наприклад, CloudFormation може бути використаний для одночасного налаштування обчислювальних ресурсів EC2, баз даних RDS і serverless-компонентів Lambda в межах одного проєкту. Це забезпечує швидкість і передбачуваність у створенні масштабованих архітектур.

Pulumi

Pulumi – це сучасний інструмент для інфраструктури як коду (IaC), який застосовує імперативний підхід до управління інфраструктурою. На відміну від декларативних рішень, Pulumi дозволяє описувати інфраструктуру за допомогою популярних мов програмування, таких як Python, JavaScript, TypeScript, або Go, що значно спрощує інтеграцію у вже існуючі робочі процеси.

Pulumi підтримує роботу з мультихмарними середовищами, включаючи AWS, Azure, Google Cloud, Kubernetes, та інші платформи. Це дозволяє створювати універсальні рішення, які охоплюють кілька хмарних провайдерів. Завдяки використанню умов, циклів і функцій мови програмування Pulumi забезпечує значну гнучкість у налаштуванні інфраструктури, що робить його зручним для складних сценаріїв.

Інструмент також має інтеграцію з DevOps-інструментами, такими як GitHub Actions, GitLab, або Jenkins, дозволяючи автоматизувати розгортання інфраструктури у CI/CD-процесах. Крім того, Pulumi підтримує управління станом інфраструктури, що допомагає гарантувати узгодженість ресурсів [11].

Pulumi особливо корисний для команд, які хочуть використовувати звичні мови програмування для автоматизації процесів. Наприклад, за допомогою Pulumi можна створити кластер Kubernetes, налаштувати serverless-функції AWS Lambda і забезпечити їх інтеграцію з базами даних у межах одного сценарію.

Цей інструмент стає вибором розробників, які шукають гнучке рішення з потужною підтримкою сучасних хмарних платформ.

Ansible

Ansible – це популярний інструмент автоматизації, створений компанією Red Hat, який дозволяє ефективно управляти конфігураціями, розгортати додатки та оркеструвати складні системи. Використовуючи імперативний підхід, Ansible надає розробникам і адміністраторам простий спосіб створення

автоматизованих сценаріїв, які можна швидко реалізувати без значного технічного навантаження.

Однією з ключових особливостей Ansible є модель без агентів. Інструмент працює через SSH, що виключає потребу встановлення додаткового програмного забезпечення на цільових вузлах, знижуючи складність і підвищуючи безпеку. Конфігурації описуються за допомогою Playbook у форматі YAML. Це зрозумілий, легкий для читання формат, який дозволяє описувати сценарії у вигляді послідовних завдань, що робить Ansible доступним навіть для новачків.

Модульність Ansible забезпечується великою бібліотекою готових модулів, які підтримують різноманітні хмарні платформи, операційні системи, мережеві пристрої та програмні рішення. Завдяки цьому Ansible легко інтегрується з такими системами, як AWS, Kubernetes, Docker, а також може управляти Windows- і Linux-серверами. Інструмент також забезпечує ідемпотентність, гарантуючи, що повторне виконання сценарію не призведе до зайвих змін, якщо система вже відповідає бажаному стану [12].

Ansible широко використовується для автоматизації завдань, таких як налаштування серверів, встановлення програмного забезпечення або управління контейнерами. Наприклад, інструмент може бути застосований для конфігурації веб-сервера Apache, налаштування кластерів Kubernetes або інтеграції хмарних ресурсів AWS. Завдяки простоті використання і широким можливостям, Ansible став одним із найпопулярніших рішень для автоматизації у світі DevOps.

Отже, кожен із інструментів має свої переваги, які роблять його ефективним для конкретних сценаріїв і завдань. Вибір залежить від потреб проєкту, компетенцій команди та хмарного середовища:

- Terraform - найкращий вибір для мультихмарних середовищ і великих проєктів із повторюваною інфраструктурою;
- AWS CloudFormation - ідеально підходить для екосистеми AWS завдяки глибокій інтеграції та автоматизації створення залежностей;

- Pulumi - зручний для розробників, які віддають перевагу використанню звичних мов програмування, ідеальний для команд із досвідом DevOps;
- Ansible - оптимальний для задач автоматизації конфігурацій серверів, налаштування додатків і управління контейнерами.

2.3 Вибір інструментів для практичної частини

Для забезпечення коректної роботи системи «Agro Monitor» та досягнення запланованих функціональних можливостей, до архітектури й інструментального стеку було висунуто такі основні вимоги:

1) система повинна забезпечувати можливість вибору та виділення аграрних ділянок на карті, налаштування параметрів супутникового аналізу (тип культури, NDVI, хмарність, часовий діапазон) та автоматизованого запуску обробки даних;
2) необхідно реалізувати асинхронну обробку супутникової інформації, формування тематичних карт (NDVI, стрес-зони), генерацію супутникових знімків та створення текстових агрономічних звітів;
3) система має підтримувати зберігання, організацію та візуалізацію отриманих результатів із можливістю інтеграції з іншими IT-сервісами й завантаженням звітів у різних форматах;
4) рішення повинно залишатися працездатним за значного зростання навантаження (наприклад, у період сезонного максимальної аграрної активності) та легко відновлюватися після збоїв;
5) з боку розробника важливо забезпечити простоту розгортання та оновлення всіх компонентів системи, використання інструментів для автоматизації інфраструктури, що підтримують повторюваність, контроль версій і швидке масштабування.

Враховуючи ці вимоги, було поставлено завдання вибрати такі інструменти й сервіси, які дозволяють досягти оптимального балансу між гнучкістю архітектури, автоматизацією розгортання, простотою підтримки та

економічною ефективністю використання хмарних ресурсів для реалізації всіх бізнес-процесів Agro Monitor.

З огляду на зазначені вимоги, проведено аналіз сучасних хмарних платформ і засобів автоматизації управління інфраструктурою для виявлення їх сильних та слабких сторін щодо реалізації подібних проектів. Основну увагу було приділено критеріям масштабованості, гнучкості, підтримки serverless-компонентів, можливостей інтеграції, зручності автоматизації розгортання та економічної доцільності.

Порівняльні характеристики провідних хмарних платформ для реалізації serverless-архітектур наведено у таблиці 2.1, а основних інструментів інфраструктури як коду (IaC) – у таблиці 2.2.

Таблиця 2.1

Порівняння платформ для безсерверних рішень

Характеристика	AWS Lambda	Azure Functions	Google Cloud Functions	OpenFaaS
Хмарна екосистема	AWS	Microsoft Azure	Google Cloud	Відсутня (локальна або Kubernetes)
Події запуску (тригери)	HTTP-запити, зміни в S3, DynamoDB, API Gateway	HTTP-запити, зміни у файлах, події в базах (CosmosDB), черги повідомлень (Service Bus)	HTTP-запити, події у Pub/Sub, Firestore, BigQuery	HTTP-запити, події на периферії (Edge), Docker-контейнери
Підтримка мов	JavaScript, Python, Go, Java, Ruby, C#, Docker-контейнери	JavaScript, Python, C#, Java, PowerShell	JavaScript, Python, Go, Java	Будь-яка мова (через Docker-контейнери)

Автоматичне масштабування	Динамічне, до 1000 паралельних виконань (можливе збільшення за запитом)	Динамічне, за Consumption Plan, або статичне за App Service Plan	Динамічне, висока продуктивність, затримка менше 50 мс	Обмежене ресурсами серверів, можлива інтеграція з Kubernetes для масштабування
Модель оплати	Pay-per-use (оплата лише за час виконання)	Consumption Plan (оплата за використання) або App Service Plan (фіксована плата за ресурси)	Pay-per-use	Витрати залежать від локальних ресурсів
Інтеграція	Глибока з іншими сервісами AWS: S3, DynamoDB, Step Functions	Інтеграція з Active Directory, Logic Apps, Power Platform, Azure DevOps	Тісна інтеграція з Pub/Sub, Firestore, BigQuery	Prometheus (моніторинг), Grafana (візуалізація)
Основні сценарії	Обробка зображень, API через API Gateway, автоматизація ETL-процесів	Інтеграція IoT, моніторинг баз даних, обробка черг повідомлень	Автоматизація аналітики даних, веб-хуки, обробка повідомлень у Pub/Sub	Обчислення на периферійних пристроях, локальна автоматизація процесів
Сильні сторони	Широкий вибір тригерів, гнучкість, глибока інтеграція з AWS	Підтримка корпоративних задач, інтеграція з Azure DevOps	Потужна обробка аналітичних даних, мінімальна затримка виконання функцій	Повна автономність, можливість уникнути vendor lock-in, підтримка Edge Computing
Слабкі сторони	Vendor lock-in (залежність від AWS), холодний старт (~100–200 мс для стандартних функцій)	Залежність від Azure, складність масштабування для великих навантажень	Vendor lock-in, обмежена гнучкість у кастомізації середовища	Вимагає налаштування інфраструктури, менше готових інтеграцій порівняно з хмарними платформами

Таблиця 2.2

Порівняння інструментів для інфраструктури як коду (IaC)

Характеристика	Terraform	AWS CloudFormation	Pulumi	Ansible
Підхід	Декларативний	Декларативний	Імперативний	Імперативний
Підтримка мультихмарності	Так	Ні (тільки AWS)	Так	Так (обмежена для інфраструктурних задач)
Мови конфігурації	HCL (HashiCorp Configuration Language), JSON	YAML, JSON	Python, TypeScript, Go, JavaScript	YAML
Модульність	Висока. Підтримує створення багаторазових модулів	Помірна. Залежить від шаблонів	Висока. Використовує код як модулі	Модульність через велику бібліотеку готових ролей
Стан інфраструктури	Відстежує файл стану для ідемпотентності	Автоматичний контроль залежностей	Відстежує стан як Terraform	Не має центрального зберігання стану
Глибока інтеграція з AWS	Так	Так	Так	Помірна
Простота налаштування	Вимагає базових знань HCL	Підходить для AWS-фахівців	Зручно для розробників із досвідом роботи з мовами програмування	Простий старт через YAML та SSH
Залежність від агента	Немає	Немає	Немає	Немає (використовує SSH)
Підтримка CI/CD	Так	Так	Так	Так

Сфера застосування	Мультихмарні середовища, автоматизація масштабної інфраструктури	AWS-екосистема, створення складних шаблонів	Гнучкі сценарії з інфраструктурою, що потребує логіки	Сервери, конфігурація ПЗ, контейнеризація

Розгляд інструментів та технологій для автоматизації інфраструктури та безсерверних рішень показав, що вибір підходящих платформ і засобів значною мірою залежить від потреб проєкту, вимог до гнучкості, масштабованості та технічних компетенцій команди.

AWS Lambda, Azure Functions та Google Cloud Functions забезпечують ефективну реалізацію безсерверної архітектури, дозволяючи швидко розгортати подієво-орієнтовані функції. Ці сервіси підтримують інтеграцію з широким спектром хмарних платформ та надають можливості для автоматичного масштабування, проте їх вибір може залежати від екосистеми, в якій працює організація.

Для управління інфраструктурою як кодом (IaC) інструменти, такі як Terraform, AWS CloudFormation, Pulumi та Ansible, надають широкий спектр можливостей для автоматизації та спрощення складних задач.

- Terraform вирізняється мультихмарністю та модульністю, забезпечуючи зручність створення інфраструктури, яка охоплює кілька хмарних провайдерів;
- AWS CloudFormation надає максимальну інтеграцію з AWS і дозволяє ефективно керувати ресурсами цієї платформи;
- Pulumi забезпечує гнучкість завдяки підтримці мов програмування, що робить його зручним для команд розробників із досвідом DevOps;
- Ansible ідеально підходить для автоматизації серверних конфігурацій, розгортання додатків і управління контейнерами, забезпечуючи простоту використання без необхідності встановлення агентів.

Таким чином, розглянуті платформи та інструменти надають усі необхідні можливості для автоматизації процесів розгортання, управління інфраструктурою та інтеграції додатків. Їх правильний вибір дозволяє оптимізувати ресурси, прискорити реалізацію проєктів та забезпечити їхню стабільність у довгостроковій перспективі.

2.4. Висновки до другого розділу

Проведений порівняльний аналіз платформ serverless-архітектур і засобів управління інфраструктурою як кодом (IaC) дозволив визначити оптимальні інструменти для практичної реалізації автоматизованого розгортання застосунку «Agro Monitor».

Враховуючи специфіку завдання, ключові вимоги до гнучкості, масштабованості, автоматизації, а також досвід роботи з різними хмарними платформами, у якості основного хмарного середовища було обрано Microsoft Azure. Це рішення обґрунтовується наявністю розвиненої екосистеми serverless-сервісів (Azure Functions, Azure Storage, Event Grid), глибокою інтеграцією з корпоративними технологіями, а також широкими можливостями для забезпечення безпеки й централізованого управління ресурсами.

Серед інструментів для інфраструктури як коду найбільш раціональним вибором став Pulumi. На відміну від декларативних рішень (як-от Terraform чи AWS CloudFormation), Pulumi дає змогу описувати інфраструктурні компоненти за допомогою повноцінних мов програмування (наприклад, TypeScript, Python), що значно спрощує підтримку коду, інтеграцію зі сторонніми бібліотеками, повторне використання логіки та автоматизацію процесів розгортання у рамках CI/CD.

Переваги Pulumi:

- можливість використання вже існуючих інженерних практик і патернів із розробки програмного забезпечення для опису інфраструктури;
- висока гнучкість під час формування умовних розгалужень, циклів та обробки помилок безпосередньо у коді;
- мультихмарна підтримка, що зберігає гнучкість у разі необхідності міграції або гібридних рішень.

Також при виборі інструментів враховувались такі критерії:

- легкість інтеграції із сучасними системами CI/CD (GitHub Actions, Azure DevOps);

- відкрита документація та наявність спільноти для швидкого пошуку рішень;
- відсутність vendor lock-in (залежності від одного провайдера) на рівні коду інфраструктури, що особливо важливо для тривалих проєктів.

Відтак, для реалізації практичної частини проекту було обрано зв'язку Microsoft Azure як платформу для розміщення безсерверних компонентів і Pulumi як засіб автоматизації IaC, що дозволило забезпечити ефективність розгортання, повторюваність процесів, простоту масштабування та підтримки системи «Agro Monitor» у хмарному середовищі.

РОЗДІЛ 3. АВТОМАТИЗАЦІЯ РОЗГОРТАННЯ ЗАСТОСУНКУ БЕЗСЕРВЕРНОЇ (SERVERLESS) АРХІТЕКТУРИ ДЛЯ СИСТЕМИ «AGRO MONITOR»

Основною предметною областю кваліфікаційної роботи є автоматизація розгортання конкретного безсерверного застосунку, реалізованого у хмарному середовищі з використанням інструментів інфраструктури як коду (IaC). Конкретний застосунок, що розглядається у межах цього дослідження, використовує сучасну архітектуру Serverless, де кожен компонент функціонує незалежно та може масштабуватись автоматично відповідно до навантаження.

У сучасних умовах аграрний сектор гостро потребує цифрових рішень для оперативного моніторингу стану земельних ресурсів, що дозволяють підвищити ефективність виробничих процесів, знизити витрати й забезпечити своєчасне прийняття управлінських рішень. Проте більшість традиційних IT-систем є маломасштабованими, складними в підтримці й часто не враховують особливостей сезонних максимальних навантаженнь, характерних для галузі.

У зв'язку з цим актуальною задачею є розробка та впровадження інформаційної системи для агромоніторингу, що поєднує переваги безсерверної архітектури, хмарної інфраструктури та автоматизованого управління ресурсами на базі підходу Infrastructure as Code (IaC).

3.1 Вибір і обґрунтування структури проєктування системи та її компонентів

Автоматизація процесів у сучасному агросекторі вимагає від інформаційних систем не лише надійності, але й виняткової гнучкості у масштабуванні, швидкої адаптації під змінне навантаження та контролю вартості. Саме ці вимоги визначили вибір безсерверної (serverless) архітектури для системи «Agro Monitor». Це рішення забезпечує максимально ефективне використання ресурсів хмарного середовища без необхідності безпосереднього

управління фізичними чи віртуальними серверами, що особливо актуально для систем зі змінною інтенсивністю обробки даних.

Основні переваги обраної структури полягають у наступному:

- *автоматичне масштабування* (система здатна миттєво реагувати на зростання чи спад навантаження, що дозволяє уникати перевантажень у пікові періоди агросезону та економити ресурси у періоди затишшя);
- *модель оплати pay-as-you-go* (споживач сплачує лише за фактичне використання потужностей, а не за постійну підтримку серверів у режимі очікування, що особливо вигідно для сезонних аграрних задач);
- *модульність та гнучкість масштабування* (кожен компонент працює автономно, що мінімізує ризики розповсюдження збоїв по всій системі та спрощує тестування й впровадження нових функцій);
- *прогнозована безпека та відповідність стандартам* (хмарна інфраструктура Azure гарантує виконання галузевих стандартів безпеки, а централізоване керування ресурсами підвищує захищеність даних).

З технічного погляду, основою інфраструктури стала концепція Infrastructure as Code (IaC), що впроваджена через платформу Pulumi. На відміну від таких інструментів, як Terraform чи AWS CloudFormation, Pulumi надає можливість описувати інфраструктурні ресурси звичними мовами програмування (Python, TypeScript), завдяки чому процес автоматизації розгортання, оновлення та адміністрування хмарної інфраструктури значно спрощується. Pulumi ефективно інтегрується з CI/CD-процесами, забезпечуючи швидке й безпечне розгортання змін у продуктивне середовище без ризику людської помилки.

Суттєвою перевагою використання Azure стало централізоване управління всіма ключовими компонентами (обчислювальні функції, черги, сховища) через декларативні IaC-інструменти. Таке рішення спрощує не тільки первинне розгортання, але й подальше масштабування або оновлення проєкту.

Варто також згадати окремі сервіси, що забезпечують безперебійну роботу й високу доступність системи: Azure Front Door виступає єдиною

точкою входу, балансує навантаження та інтегрує захист через Web Application Firewall, а Azure CDN забезпечує низьку латентність і швидке кешування статичних ресурсів для користувачів з різних регіонів.

Таким чином, стратегія побудови інфраструктури «Agro Monitor» базується на поєднанні передових хмарних підходів, що забезпечують надійність, масштабованість і контрольовані витрати. Детальна реалізація обраної архітектури розглядається у наступному розділі.

3.2 Основні рішення з реалізації системи «Agro Monitor» та її компонентів

На етапі імплементації система «Agro Monitor» була змодельована за принципом багатошарової стратифікації. Кожен із логічних шарів виконує чітко визначений набір функцій, що забезпечує ізоляцію навантажень і дозволяє системі залишатися стабільною навіть за умови високої інтенсивності запитів або виконання ресурсоємних алгоритмів.

Презентаційний (клієнтський) шар реалізовано на основі Next.js, який поєднує переваги SSR (server-side rendering) із гнучкістю знайомої екосистеми React. Такий вибір дозволяє формувати HTML-сторінки ще до повного завантаження клієнтського JavaScript-коду, що значно зменшує час відгуку для користувачів із повільним мобільним інтернетом. Крім того, інтеграція з react-leaflet дає можливість організувати ефективну роботу з геоданими та інтерактивними картами, зберігаючи при цьому простоту розробки й масштабування інтерфейсу.

Код конфігурації інтеграції карти у клієнтському компоненті, що забезпечує базову ініціалізацію карти для відображення агрономічних даних за координатами полів:

```
import { MapContainer, TileLayer, Marker } from "react-leaflet";
<MapContainer center={[48.3794, 31.1656]} zoom={6}
scrollWheelZoom={false}>
  <TileLayer
    url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
    attribution="&copy; OpenStreetMap contributors"
```

```
        />
        <Marker position={[48.3794, 31.1656]} />
      </MapContainer>
```

Сервісний (логічний) шар представлений HTTP-API на FastAPI, що розгортається у вигляді Azure Functions. FastAPI працює на асинхронному ASGI-стеку, забезпечує валідацію через Pydantic і автоматично формує OpenAPI-специфікацію. Це дає змогу мобільним і web-клієнтам отримувати передбачувану структуру даних і гарантує швидку інтеграцію з іншими системами. Для мінімізації cold-start часу на критичних маршрутах використовується Premium Plan з наперед прогрітими інстансами, що забезпечує середню затримку відповіді на рівні 200–300 мс.

Фрагменту функції для збереження результатів аналізу в Azure Blob:

```
      import { AzureFunction, Context, HttpRequest } from
"@azure/functions";
      import { BlobServiceClient } from "@azure/storage-blob";

      const httpTrigger: AzureFunction = async function (context: Context,
req: HttpRequest): Promise<void> {
        const blobServiceClient =
BlobServiceClient.fromConnectionString(process.env["AzureWebJobsStorage"]);
        const containerClient =
blobServiceClient.getContainerClient("results");
        const blobName = "report.json";
        const blockBlobClient =
containerClient.getBlockBlobClient(blobName);

        await blockBlobClient.upload(JSON.stringify(req.body),
req.body.length);
        context.res = {
          status: 200,
          body: "Report saved to Blob Storage!"
        };
      };
      export default httpTrigger;
```

Ця функція приймає запит із даними, обробляє його та зберігає результат у хмарному сховищі, що підвищує відмовостійкість системи та дозволяє легко масштабувати обробку даних.

Важливою особливістю реалізації є асинхронна обробка складних задач. Черга Azure Storage Queue забезпечує відокремлення миттєвої реакції API від тривалих обчислень. Після прийому запиту система одразу повертає ідентифікатор задачі, а її обробку виконує окремий воркер – Python-функція, що використовує бібліотеки rasterio, numpy, shapely для аналізу супутникових знімків Sentinel-2. Завдяки serverless-моделі кількість воркерів масштабується відповідно до черги задач: під час максимальних навантажень (наприклад, у період збору врожаю) Azure автоматично запускає десятки паралельних екземплярів і завершує їх після виконання.

Сховище даних організовано на базі Azure Blob Storage із чітким розділенням на raw-дані (GeoTIFF-знімки) та processed-результати (PNG-тайли, GeoJSON). Автоматизована політика переведення даних у холодні тири дозволяє оптимізувати витрати на зберігання без втрати доступності. Метадані звітів розміщуються у Azure Table Storage із мінімальною затримкою доступу (<10 мс).

Контент-доставка здійснюється через Azure CDN, що гарантує швидке отримання результатів агрономами по всій території України, незалежно від розташування основного сховища. Azure Front Door додатково шифрує трафік і визначає найоптимальнішу точку входу для кожного користувача, що покращує загальний користувацький досвід.

Інфраструктура як код централізовано підтримується у каталозі /infra/ репозиторію Git і реалізована на Pulumi (Python). Кожна зміна в інфраструктурі проходить рев'ю через pull-request, що забезпечує прозорість та контроль якості. Розгортання нових середовищ можливе за лічені хвилини, а оперативний моніторинг витрат відбувається через щоденні звіти Cost Analysis у Slack-каналі DevOps-команди.

Важливою особливістю побудованої архітектури є чітко визначена схема взаємодії між усіма компонентами системи, що наведена на рис. 3.1.

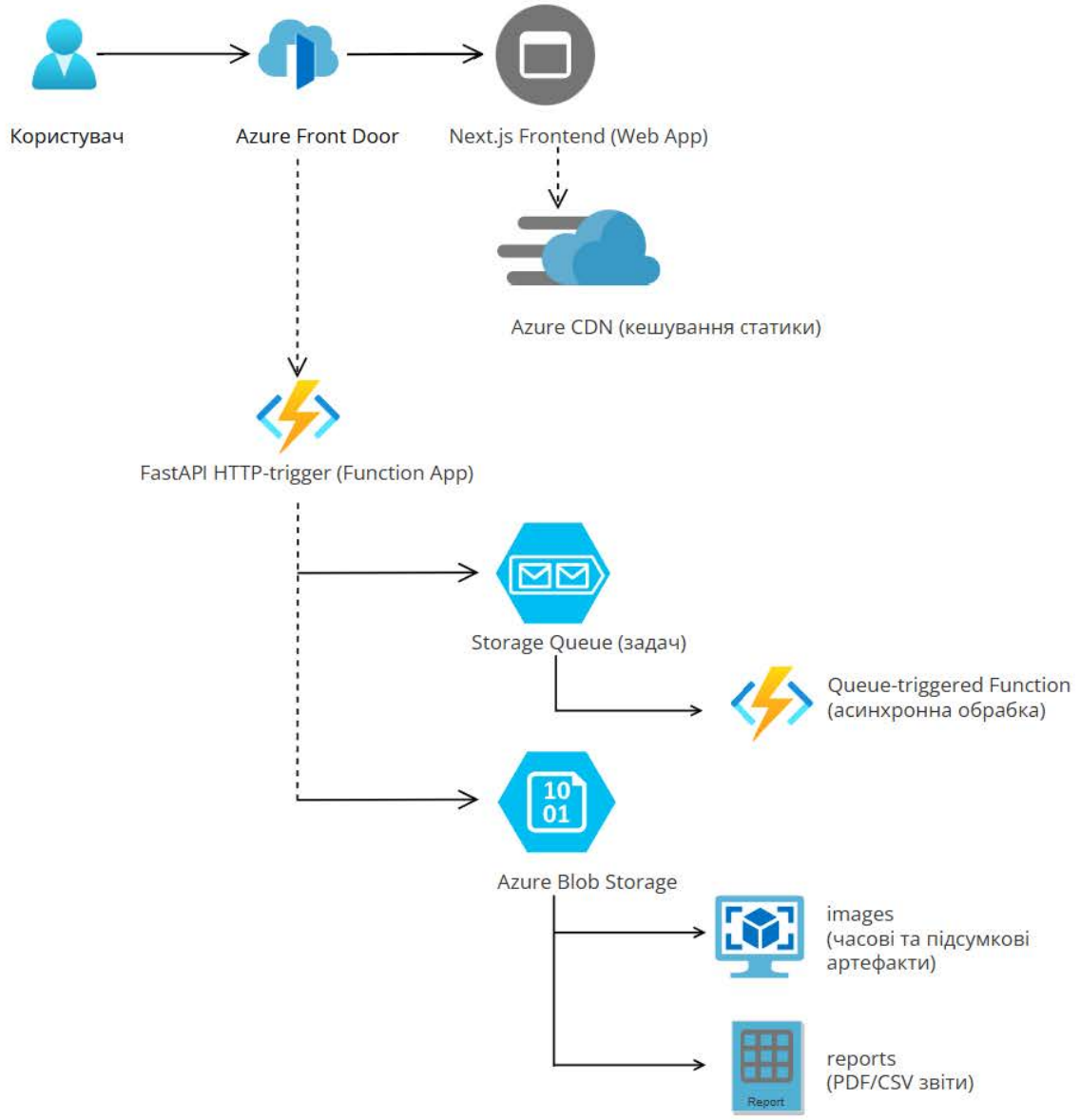


Рисунок 3.1 – Схема взаємодії основних компонентів системи «Agro Monitor»

На цій схемі візуалізовано основні інформаційні потоки: від ініціалізації користувацького запиту у веб-інтерфейсі до отримання результату після обробки супутникових знімків та візуалізації даних на карті. Взаємодія між компонентами побудована за допомогою стандартизованих API, асинхронних черг та єдиного сховища даних, що забезпечує гнучкість, надійність та прозорість роботи всієї системи.

Зокрема, користувач взаємодіє із клієнтською частиною, яка надсилає запит на аналіз земельної ділянки через захищене API. Серверна частина

приймає запит, здійснює необхідну валідацію, після чого створює завдання у черзі для асинхронної обробки. Воркери, що працюють незалежно від основного потоку запитів, виконують обробку супутникових даних, розраховують необхідні агрономічні індекси, формують результати і зберігають їх у сховищі. Після завершення обробки користувач отримує оновлену інформацію у веб-інтерфейсі у вигляді інтерактивної карти та супутньої аналітики.

Як видно з наведеної схеми, така організація забезпечує розмежування відповідальності між компонентами, дозволяє ефективно масштабувати підсистеми під час максимальних навантажень та гарантує стабільну роботу навіть за великої кількості одночасних користувачів. Описана архітектура створює міцний фундамент для подальшого розвитку функціоналу й оперативного реагування на нові виклики у сфері цифрового агромоніторингу.

На завершення, обраний стек технологій продемонстрував свою ефективність на практиці: користувач у реальному часі отримує інтерактивну NDVI-карту за кілька хвилин після формування запиту, а платформа автоматично пристосовується до зміни навантаження, забезпечуючи як економію бюджету, так і стабільність роботи. Незважаючи на деякі компроміси (наприклад, cold-start чи vendor lock-in), для аграрної сфери, що працює у сезонному ритмі, переваги serverless-підходу в масштабуванні, глобальній доставці й автоматизації безперечно переважають.

Детальний опис використаного системного програмного забезпечення, версій та ключових параметрів наведено у наступній главі.

3.3 Структура проєкту та організація компонентів

Проєктування і реалізація системи «Agro Monitor» спираються на ідеологію багаторівневої архітектури та сувору ізоляцію відповідальності кожного компонента. Такий підхід дозволяє досягати високої якості командної роботи, швидко впроваджувати нову функціональність і забезпечувати

стабільність навіть у складних сценаріях використання. Репозиторій проєкту структурований так, щоб кожна команда або розробник могли працювати автономно в межах своєї області відповідальності, а нові можливості та виправлення інтегрувалися максимально прозоро й без ризику порушити роботу системи в цілому. Вигляд файлової структури репозиторію проєкту «Agro Monitor» із коротким поясненням призначення основних каталогів та ключових файлів наведено на рис. 3.2.

```
agro-monitor/
│
├── frontend/                 # Клієнтська частина (Next.js)
│   ├── public/               # Статичні ресурси, іконки, шрифти
│   ├── src/                  # Вихідний код фронтенду (сторінки, компоненти, логіка)
│   │   ├── components/       # UI-компоненти
│   │   ├── pages/            # Сторінки Next.js
│   │   ├── hooks/            # Кастомні React-хуки
│   │   ├── utils/            # Допоміжні функції та утиліти
│   │   └── ...               # Інші каталоги/файли логіки
│   ├── package.json          # Опис залежностей і налаштувань JS-проєкту
│   └── ...                   # Конфігураційні файли Next.js, ESLint, TS/JS
│
├── backend/                  # Бекенд (FastAPI, Azure Functions)
│   ├── main.py               # Головна точка входу FastAPI
│   ├── api/                  # Опис маршрутів API
│   ├── models/               # Pydantic-моделі (схеми даних)
│   ├── services/             # Бізнес-логіка, взаємодія з воркерами та сховищем
│   ├── utils/                # Допоміжні скрипти та обробники
│   └── requirements.txt      # Python-залежності для бекенду
│
├── workers/                  # Асинхронні воркери для обробки задач
│   ├── ndvi_worker.py        # Воркер для розрахунку NDVI
│   ├── geo_worker.py         # Воркер для геоопераций (вирізка полігонів)
│   ├── utils/                # Допоміжні функції для воркерів
│   └── requirements.txt      # Python-залежності воркерів
│
├── infra/                    # Інфраструктура як код (Pulumi)
│   ├── __main__.py           # Головний файл сценарію Pulumi (опис усіх ресурсів)
│   ├── config/               # Конфігурації стеків (prod, dev тощо)
│   ├── requirements.txt      # Залежності для Pulumi-сценарію
│   └── Pulumi.yaml           # Основний конфігураційний файл Pulumi-проєкту
│
├── tests/                    # Автоматизовані тести (unit та інтеграційні)
│   ├── frontend/             # Тести для UI (Vitest, Playwright тощо)
│   ├── backend/              # Тести бекенду (Pytest)
│   └── workers/              # Тести воркерів
│
├── .github/                  # Налаштування GitHub Actions для CI/CD
│   └── workflows/            # YAML-файли workflow для автоматизації деплою та тестів
│
├── docker-compose.yaml       # Композиція для локального запуску всіх сервісів
├── README.md                 # Опис проєкту, інструкція для розгортання
└── .gitignore                # Ігнорування зайвих файлів у git
```

Рисунок 3.2 - Файлова структура репозиторію проєкту «Agro Monitor»

Вся логіка організації коду і сервісів будується на принципі розділення за ролями.

Клієнтська частина розташована у каталозі */frontend* і реалізована на базі Next.js. Це дозволяє комбінувати переваги серверного рендерингу для швидкого старту інтерфейсу із гнучкістю сучасної екосистеми React. Завдяки цій архітектурі інтерфейс працює коректно навіть у випадках нестабільного інтернет-з'єднання, а кожен його компонент ізольовано відповідає за свою частину візуалізації чи взаємодії з користувачем. Для інтеграції картографічного функціоналу застосовується react-leaflet із підтримкою OpenStreetMap, що забезпечує динамічне відображення полів, контурів і результатів супутникового аналізу, зручних для практичного використання в аграрній сфері.

Серверна логіка, сконцентрована в каталозі */backend*, реалізована на FastAPI і побудована за модульним принципом: маршрутизація, бізнес-сервіси, схеми валідації даних, менеджери для взаємодії з іншими підсистемами. Всі дані проходять сувору валідацію засобами Pydantic, що мінімізує ризик потрапляння помилкової інформації. Асинхронна архітектура дозволяє системі залишатися чутливою до максимальних навантажень, ефективно масштабуватися й швидко обробляти великі об'єми запитів. Важливо, що автоматична генерація OpenAPI-специфікації спрощує інтеграцію з мобільними застосунками або зовнішніми API-партнерами.

Асинхронна обробка ресурсомістких задач відокремлена у вигляді Python-модулів у каталозі */workers*. Кожен асинхронний процес – це ізольований модуль для реалізації окремого алгоритму: наприклад, обрізка супутникового знімка GeoJSON-полігоном чи генерація NDVI-мапи. Для роботи з растровими та векторними даними використовуються бібліотеки rasterio, numpy, shapely. Такий підхід не лише оптимізує використання обчислювальних ресурсів Azure, але й дозволяє динамічно масштабувати потужності залежно від навантаження на систему.

Інфраструктурний шар кодується і підтримується у каталозі */infra* за допомогою Pulumi на Python, що забезпечує декларативно-імперативний підхід

до Infrastructure as Code. Всі ключові хмарні ресурси – від сховищ до функцій та черг – описуються як програмний код, що проходить рев'ю через систему контролю версій. Будь-які зміни, додавання чи оновлення ресурсів фіксуються, і при необхідності можна відтворити стан інфраструктури у будь-який момент, або розгорнути клон продуктивного середовища за лічені хвилини. Pulumi інтегрується з CI/CD-конвеєрами, що дозволяє безперервно впроваджувати нові функції й підтримувати стабільність роботи навіть у складних сценаріях масштабування.

Тестування та забезпечення якості коду зосереджено у каталозі */tests*. Автоматизовані тести охоплюють як окремі модулі (unit-тести), так і складні інтеграційні сценарії, де перевіряється робота кількох підсистем у комплексі. Це значно спрощує процес впровадження нових функцій, допомагає оперативно виявляти й усувати помилки, а також гарантує відповідність реалізації початковим вимогам.

Всі ці підходи у підсумку не тільки прискорюють процес розробки та підтримки, а й підвищують якість інтеграції нових функцій і прозорість взаємодії між компонентами. Організація проєкту дозволяє команді розробників гнучко реагувати на запити користувачів, швидко тестувати гіпотези і розгортати оновлення без ризику для вже працюючої системи.

Щоб підкріпити опис логіки та архітектури, нижче наведено фрагмент коду API-ендпоїнта (повний вміст файлу *function_app.py*_ наведено у додатку Б) , реалізованого з використанням FastAPI. Він ілюструє, як система обробляє вхідні запити користувачів і забезпечує асинхронне виконання обчислювальних завдань:

```
@app.get("/api/ndvi/{field_id}")
async def calculate_ndvi(field_id: str):
    # Перевірка правильності ідентифікатора поля
    validate_field_id(field_id)
    # Додавання завдання до черги для асинхронної обробки
    enqueue_task(field_id)
    # Повернення статусу обробки
    return {"status": "processing", "field_id": field_id}
```

У цьому фрагменті чітко простежується розподіл обов'язків: API здійснює лише валідацію даних та ініціює обробку, тоді як безпосередні обчислення виконуються окремими асинхронними модулями (воркерами). Така ізоляція навантажень гарантує швидкий час відповіді інтерфейсу користувача і дозволяє ефективно масштабувати ресурси під час пікових навантажень.

Для управління взаємодією всіх компонентів використовується Docker Compose, що описує середовище виконання кожного модуля та їхні взаємозв'язки. У нижче наведеному фрагменті конфігураційного файлу *docker-compose.yml* (повний вміст файлу наведено у додатку Б) окремо визначені сервіси для API та бази даних, що дозволяє ефективно ізолювати їхню роботу та налаштувати масштабування залежно від навантаження. Така організація також спрощує процес оновлення окремих компонентів без зупинки всієї системи:

```
version: "3.9"
services:
  backend:
    build: ./backend
    environment:
      - DATABASE_URL=postgres://user:password@db:5432/agro
    depends_on:
      - db
  db:
    image: postgres:13
    volumes:
      - postgres_data:/var/lib/postgresql/data
volumes:
  postgres_data:
```

На рівні вибору основних компонентів та реалізації рішень, система «Agro Monitor» спирається на принципи модульності, гнучкості та повної автоматизації життєвого циклу – від опису інфраструктури до деплою і масштабування. Це дозволяє досягти як надійної роботи, так і оперативної адаптації під зміну вимог ринку чи специфічні особливості аграрної галузі. Таке організаційне та технічне підґрунтя створює не лише стабільну основу для роботи системи, але й сприяє безперервному розвитку, впровадженню нових технологій та оптимізації бізнес-процесів на базі сучасних хмарних рішень.

3.4 Опис використовуваного системного програмного забезпечення

Розробка та впровадження системи «Agro Monitor» здійснювалися на основі ретельного відбору сучасного системного програмного забезпечення, здатного задовольнити як функціональні, так і стратегічні вимоги проєкту. У процесі вибору враховувалися не лише технологічні характеристики, а й такі важливі критерії, як надійність, відтворюваність, масштабованість і відповідність концепції інфраструктури як коду (IaC) та безсерверної архітектури. Саме такий підхід дозволив уникнути вузьких місць і в процесі розгортання, і під час подальшої експлуатації, а також забезпечити стабільний контроль витрат, що особливо актуально для аграрного сектору з його сезонними максимальними навантаженнями.

Хмарна платформа та інфраструктурні сервіси

Ключову роль у забезпеченні гнучкості та автоматизації процесів відіграє хмарна платформа Microsoft Azure. Саме ця платформа була обрана як базова, оскільки надає широкий спектр сервісів для організації serverless-архітектури, а також глибоко інтегрується з інструментами автоматизації IaC. Можливість централізовано управляти всіма важливими компонентами – обчислювальними функціями, чергами, сховищами – через декларативний опис інфраструктури значно полегшує розгортання та масштабування, мінімізує людський фактор і забезпечує гнучкість для майбутнього розвитку проєкту. Додатково Azure гарантує високий рівень доступності сервісів та оптимізацію витрат завдяки моделі pay-as-you-go, що є важливим для систем з динамічним навантаженням.

Бекенд-середовище та виконання обчислень

Бекендова частина системи реалізована у вигляді серверлес-функцій на базі Azure Functions, що працюють у Linux-контейнерах з використанням Python 3.11. В якості фреймворка для побудови HTTP-API обрано FastAPI – рішення, що поєднує високу продуктивність з суворою типізацією і автоматичною валідацією даних за допомогою Pydantic. Архітектура бекенду передбачає чітке розмежування відповідальності: окремі ендпоїнти відповідають за роботу з

полями, індексами NDVI, а також асинхронними задачами. Такий поділ дозволяє ефективно масштабувати окремі підсистеми та швидко розширювати функціонал без ризику негативного впливу на інші компоненти. Додатковою перевагою serverless-підходу стало автоматичне масштабування обчислювальних ресурсів залежно від поточного навантаження, що дозволяє підтримувати стабільну роботу навіть у періоди сезонних піків активності користувачів.

Клієнтський інтерфейс та інтеграція з Web

Фронтенд системи побудовано на базі Next.js 15 із застосуванням сучасного JavaScript-стеку, що включає React 19 та SWR 2.3 для ефективного кешування та оновлення даних. Архітектура поєднує можливості статичної генерації сторінок та інкрементального оновлення, завдяки чому користувачі отримують актуальні дані навіть за нестабільного інтернет-з'єднання. Інтерфейс підтримує інтеграцію картографічного функціоналу – для цього використовується Leaflet разом із React-Leaflet-Draw, що дозволяє наочно відображати результати супутникового моніторингу й оперативно взаємодіяти із просторовими об'єктами.

Сховище даних та асинхронні черги

Сховище даних організовано на основі Azure Blob Storage з поділом на raw і processed шари, що дозволяє зберігати як сирі супутникові знімки (GeoTIFF), так і оброблені результати, такі як NDVI-растр чи тематичні маски. Запроваджена політика збереження та переведення застарілих даних у дешевші шари (Cool tier) дає можливість оптимізувати експлуатаційні витрати, не втрачаючи доступності важливої інформації. Для виконання об'ємних або тривалих обчислень використовується асинхронна обробка задач за допомогою Azure Queue Storage, що забезпечує стабільну роботу навіть при великій кількості одночасних запитів, оскільки кожне завдання динамічно підхоплюється і обробляється окремими воркерами.

Геопросторові та обчислювальні бібліотеки

Особлива увага при побудові системи приділялася геопросторовій аналітиці й обробці супутникових знімків. Для цього у складі бекенду використовується сучасний стек Python-бібліотек: planetary-computer і pystac-client дозволяють здійснювати пошук і агрегацію сцен Sentinel-2, а rasterio, numpy, shapely і pyproj відповідають за обробку, вирізання контурів полів та координатні перетворення. Такий набір інструментів забезпечує максимальну гнучкість і дозволяє адаптувати систему під потреби нових агрономічних алгоритмів або моделей аналізу.

Системи моніторингу та спостереження

Забезпечення стабільної роботи системи, її спостережність і контроль за основними показниками виконання реалізовано через Azure Application Insights та Log Analytics. Всі ключові події – від запуску функцій до фіксації винятків і завершення обробки задач – централізовано реєструються й доступні для оперативного аналізу за допомогою KQL-запитів. Додатково впроваджено підтримку OpenTelemetry у FastAPI, що дозволяє розробникам відстежувати ланцюжки викликів і своєчасно ідентифікувати потенційні вузькі місця або аномалії в роботі сервісів. Система моніторингу є основою для побудови прозорого DevOps-циклу, автоматичного сповіщення про інциденти й планування розвитку платформи.

Інфраструктура як код та автоматизація розгортання

Інфраструктура всієї системи описується і керується через Pulumi із використанням Python SDK. Такий підхід дозволяє централізовано керувати всіма ресурсами – від ресурсної групи до черг повідомлень і логів – і відтворювати середовище у випадку необхідності або для створення тестових оточень. Pulumi дозволяє поєднувати декларативний та імперативний стилі опису, що, у свою чергу, дає розробникам більше гнучкості під час розгортання чи оновлення інфраструктури. Стан усіх стеків зберігається у захищеному сховищі Pulumi Service, що спрощує контроль змін і історію розгортання.

CI/CD, тестування та локальна розробка

Процес складання, тестування та розгортання системи автоматизований через GitHub Actions, де для кожного з основних компонентів налаштовано окремі workflow. Весь ланцюжок включає етапи lint-контролю, тестування, створення релізних артефактів та їх автоматичний деплой у хмарне середовище. Під час кожного деплою у систему моніторингу передаються метадані (build-номер, git-hash), що дозволяє відстежувати історію версій та полегшує діагностику у разі виникнення проблем у продуктивному середовищі. Для локальної розробки застосовується Docker Compose, який дозволяє швидко запускати повний стек застосунку – з бекендом, фронтендом і емульованим сховищем – навіть без підключення до Azure. Вбудовані тести (Pytest, Vitest) перевіряють як окремі модулі, так і комплексні інтеграційні сценарії, що гарантує надійність кожної нової функції ще до моменту публікації.

Керування секретами та безпекою

Важливим аспектом є питання безпеки й управління секретами. Для захисту конфіденційних даних використовується Azure App Settings у поєднанні з Managed Identity, що забезпечує безпечний доступ до хмарних ресурсів. Усі секрети для процесів CI/CD зберігаються у GitHub Secrets, унеможливлюючи їхнє потрапляння у відкриті репозиторії. Окрему увагу приділено налаштуванню CORS-політик для захисту взаємодії між фронтендом і бекендом, а також реалізовано рольове управління доступом (RBAC) до критичних ресурсів – черг, сховищ, API – що мінімізує ризик несанкціонованих дій.

Загалом, вибір і впровадження системного програмного забезпечення для «Agro Monitor» – це результат глибоко зваженого підходу, який дозволяє об'єднати гнучкість сучасних технологій, контрольовану вартість експлуатації, надійність і високий рівень безпеки. Кожен компонент гармонійно інтегрується у єдину екосистему, що дозволяє не лише ефективно вирішувати завдання аграрної аналітики, а й гарантує безперервний розвиток та можливість оперативного реагування на нові виклики чи запити користувачів.

3.5. Інструкція роботи користувача з системою

Для доступу до веб-застосунку «Agro Monitor» відкрийте у браузері сторінку https://frontendc613cf2f.azurewebsites.net/. Система не потребує попередньої реєстрації або авторизації: кожен користувач одразу отримує доступ до основного функціоналу для аналізу полів.

Після завантаження сторінки відображається інтерактивна карта місцевості на базі OpenStreetMap (з використанням Leaflet). Для переміщення по карті використовуйте мишу (затисніть ліву кнопку) або сенсорну панель. Збільшення й зменшення масштабу доступні за допомогою колеса миші або кнопок «+» та «–» у лівому верхньому куті карти (див. рис. 3.3).

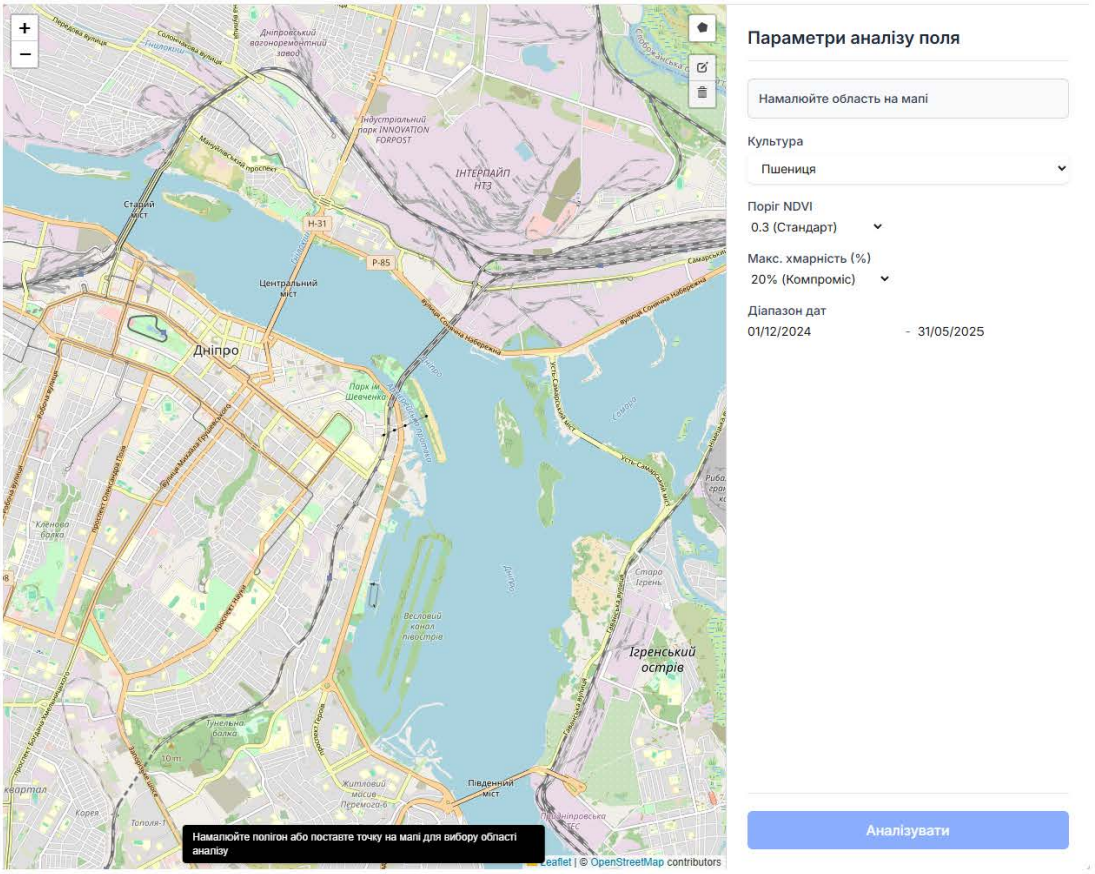


Рисунок 3.3 – Вигляд завантаженої сторінки системи «Agro Monitor»

Для запуску аналізу потрібно виділити полігон на карті, який відповідає ділянці поля для дослідження. Натисніть на кнопку  *«Намалюйте полігон»* (рис. 3.3, 3.4) та окресліть потрібну область.

Праворуч на екрані відкривається форма "Параметри аналізу поля". Заповніть такі поля:

Культура – виберіть зі списку культуру, яка вирощується на обраній ділянці (наприклад, «Пшениця»).

Поріг NDVI – оберіть бажаний NDVI-поріг для аналізу (наприклад, 0.3 – стандартне значення для оцінки рослинності).

Макс. хмарність (%) – вкажіть максимально допустимий відсоток хмарності на супутникових знімках (наприклад, 20% для оптимальної якості).

Діапазон дат – встановіть початкову і кінцеву дату для вибору архівних супутникових знімків, які будуть використані для аналізу поля (див. рис 3.4).

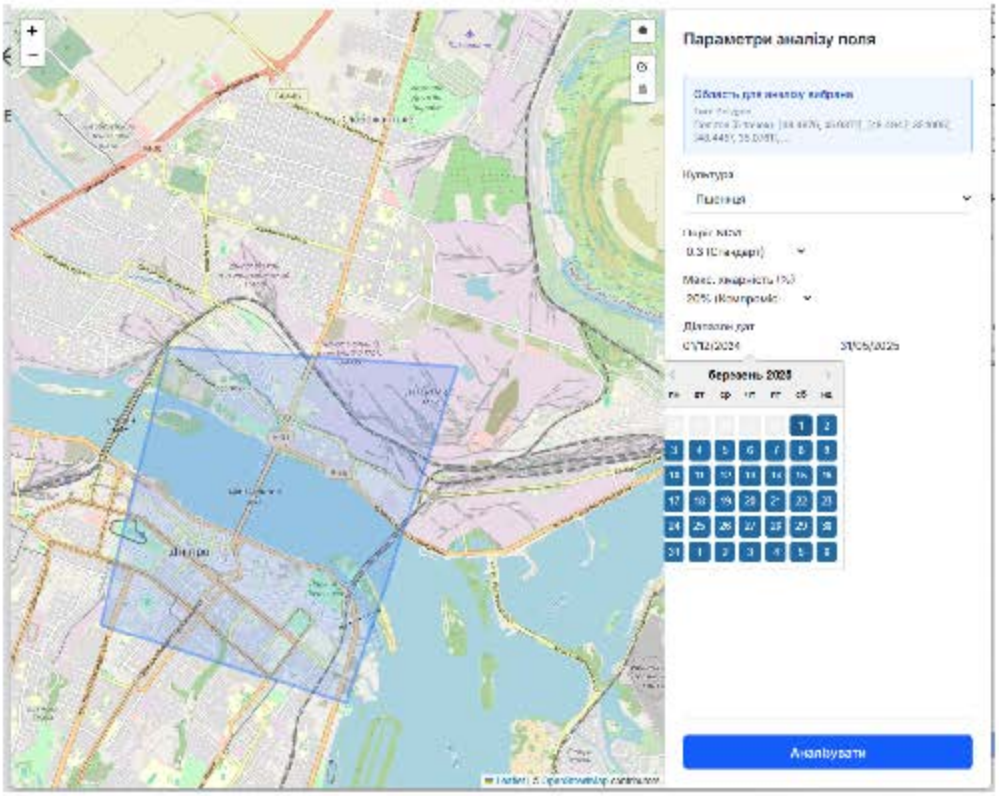


Рисунок 3.4 – Вигляд етап виділення ділянки поля та задання параметрів для обробки на сторінці системи «Agro Monitor»

Після заповнення параметрів стане доступною кнопка «Аналізувати» внизу форми. Натисніть її. Після того система почне обробляти запит. Користувач отримає повідомлення про прийняття задачі в роботу.

Аналіз виконується асинхронно: система формує унікальний ідентифікатор задачі. Протягом декількох хвилин (у залежності від розміру поля та вибраного діапазону дат) система проведе аналіз та сформує звіт.

Після завершення обробки користувач отримує доступ до детального звіту, що містить такі ключові елементи (див. рис. 3.5, 3.6):

- карта області аналізу із виділеним полігоном;
- основні параметри аналізу: тип культури, NDVI-поріг, хмарність, діапазон дат;
- площа аналізованої ділянки (у квадратних кілометрах);
- виявлені стрес-зони: відсоток площі, де спостерігаються ознаки стресу рослин (NDVI нижче порогу).

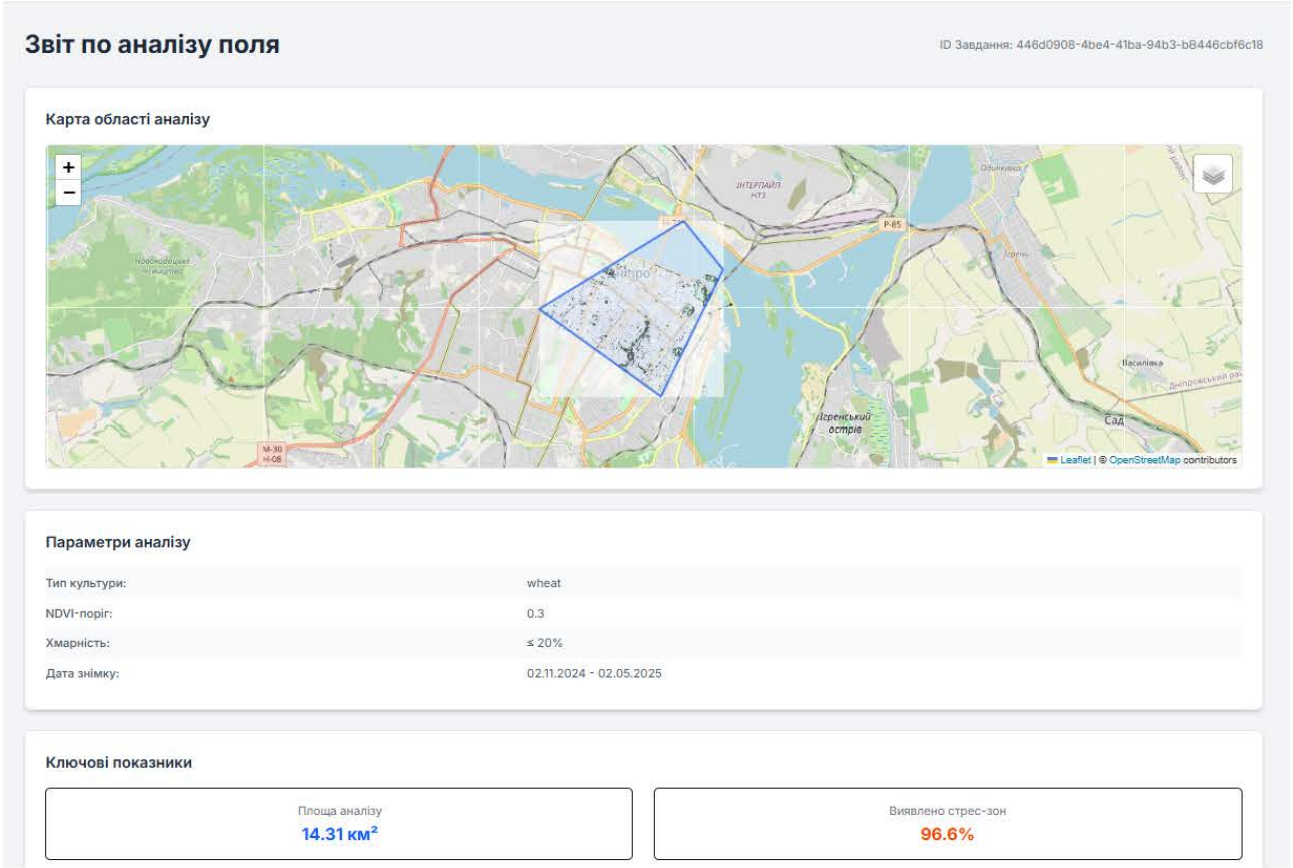


Рисунок 3.5 – Вигляд результатів роботи системи «Agro Monitor»

- супутникові знімки:
    - оригінальне RGB-зображення з Sentinel-2;
    - NDVI-карта – зображення індексу NDVI, де різними кольорами позначено рівень вегетації;
    - мапа стрес-зон – зони, де NDVI < порогового значення.
- текстовий звіт із оцінкою стану поля й короткими агрономічними рекомендаціями.

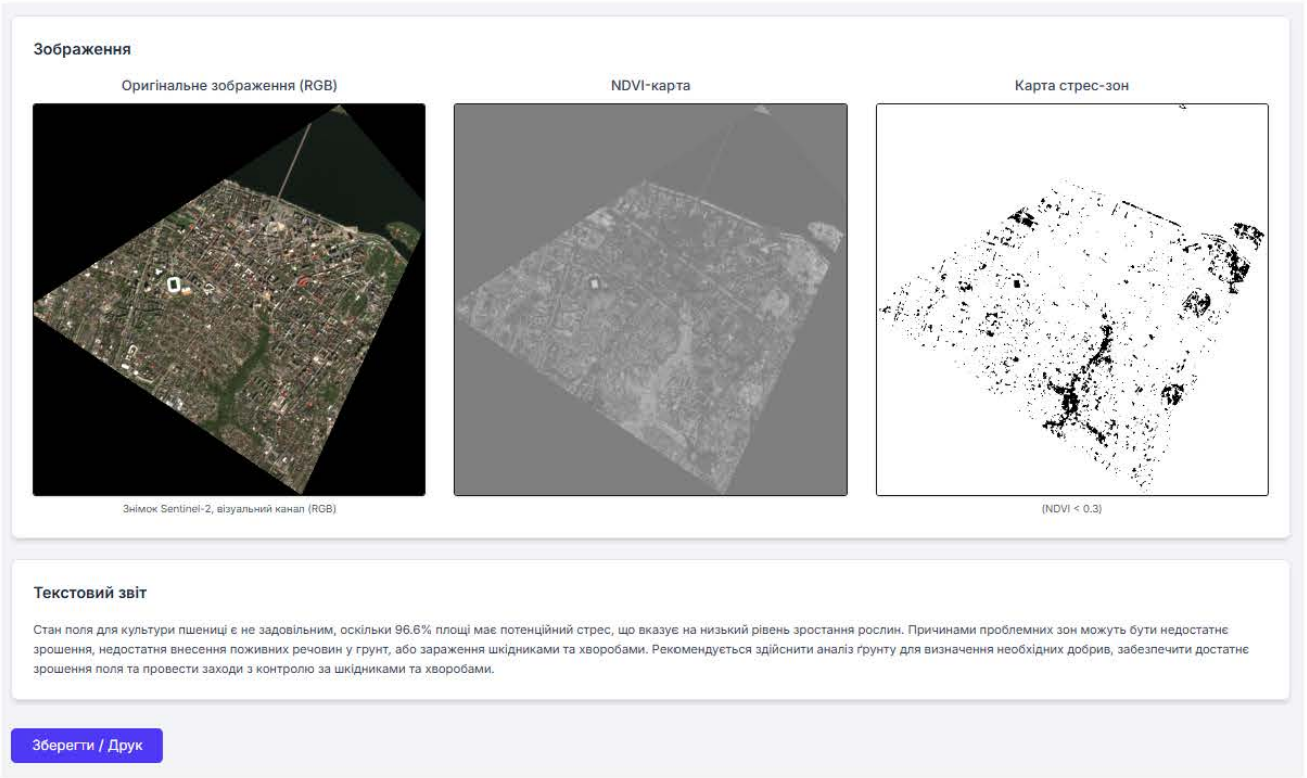


Рисунок 3.6 – Вигляд супутникових знімків, опрацьованих за допомогою системи «Agro Monitor»

Звіт можна відкрити за унікальним посиланням, яке генерується після завершення розрахунків, або у вікні системи після сповіщення.

Усі результати аналізу можна переглянути на карті, а також (за необхідності) завантажити:

- зображення NDVI-карт та оригінальні супутникові знімки (у форматі PNG або JPEG);
- текстові звіти (у форматі PDF або CSV – якщо ця функція ввімкнена для вашої ролі/акаунта). (Вигляд сформованого звіту наведено у додатку А.)

Особливі можливості та поради:

- для великих ділянок рекомендується використовувати менший діапазон дат або вищий поріг хмарності для швидкого отримання результату;
- для точнішого аналізу обирайте меншу максимально допустиму хмарність (до 20%) – це забезпечить якісніші знімки;.
- NDVI-поріг допомагає визначити стрес-зони: чим нижчий індекс, тим гірший стан вегетації рослин;
- після виконання аналізу унікальне посилання на звіт можна скопіювати для повторного перегляду або надсилання агрономам, керівникам тощо.

Поширені помилки та вирішення

- якщо на карті не вдається намалювати полігон – переконайтеся, що обрано інструмент малювання, а ділянка не перекриває вже виділену область;.
- якщо після натискання «Аналізувати» немає зворотного зв'язку – перевірте стабільність інтернет-з'єднання, а також чи правильно заповнені всі параметри форми;
- у разі виникнення технічних помилок або появи повідомлення про відсутність супутникових даних у заданому діапазоні дат – спробуйте змінити дати чи поріг хмарності.

3.6 Висновки до третього розділу

В результаті роботи було виконано повномасштабну практичну реалізацію системи «Agro Monitor» на основі сучасної serverless-архітектури з повною автоматизацією процесів розгортання та масштабування. Для досягнення поставлених цілей було обґрунтовано вибір багатошарової структури, де кожен

компонент системи – від клієнтського інтерфейсу до обчислювальних сервісів і сховищ – функціонує ізольовано, взаємодіє через стандартизовані API та масштабується відповідно до поточного навантаження

Практичний проєкт реалізовано на платформі Microsoft Azure із використанням таких технологій, як Next.js (клієнтська частина), FastAPI (логічний рівень), Azure Functions (асинхронна обробка даних), а для управління інфраструктурою застосовано IaC-інструмент Pulumi. Це дозволило досягти високої швидкості розгортання нових середовищ, гнучкості в інтеграції нових функцій, гарантованої відтворюваності та стабільної роботи навіть у періоди максимальних навантажень

Ключовим результатом впровадження є можливість автоматизованого збору, аналізу й візуалізації супутникових агроданих із наданням кінцевому користувачу інтерактивної карти поля, NDVI-карт, виявлення стрес-зон і формування текстового звіту з агрономічними рекомендаціями. Уся обробка відбувається асинхронно, а користувач отримує результати за унікальним посиланням або безпосередньо у веб-інтерфейсі, з можливістю завантаження зображень та звітів у різних форматах

Особливу увагу у проєкті приділено питанням модульності, ізоляції компонентів та CI/CD, що дало змогу організувати ефективну командну роботу, спростити впровадження оновлень і гарантувати надійність навіть при масштабуванні. Проектна структура, побудована за принципами розділення відповідальності, створює передумови для подальшого розвитку системи, впровадження нових сервісів та інтеграції із зовнішніми платформами

Таким чином, впроваджене рішення демонструє переваги сучасного підходу до цифрового агромоніторингу: мінімізацію операційних витрат, гнучке масштабування, простоту супроводу та швидку адаптацію під галузеві потреби. Практика довела ефективність автоматизації розгортання та serverless-парадигми для реальних завдань аграрної сфери, а отримані результати можуть стати основою для подальших впроваджень подібних систем у різних галузях.

ВИСНОВКИ

Кваліфікаційна робота присвячена розробці та впровадженню автоматизованої системи розгортання безсерверних застосунків із використанням концепції «інфраструктура як код» (IaC) на прикладі інформаційної системи «Agro Monitor» для аграрної галузі. У процесі виконання дослідження було вирішено комплекс завдань, що охоплюють аналіз теоретичних засад, вибір і обґрунтування технологічних рішень, практичну реалізацію та оцінку ефективності впровадженого підходу.

Розглянуто сучасні підходи до організації хмарної інфраструктури, зокрема serverless-архітектуру, яка дозволяє компаніям мінімізувати експлуатаційні витрати, підвищити гнучкість масштабування та зосередитися на бізнес-логіці замість підтримки серверної інфраструктури. Проаналізовано концепцію IaC як основного інструменту автоматизації, що забезпечує стандартизацію, прозорість та повторюваність процесів розгортання й супроводу IT-систем. Висвітлено переваги поєднання serverless-підходу з автоматизацією DevOps-процесів для прискорення впровадження інновацій та зниження кількості людських помилок.

Здійснено порівняльний аналіз провідних хмарних платформ (AWS, Azure, Google Cloud) і популярних IaC-інструментів (Terraform, Pulumi, AWS CloudFormation, Ansible). Вибір на користь Microsoft Azure та Pulumi обґрунтовано багатством серверлес-сервісів, широкою підтримкою інтеграцій, можливістю використання знайомих мов програмування для опису інфраструктури та простотою вбудовування в CI/CD-процеси.

На основі аналізу бізнес-процесів і функціональних потреб сформульовано вимоги до системи Agro Monitor. Основний акцент зроблено на масштабованості, асинхронній обробці великих обсягів супутникових даних, автоматизації збору, аналізу й візуалізації агроінформації. Було розроблено багатошарову архітектуру, що включає клієнтський фронтенд (Next.js), серверний рівень (FastAPI), підсистему асинхронних обчислень (Azure

Functions), централізовані сховища (Azure Storage), інтеграцію з супутниковими платформами (Sentinel-2) та організацію взаємодії через стандартизовані API.

На практиці реалізовано автоматизований процес розгортання всіх компонентів системи за допомогою інструменту Pulumi, що дозволило скоротити час на налаштування середовищ, мінімізувати кількість помилок, забезпечити відтворюваність і контроль версій. Здійснено інтеграцію з системами CI/CD, що спростило тестування та впровадження нових версій сервісу. Особлива увага приділялась питанням ізоляції компонентів, організації масштабування під навантаження та забезпечення безпеки даних.

Завдяки впровадженню serverless-архітектури користувачі системи отримали змогу самостійно запускати агроаналіз: виділяти ділянки поля, задавати параметри обробки, отримувати інтерактивні карти, тематичні NDVI-знімки, карти стрес-зон, а також повні текстові звіти з агрономічними рекомендаціями. Всі етапи обробки виконуються асинхронно, результати надаються через веб-інтерфейс з можливістю завантаження у різних форматах. Система легко масштабується під зростаючі потреби агросектору й залишає простір для подальшого розвитку, зокрема – інтеграції нових джерел даних або додаткових сервісів.

В результаті реалізації проєкту доведено, що поєднання сучасних IaC-підходів та serverless-технологій дозволяє забезпечити високу гнучкість, відмовостійкість, економічність та простоту супроводу складних інформаційних систем у хмарних середовищах. Автоматизація розгортання мінімізувала людський фактор, скоротила час підготовки нових середовищ і підвищила стабільність оновлень. Методика та програмні рішення, апробовані в рамках дипломної роботи, можуть бути масштабовані й адаптовані для інших галузей, де критично важливими є швидкість впровадження, стабільність і контрольованість хмарної інфраструктури.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Datadog. Огляд безсерверної архітектури [Електронний ресурс]. – Режим доступу: https://www.datadoghq.com/knowledge-center/serverless-architecture/. – Дата звернення: 07 січня 2025.
2. RedHat. Що таке безсерверна архітектура? [Електронний ресурс]. – Режим доступу: https://www.redhat.com/en/topics/cloud-native-apps/what-is-serverless. – Дата звернення: 07 січня 2025.
3. Інфраструктура як код [Електронний ресурс] // Вікіпедія. – Режим доступу: https://uk.wikipedia.org/wiki/Інфраструктура_як_код. – Дата звернення: 07 січня 2025.
4. DOU. Infrastructure as Code: базові принципи vs інструменти, що допомагають їх реалізувати [Електронний ресурс]. – Режим доступу: https://dou.ua/lenta/articles/infrastructure-as-code/. – Дата звернення: 07 січня 2025.
5. IBM. What is Serverless Computing? [Електронний ресурс]. – Режим доступу: https://www.ibm.com/cloud/learn/serverless. – Дата звернення: 07 січня 2025.
6. Amazon Web Services (AWS). AWS Lambda [Електронний ресурс]. – Режим доступу: https://aws.amazon.com/lambda/. – Дата звернення: 07 січня 2025.
7. Microsoft. Azure Functions [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/azure/azure-functions/. – Дата звернення: 07 січня 2025.
8. Google Cloud. Google Cloud Functions [Електронний ресурс]. – Режим доступу: https://cloud.google.com/functions. – Дата звернення: 07 січня 2025.

9. HashiCorp. Terraform Documentation [Електронний ресурс]. – Режим доступу: https://developer.hashicorp.com/terraform/docs. – Дата звернення: 07 січня 2025.
10. Amazon Web Services (AWS). AWS CloudFormation [Електронний ресурс]. – Режим доступу: https://aws.amazon.com/cloudformation/. – Дата звернення: 07 січня 2025.
11. Pulumi. Pulumi Documentation [Електронний ресурс]. – Режим доступу: https://www.pulumi.com/docs/. – Дата звернення: 07 січня 2025.
12. RedHat. Ansible Documentation [Електронний ресурс]. – Режим доступу: https://docs.ansible.com/. – Дата звернення: 07 січня 2025.
13. Amazon Web Services (AWS). CI/CD Best Practices with IaC [Електронний ресурс]. – Режим доступу: https://aws.amazon.com/devops/continuous-integration/. – Дата звернення: 07 січня 2025.
14. Checkov. Checkov Documentation [Електронний ресурс]. – Режим доступу: https://www.checkov.io/. – Дата звернення: 07 січня 2025.
15. Aqua Security. Tfsec Documentation [Електронний ресурс]. – Режим доступу: https://aquasecurity.github.io/tfsec/. – Дата звернення: 07 січня 2025.

ДОДАТКИ

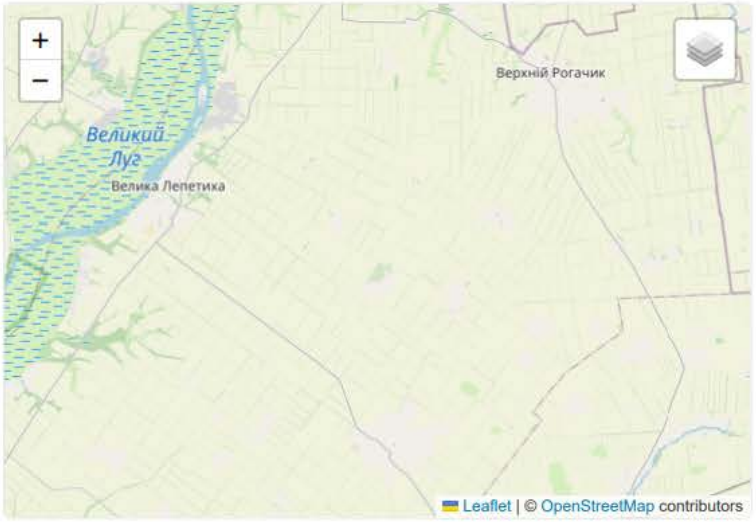
Додаток А

Текстовий звіт результатів аналізу

Звіт по аналізу поля

ID Завдання: 5bd031bd-3395-4982-a269-59a7ff2b3253

Карта області аналізу



Параметри аналізу

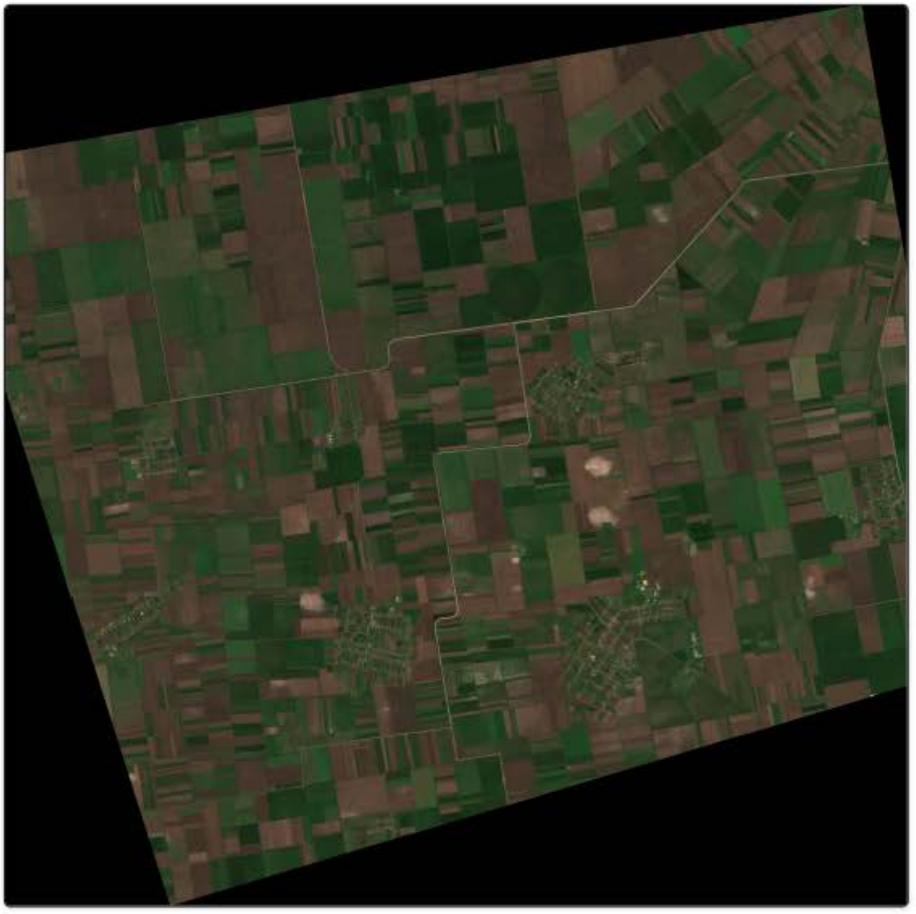
Тип культури:	wheat
NDVI-поріг:	0.3
Хмарність:	≤ 20%
Дата знімку:	02.11.2024 - 02.05.2025

Ключові показники

Площа аналізу	Виявлено стрес-зон
553.44 км²	63.3%

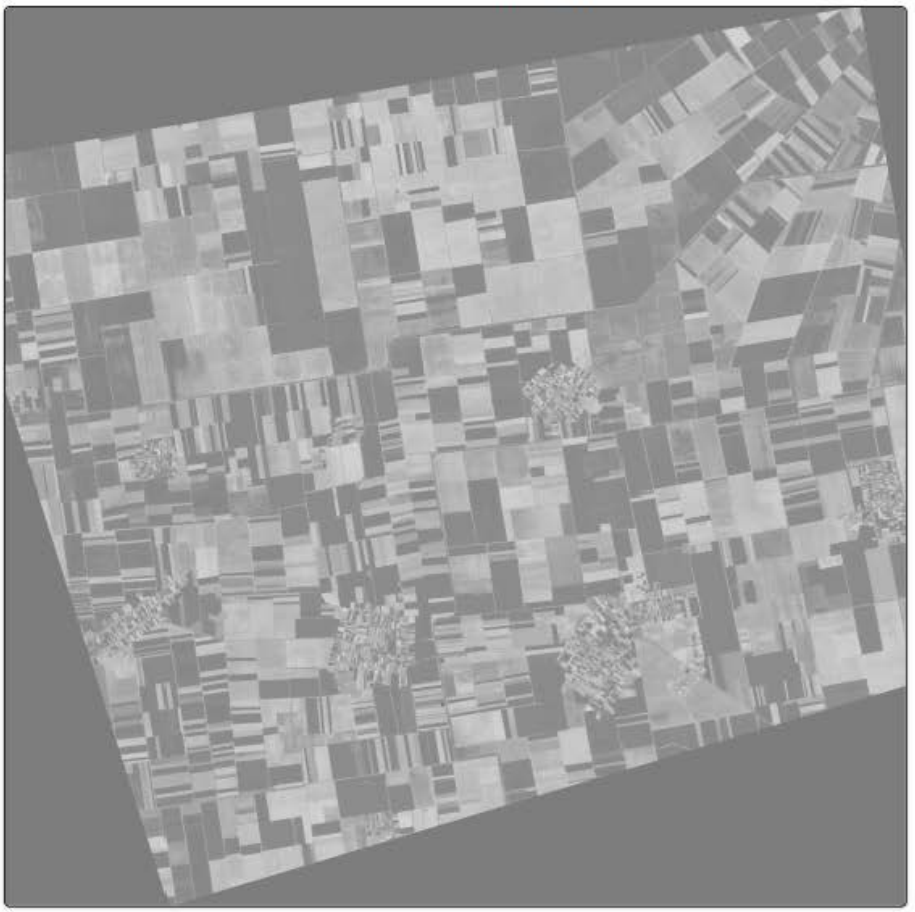
Зображення

Оригінальне зображення (RGB)



Знімок Sentinel-2, візуальний канал (RGB)

NDVI-карта



Карта стрес-зон

(NDVI < 0.3)

Текстовий звіт

Стан поля для культури пшениці можна оцінити як помірний. Середній NDVI показник нижче оптимального рівня, що може свідчити про недостатнє живлення рослин або стресові умови. Потенційний стрес спостерігається на 63.3% площі, що може бути наслідком недостатнього зрошення або захисту від шкідників та хвороб. Рекомендується провести додаткове зрошення, внести добрива та вчасно обробити рослини від шкідників для підвищення врожайності.

Додаток Б

Текст програми

__main__.py

```
import pulumi
import pulumi_azure_native.resources as resources
import pulumi_azure_native.storage as storage
import pulumi_azure_native.web as web
import pulumi_azure_native.authorization as authorization
import pulumi_azure_native.insights as insights
import pulumi_azure_native.operationalinsights as operationalinsights
import os
import hashlib

config = pulumi.Config()
location = config.require("location")
repo_token = config.get_secret("repoToken")
openai_api_key = config.get_secret("openaiApiKey")

project, stack = pulumi.get_project(), pulumi.get_stack()

unique_suffix =
hashlib.sha1(f"{project}-{stack}".encode("utf-8")).hexdigest()[:6]

rg_name = f"rg-{project[:8]}-{stack[:8]}-{unique_suffix}"
sa_name = f"sa{project[:4]}{stack[:4]}{unique_suffix}"

rg = resources.ResourceGroup(
    "rg",
    resource_group_name=rg_name,
    location=location,
)

sa = storage.StorageAccount(
    "sa",
```

```
    account_name=sa_name,
    resource_group_name=rg.name,
    location=rg.location,
    sku=storage.SkuArgs(name=storage.SkuName.STANDARD_LRS),
    kind=storage.Kind.STORAGE_V2,
    allow_blob_public_access=True,
)

images_container = storage.BlobContainer(
    "images",
    account_name=sa.name,
    resource_group_name=rg.name,
    container_name="images",
    public_access=storage.PublicAccess.BLOB,
    opts=pulumi.ResourceOptions(depends_on=[sa]),
)
reports_container = storage.BlobContainer("reports",
account_name=sa.name, resource_group_name=rg.name,
container_name="reports")
queue = storage.Queue("analysis-requests", account_name=sa.name,
resource_group_name=rg.name, queue_name="analysis-requests")

sa_keys =
storage.list_storage_account_keys_output(resource_group_name=rg.name,
account_name=sa.name)
connection_string = pulumi.Output.format(

"DefaultEndpointsProtocol=https;AccountName={0};AccountKey={1};EndpointSu
ffix=core.windows.net",
    sa.name,
    sa_keys.keys[0].value,
)

func_plan = web.AppServicePlan(
```

```
    "func-plan",
    resource_group_name=rg.name,
    location=rg.location,
    kind="functionapp",
    sku=web.SkuDescriptionArgs(name="Y1", tier="Dynamic"),
    reserved=True,
)

# Create Log Analytics workspace
log_analytics_workspace = operationalinsights.Workspace(
    "log-analytics",
    resource_group_name=rg.name,
    location=rg.location,
    sku=operationalinsights.WorkspaceSkuArgs(
        name="PerGB2018"
    ),
    retention_in_days=30,
)

# Create Application Insights
app_insights = insights.Component(
    "app-insights",
    resource_group_name=rg.name,
    location=rg.location,
    kind="web",
    application_type="web",
    workspace_resource_id=log_analytics_workspace.id,
)

app_settings = [
                    web.NameValuePairArgs(name="FUNCTIONS_WORKER_RUNTIME",
value="python"),
                web.NameValuePairArgs(name="FUNCTIONS_EXTENSION_VERSION",
value="~4"),
```

```
                        web.NameValuePairArgs(name="AzureWebJobsStorage",
value=connection_string),
    web.NameValuePairArgs(name="ANALYSIS_QUEUE_NAME", value=queue.name),
    web.NameValuePairArgs(name="IMAGES_CONTAINER_NAME", value="images"),
                    web.NameValuePairArgs(name="REPORTS_CONTAINER_NAME",
value="reports"),
            web.NameValuePairArgs(name="AZURE_STORAGE_CONNECTION_STRING",
value=connection_string),
             web.NameValuePairArgs(name="APPINSIGHTS_INSTRUMENTATIONKEY",
value=app_insights.instrumentation_key),
      web.NameValuePairArgs(name="APPLICATIONINSIGHTS_CONNECTION_STRING",
value=app_insights.connection_string),
    web.NameValuePairArgs(name="OPENAI_API_KEY", value=openai_api_key),
]

func_app = web.WebApp(
    "func-api",
    resource_group_name=rg.name,
    location=rg.location,
    server_farm_id=func_plan.id,
    kind="functionapp",
    site_config=web.SiteConfigArgs(
        app_settings=app_settings,
        linux_fx_version="Python|3.11",
        cors=web.CorsSettingsArgs(
            allowed_origins=["*"],
        ),
    ),
    identity=web.ManagedServiceIdentityArgs(type="SystemAssigned"),
)

current_config = authorization.get_client_config()
```

```
storage_queue_data_contributor_role_id                                    =
f"/subscriptions/{current_config.subscription_id}/providers/Microsoft.Aut
horization/roleDefinitions/974c5e8b-45b9-4653-ba55-5f855dd0fb88"

queue_role_assignment = authorization.RoleAssignment(
    "funcQueueRoleAssignment",
    principal_id=func_app.identity.principal_id,
    principal_type=authorization.PrincipalType.SERVICE_PRINCIPAL,
    role_definition_id=storage_queue_data_contributor_role_id,
    scope=sa.id,
)

# Assign Storage Blob Data Contributor role to the function app identity
storage_blob_data_contributor_role_id                                     =
f"/subscriptions/{current_config.subscription_id}/providers/Microsoft.Aut
horization/roleDefinitions/ba92f5b4-2d11-453d-a403-e96b0029c9fe"

blob_role_assignment = authorization.RoleAssignment(
    "funcBlobRoleAssignment", # New resource name
    principal_id=func_app.identity.principal_id,
    principal_type=authorization.PrincipalType.SERVICE_PRINCIPAL,
     role_definition_id=storage_blob_data_contributor_role_id, # Use the
blob role ID
    scope=sa.id, # Scope to the storage account
)

backend_api_url     =     func_app.default_host_name.apply(lambda     h:
f"https://{h}")

front_plan = web.AppServicePlan(
    "front-plan",
    resource_group_name=rg.name,
    location=rg.location,
    kind="app",
```

```
    sku=web.SkuDescriptionArgs(name="B1", tier="Basic"),
    reserved=True,
)

front_app = web.WebApp(
    "frontend",
    resource_group_name=rg.name,
    location=rg.location,
    server_farm_id=front_plan.id,
    kind="app",
    site_config=web.SiteConfigArgs(
        linux_fx_version="NODE|18-lts",
        app_command_line="node server.js",
        app_settings=[
                        web.NameValuePairArgs(name="NEXT_PUBLIC_API_URL",
value=backend_api_url),
                    web.NameValuePairArgs(name="WEBSITE_RUN_FROM_PACKAGE",
value="1"),
        ]
    ),
    identity=web.ManagedServiceIdentityArgs(type="SystemAssigned"),
)

pulumi.export("function_app_endpoint", backend_api_url)
pulumi.export("frontend_endpoint",
front_app.default_host_name.apply(lambda h: f"https://{h}"))
pulumi.export("storage_account_name", sa.name)
pulumi.export("analysis_queue_name", queue.name)
pulumi.export("images_container_name", images_container.name)
pulumi.export("reports_container_name", reports_container.name)

pulumi.export("resource_group_name", rg.name)
pulumi.export("function_app_name", func_app.name)
pulumi.export("frontend_app_name", front_app.name)
```

```
pulumi.export("function_app_principal_id",
func_app.identity.principal_id)
pulumi.export("app_insights_name", app_insights.name)
pulumi.export("app_insights_instrumentation_key",
app_insights.instrumentation_key)
pulumi.export("log_analytics_workspace_name",
log_analytics_workspace.name)
pulumi.export("log_analytics_workspace_id", log_analytics_workspace.id)
```

function_app.py

```
import logging
import os
import azure.functions as func
from fastapi import FastAPI, Request
from fastapi.responses import JSONResponse
from fastapi.middleware.cors import CORSMiddleware
from contextlib import asynccontextmanager
from azure.storage.blob import BlobServiceClient
from azure.storage.queue import QueueServiceClient

from routers.analysis import router as analysis_router
from routers.report import router as report_router
from routers.progress import router as progress_router
from shared_code.queue_handler import process_analysis
from shared_code.helpers.blob import get_sync_blob_service_client

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

@asynccontextmanager
async def lifespan(app: FastAPI):
    conn_str = os.getenv("AZURE_STORAGE_CONNECTION_STRING")
    if not conn_str:
        raise RuntimeError("AZURE_STORAGE_CONNECTION_STRING must be set")
    blob_client = BlobServiceClient.from_connection_string(conn_str)
    queue_client = QueueServiceClient.from_connection_string(conn_str)
    app.state.blob_service_client = blob_client
    app.state.queue_service_client = queue_client
    logger.info("Lifespan startup: Sync clients initialized.")
    try:
        yield
    finally:
        logger.info("Lifespan shutdown.")
```

```
fastapi_app = FastAPI(
    title="Agro Monitor Backend API",
      description="API for retrieving and analyzing agricultural field
data",
    version="1.0.0",
    lifespan=lifespan,
)

# Add CORS middleware
origins = [
       "https://frontendc613cf2f.azurewebsites.net",    # Your deployed
frontend
    "http://localhost:3000",                          # Local development
    # Add any other origins if needed
]

fastapi_app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

@fastapi_app.exception_handler(Exception)
async def global_exception_handler(request: Request, exc: Exception):
    logger.error(f"Unhandled exception: {str(exc)}", exc_info=True)
    return JSONResponse(
        status_code=500,
        content={"detail": "Internal server error", "error": str(exc)}
    )

fastapi_app.include_router(analysis_router, prefix="/api/analyze")
fastapi_app.include_router(report_router, prefix="/api/report")
fastapi_app.include_router(progress_router, prefix="/api/progress")

@fastapi_app.get("/", include_in_schema=False)
async def read_root():
    return {"message": "Welcome to AgroMonitor API"}

app                =                func.AsgiFunctionApp(app=fastapi_app,
http_auth_level=func.AuthLevel.ANONYMOUS)

@app.function_name(name="process_analysis_job")
@app.queue_trigger(
    arg_name="msg",
    queue_name="%ANALYSIS_QUEUE_NAME%",
    connection="AZURE_STORAGE_CONNECTION_STRING",
)
def process_analysis_job(msg: func.QueueMessage):
    logger.info(f"Processing message: {msg.id}")
```

```
    try:
        content = msg.get_body().decode('utf-8')
        logger.info(f"Message content: {content}")

        process_analysis(msg)

    except Exception as e:
            logger.error(f"Error processing message {msg.id}: {str(e)}",
exc_info=True)
        raise
```