НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут телекомунікаційних систем

Кафедра телекомунікацій

До захисту допущено:

В.о.завідувача кафедри

_____ Сергій КРАВЧУК

«___»_____2026 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія та програмування інфокомунікацій»

спеціальності 172 «Телекомунікації та радіотехніка»

на тему: «Використвння Google Cloud Platform для автоматизації процесів у DevOps»

Виконав:
студент IV курсу, групи ТЗ-22
Омельчук Денис Андрійович _____

Керівник:
старший викладач кафедри ТК НН ІТС, к.ф.-м.н.
Руренко Олександр Григорович _____

Рецензент:
Доцент кафедри ІТТ НН ІТС, к.т.н., доцент
Правило Валерій Володимирович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2026 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут телекомунікаційних систем
Кафедра телекомунікацій

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 172 «Телекомунікації та радіотехніка»

Освітньо-професійна програма «Інженерія та програмування інфокомунікацій»

ЗАТВЕРДЖУЮ
В.о.завідувача кафедри
_____ Сергій КРАВЧУК
«___»_____2026 р.

ЗАВДАННЯ
на дипломну роботу студенту
Омельчуку Денису Андрійовичу

1. Тема роботи «Використвння Google Cloud Platform для автоматизації процесів у DevOps», керівник роботи Руренко Олександр Григорович, Кандидат наук, Старший викладач, затверджені наказом по університету від «26» травня 2026 р. № 1912-с.

2. Термін подання студентом роботи 10 червня 2026 р.

3. Вихідні дані до роботи: Хмарна платформа Google Cloud Platform (GCP); система управління контейнерами Google Kubernetes Engine (GKE); інструмент Infrastructure as Code — Terraform; система CI/CD — GitHub Actions; база даних Cloud SQL (PostgreSQL 15); сховище секретів Secret Manager; реєстр артефактів Artifact Registry; механізм безключової автентифікації Workload Identity Federation (WIF); методологія Zero Trust; метрики DORA для оцінки ефективності DevOps-практик.

4. Зміст роботи: Провести аналіз сучасних DevOps-практик, парадигм автоматизації (CI/CD, IaC, GitOps) та сервісів Google Cloud Platform. Спроектувати архітектуру системи: мережеву топологію, модель ідентифікації та безпеки Zero Trust, структуру Terraform-модулів і CI/CD-конвеєра. Реалізувати інфраструктуру за допомогою Terraform (п'ять модулів: vpc, iam, db, gke, registry), контейнеризувати Flask-застосунок, розробити Kubernetes-

маніфести та налаштувати GitHub Actions workflow з Workload Identity Federation. Оцінити ефективність реалізованих рішень за метриками DORA, провести аналіз вартості та аудит безпеки за CIS Benchmark for GKE.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): презентація для захисту дипломної роботи (9 слайдів): слайд 1 — титульний; слайд 2 — «Актуальність та мета дослідження»; слайд 3 — «Хмарна архітектура IaC (Технологічний стек)»; слайд 4 — «Архітектура системи»; слайд 5 — «Безпека: Zero Trust та Workload Identity Federation»; слайд 6 — «Реалізація Terraform-модулів для CI/CD»; слайд 7 — «Результати: метрики DORA»; слайд 8 — «Оцінка критеріїв ціна — ефективність»; слайд 9 — «Напрямки розвитку»; архітектурна діаграма системи автоматизації хмарної інфраструктури (рис. 2.1).

6. Дата видачі завдання «01» лютого 2026 р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання, ознайомлення з темою та постановка задачі	01.02.2026	Виконано
2	Аналіз літератури, огляд DevOps-практик та сервісів GCP (Розділ 1)	01.02–28.02.2026	Виконано
3	Проектування архітектури, мережевої топології та CI/CD-конвеєра (Розділ 2)	01.03–20.03.2026	Виконано
4	Реалізація Terraform-модулів (vpc, iam, db, gke, registry)	21.03–07.04.2026	Виконано
5	Контейнеризація застосунку, розробка K8s-маніфестів, налаштування GitHub Actions (Розділ 3)	08.04–25.04.2026	Виконано
6	Оцінка ефективності за метриками DORA, аналіз вартості та безпеки (Розділ 4)	26.04–10.05.2026	Виконано
7	Оформлення пояснювальної записки згідно вимог ДСТУ 3008:2015	11.05–25.05.2026	Виконано
8	Підготовка демонстраційного матеріалу (презентація для захисту)	26.05–05.06.2026	Виконано
9	Подання роботи на кафедру та захист дипломної роботи	10.06.2026	Виконано

Студент Денис ОМЕЛЬЧУК

Керівник Олександр РУРЕНКО

РЕФЕРАТ

Дипломна робота містить: 59 сторінок, 1 рисунок, 9 таблиць, 27 джерел, 5 додатків.

Об'єкт дослідження — процес розгортання та управління хмарною інфраструктурою на базі Google Kubernetes Engine.

Мета роботи — розробка автоматизованої системи розгортання хмарної інфраструктури з використанням Terraform, GitHub Actions та Google Kubernetes Engine для забезпечення принципів DevOps та Zero Trust безпеки.

Методи дослідження — інфраструктура як код (IaC), CI/CD автоматизація, федерація ідентифікації робочих навантажень (WIF), аналіз за метриками DORA.

У роботі розроблено п'ять Terraform-модулів для автоматизованого керування мережею VPC, IAM, базою даних Cloud SQL, кластером GKE та Artifact Registry. Реалізовано GitHub Actions CI/CD пайплайн із keyless-автентифікацією через Workload Identity Federation. Розгорнуто систему спостережуваності на базі kube-prometheus-stack. Проведено кількісну оцінку ефективності системи за чотирма метриками DORA.

Результати роботи можуть бути використані як еталонна реалізація DevOps-платформи на базі Google Cloud Platform для навчальних і реальних проектів.

Ключові слова: KUBERNETES, TERRAFORM, DEVOPS, GKE, GITHUB ACTIONS, WORKLOAD IDENTITY, CI/CD, IAC, ZERO TRUST, POSTGRESQL, МОНІТОРИНГ, АВТОМАТИЗАЦІЯ.

ABSTRACT

The thesis contains: 59 pages, 1 figure, 9 tables, 27 sources, 5 appendices.

Object of research — the process of deploying and managing cloud infrastructure based on Google Kubernetes Engine.

Purpose of the work — development of an automated cloud infrastructure deployment system using Terraform, GitHub Actions, and Google Kubernetes Engine to ensure DevOps principles and Zero Trust security.

Research methods — Infrastructure as Code (IaC), CI/CD automation, Workload Identity Federation (WIF), DORA metrics analysis.

The thesis develops five Terraform modules for automated management of VPC network, IAM, Cloud SQL database, GKE cluster, and Artifact Registry. A GitHub Actions CI/CD pipeline with keyless authentication via Workload Identity Federation is implemented. An observability system based on kube-prometheus-stack is deployed. System effectiveness is evaluated quantitatively using four DORA metrics.

The results can be used as a reference DevOps platform implementation on Google Cloud Platform for educational and real-world projects.

Keywords: KUBERNETES, TERRAFORM, DEVOPS, GKE, GITHUB ACTIONS, WORKLOAD IDENTITY, CI/CD, IAC, ZERO TRUST, POSTGRESQL, MONITORING, AUTOMATION.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API	Application Programming Interface — інтерфейс програмування застосунків
CD	Continuous Delivery / Continuous Deployment — безперервне постачання / розгортання
CI	Continuous Integration — безперервна інтеграція
CIDR	Classless Inter-Domain Routing — безкласова міждоменна маршрутизація
CSI	Container Storage Interface — інтерфейс сховища для контейнерів
CVE	Common Vulnerabilities and Exposures — загальний реєстр вразливостей
DAG	Directed Acyclic Graph — направлений ациклічний граф
DNS	Domain Name System — система доменних імен
DORA	DevOps Research and Assessment — дослідницька та оціночна організація DevOps
GCP	Google Cloud Platform — хмарна платформа Google
GKE	Google Kubernetes Engine — керований сервіс Kubernetes від Google
GSA	Google Service Account — сервісний акаунт Google
HTTP	HyperText Transfer Protocol — протокол передачі гіпертексту
HTTPS	HyperText Transfer Protocol Secure — захищений протокол передачі гіпертексту
IaC	Infrastructure as Code — інфраструктура як код
IAM	Identity and Access Management — управління ідентифікацією та доступом
IP	Internet Protocol — інтернет-протокол
JSON	JavaScript Object Notation — текстовий формат обміну даними
KSA	Kubernetes Service Account — сервісний акаунт Kubernetes

NAT	Network Address Translation — трансляція мережевих адрес
NEG	Network Endpoint Group — група мережевих точок доступу
OCI	Open Container Initiative — ініціатива відкритих контейнерів
OIDC	OpenID Connect — протокол автентифікації на базі OAuth 2.0
RBAC	Role-Based Access Control — контроль доступу на основі ролей
SQL	Structured Query Language — мова структурованих запитів
TCP	Transmission Control Protocol — протокол керування передачею
TLS	Transport Layer Security — протокол безпеки транспортного рівня
URL	Uniform Resource Locator — уніфікований локатор ресурсів
VPC	Virtual Private Cloud — віртуальна приватна хмара
WIF	Workload Identity Federation — федерація ідентифікації робочих навантажень

ВСТУП

Сучасна розробка програмного забезпечення нерозривно пов'язана з хмарними технологіями. Зростання складності систем, вимоги до безперервної доступності та необхідність скорочення часу виходу продукту на ринок спонукають команди переходити від ручного управління інфраструктурою до повністю автоматизованих конвеєрів. Водночас питання безпеки стають дедалі критичнішими: витоки ключів сервісних акаунтів, надмірні привілеї та ручні зміни в production залишаються найчастішими векторами атак на хмарні системи.

У цьому контексті актуальності набуває підхід, що поєднує Infrastructure as Code, безключову автентифікацію через Workload Identity Federation та принципи Zero Trust у межах єдиної автоматизованої платформи на Google Kubernetes Engine. Дослідження такої платформи дозволяє не лише вирішити практичні задачі конкретного проекту, а й сформувати відтворювані архітектурні рішення, придатні для масштабування на виробничі середовища.

Мета роботи — спроектувати та реалізувати референсну Cloud-Native платформу на Google Cloud Platform із повністю автоматизованим CI/CD-конвеєром, безключовою автентифікацією та безпечною доставкою секретів, оцінити її ефективність за метриками DORA та витратами.

Для досягнення мети поставлено такі завдання:

1. Провести аналіз сучасних DevOps-практик, парадигм автоматизації та сервісів Google Cloud Platform, що є підґрунтям для проектування системи.

2. Спроектувати архітектуру системи: мережеву топологію, модель ідентифікації та безпеки, структуру Terraform-модулів і CI/CD-конвеєра.

3. Реалізувати інфраструктуру за допомогою Terraform (п'ять модулів), контейнеризувати застосунок, розробити Kubernetes-маніфести та налаштувати GitHub Actions workflow.

4. Оцінити ефективність реалізованих рішень за метриками DORA, провести аналіз вартості та аудит безпеки.

Об'єкт дослідження — процес автоматизації розгортання та управління хмарною інфраструктурою в Google Cloud Platform.

Предмет дослідження — методи та інструменти реалізації безпечного CI/CD-конвеєра з Workload Identity Federation, Infrastructure as Code на основі Terraform та оркестрації контейнерів у Google Kubernetes Engine.

Методи дослідження: системний аналіз архітектурних рішень, порівняльний аналіз альтернативних інструментів, практична реалізація та вимірювання показників ефективності методом контрольованого експерименту.

Практичне значення результатів. Реалізований проект gke-terraform-project є відтворюваним шаблоном Cloud-Native платформи з відкритим кодом. Архітектурні рішення — безключова автентифікація, приватний кластер, доставка секретів через CSI Driver — безпосередньо застосовні у виробничих середовищах малих і середніх команд.

Наукова новизна отриманих результатів полягає у розробці цілісного відтворюваного рішення для автоматизованого розгортання хмарної інфраструктури на базі GKE, що реалізує принципи Zero Trust через Workload Identity Federation і виключає використання довгострокових сервісних ключів як у CI/CD-пайплайні, так і в Kubernetes-подах.

Структура роботи. Робота складається зі вступу, чотирьох розділів, висновків та списку використаних джерел (27 позицій). Загальний обсяг — 80 сторінок, 9 таблиць, 28 лістингів.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ DEVOPS ТА ОГЛЯД СЕРВІСІВ GOOGLE CLOUD PLATFORM

1.1 Що таке DevOps і навіщо він потрібен

Ще наприкінці 2000-х більшість IT-команд жила за принципом «розробка — окремо, операції — окремо»: розробники здавали код у реліз раз на кілька місяців, а адміністратори отримували бінарник без чіткого розуміння того, що змінилося і чого чекати. Наслідки були передбачувані — затримки, взаємні претензії і багатогодинні інциденти у production. Поворотним моментом стала конференція DevOpsDays у Генті 2009 року, де Patrick Debois і Andrew Shafer запропонували подивитися на проблему з іншого кута: не ділити команду, а об'єднати людей навколо спільних метрик і спільної відповідальності за результат [12].

На практиці DevOps проявляється на двох рівнях. Організаційний — це злам функціональних сило: ті, хто пишуть код, і ті, хто підтримують його в production, мають спільні цілі та спільно несуть відповідальність за надійність. Технічний — це автоматизація всього, що досі робилося вручну: збірка, тестування, розгортання, моніторинг і навіть підняття самої інфраструктури. Другий рівень без першого не дає результату — автоматизація лише прискорює хаос, якщо процес нею не керується.

Основу DevOps описують дві моделі. Перша — CALMS [12]: Culture, Automation, Lean, Measurement, Sharing — п'ять осей, за якими оцінюють зрілість команди. Друга — «Три Шляхи» (The Three Ways) [1]: Flow — зменшення часу від ідеї до production, Feedback — швидке виявлення та усунення проблем, Continual Learning — культура експериментів і навчання на відмовах.

Ефективність DevOps вимірюють чотирма метриками DORA [2; 13]: Deployment Frequency, Lead Time for Changes, Mean Time to Recovery та Change Failure Rate. Цими ж метриками буде оцінюватись результат практичної частини роботи у Розділі 4.

Із DevOps виросли похідні напрямки. DevSecOps вбудовує безпеку у конвеєр за принципом shift-left — у проекті це етап lint-and-scan з Trivy, TFLint та Hadolint. GitOps використовує Git як єдине джерело істини про стан

системи. MLOps застосовує DevOps до моделей машинного навчання. Platform Engineering будує внутрішні платформи для розробників.

1.2 Ключові парадигми автоматизації

Усі чотири парадигми автоматизації, що описані нижче, застосовано у проекті gke-terraform-project.

1.2.1 CI, CD та Continuous Deployment

Практика Continuous Integration будується на одному ключовому правилі: кожна зміна в коді інтегрується у спільну гілку якомога частіше — в ідеалі щодня або навіть кілька разів на день [4; 8]. При кожному злитті автоматично запускаються збірка та тести. Це радикально скорочує час між появою дефекту та його виявленням: замість того щоб дізнатися про конфлікт через тижні, команда отримує зворотний зв'язок за хвилини після push.

Continuous Delivery розвиває CI: кожен успішно зібраний і протестований артефакт готовий до виходу у production [3]. Рішення про реліз залишається за людиною — достатньо натиснути кнопку. Continuous Deployment іде ще далі і прибирає навіть цей крок: якщо автоматичні перевірки пройшли — зміна потрапляє до користувачів автоматично. У цьому проекті реалізовано саме Continuous Deployment: кожен push у гілку main ініціює workflow Deploy to GKE, який без жодного ручного підтвердження виконує terraform apply, збирає Docker-образ і розгортає оновлення командою kubectl rollout. Порівняння підходів наведено у Таблиці 1.1.

Таблиця 1.1

CI vs CD vs Continuous Deployment

Критерій	Continuous Integration	Continuous Delivery	Continuous Deployment
Тригер	Кожен коміт	Успішне CI	Успішне CD
Результат	Перевірений білд	Реліз-кандидат	Зміни у production
Ручний крок	Немає	Натиснути «Release»	Немає
Зворотний зв'язок	Хвилини	Години	Хвилини – години
Типовий ризик	Поломка білду	Затримка релізу	Аварія в production

1.2.2 Хмарні технології на основі Infrastructure as Code (IaC)

Традиційне управління інфраструктурою через веб-консолі або ручні скрипти породжує дві проблеми: стан середовища нікому невідомий точно, і

відтворити його на новому майданчику майже неможливо. Infrastructure as Code (IaC) вирішує обидві: весь стек — мережі, кластери, бази даних, сервісні акаунти — описується у текстових файлах, які зберігаються у Git і виконуються автоматизованим інструментом [7]. У проекті роль такого інструменту виконує Terraform [22]: п'ять модулів (vpc, iam, db, gke, registry) та конфігурація environments/dev повністю описують хмарний стек GCP, а команда terraform apply будує його з нуля незалежно від поточного стану.

IaC буває двох видів: імперативна — описує послідовність команд (bash, Ansible), та декларативна — описує кінцевий стан системи, а інструмент сам визначає, що потрібно змінити (Terraform, Pulumi, CloudFormation). Декларативний стиль ідемпотентний: повторний запуск над уже готовою системою нічого не зламає. Це усуває «дрейф конфігурації» між реальним станом інфраструктури та її описом у коді [22].

1.2.3 Технологія GitOps

GitOps, сформульований Alexis Richardson у 2017 році, переносить в інфраструктуру підхід, добре знайомий розробникам: єдиним джерелом істини про стан production є Git [11]. Будь-яка зміна — нова версія сервісу, виправлення конфігурації, оновлення сертифіката — проходить через pull request. Замість того щоб CI-агент штовхав зміни назовні, спеціальний оператор всередині кластера (Argo CD або Flux) сам тягне бажаний стан з репозиторію та виявляє розбіжності. Це звужує поверхню атаки: CI-сервер не має прямого доступу до production і не потребує довгострокових ключів. У даному проекті застосовано класичний push-підхід через kubectl apply; GitOps з Argo CD розглядається як пріоритетний напрямок розвитку.

1.2.4 Сталі інфраструктури (Immutable Infrastructure)

Ідея Immutable Infrastructure проста: запущений екземпляр ніколи не змінюється після деплою — він замінюється новим [10]. Це позбавляє від класичної проблеми «сніжного» сервера, де ніхто вже не пам'ятає, чому конфігурація саме така і що було доналаштовано вручну місяць тому. Контейнери є природним носієм цієї ідеї: Docker-образ збирається один раз, тестується та запускається у будь-якому середовищі без правок. У проекті цей принцип закріплено конфігурацією Kubernetes: securityContext встановлює readOnlyRootFilesystem: true, а тимчасові файли Gunicorn записуються в окремий emptyDir-том.

1.3 Огляд хмарних архітектур: загальний підхід

Хмарна нативність — це не про те, де запускається застосунок, а про те, як він спроектований. CNCF об'єднує під Cloud-Native п'ять характеристик: контейнеризацію, мікросервіси, сервісні мережі, незмінну інфраструктуру та декларативні API [13]. Спільне у всіх п'яти — розрахунок на горизонтальне масштабування, відновлення без людського втручання та незалежне оновлення компонентів. Застосунок цього проекту дотримується цих принципів: він упакований у контейнер, конфігурується через змінні середовища, масштабується репліками та замінюється через rolling update без downtime.

Мікросервісна архітектура пропонує ділити систему на невеликі автономні компоненти, кожен з яких має власну область відповідальності, власну базу даних і власний цикл релізу [6]. Перевага — слабка зв'язаність: відмова одного сервісу не каскадує на всю систему, кожен компонент масштабується незалежно. Разом з тим зростає операційна складність: потрібні розподілене трасування, узгодженість даних між сервісами та зрілі практики розгортання. Застосунок цього проекту — монолітний Flask-сервіс; така простота свідома і дозволяє сфокусуватися на інфраструктурних патернах, а не на міжсервісній комунікації.

Twelve-Factor App [9] формулює 12 правил для хмарних застосунків: один кодовий каталог, явні залежності, конфіг через environment variables, зовнішні сервіси як підключені ресурси, розділення стадій build/release/run, stateless-процеси, прив'язка до порту, горизонтальне масштабування, швидкий старт і коректне завершення, однаковість dev/staging/prod, логи як потік подій, адмін-скрипти як одноразові процеси. У застосунку проекту (app/app.py + Gunicorn у Dockerfile) дотримано усіх дванадцяти факторів.

Серед типових патернів Cloud-Native виділяють sidecar — допоміжний контейнер у тому самому Pod-і поряд з основним, ambassador — окремий проксі для взаємодії з зовнішніми сервісами, та service mesh — уніфікований шар управління трафіком і безпекою на основі проксі Envoy [6]. У проекті використано sidecar-патерн: поряд із основним контейнером app у тому самому Pod-і запускається cloud-sql-proxy, який створює зашифрований тунель до Cloud SQL.

1.4 Чому саме Google Cloud Platform

Ринок публічної хмари тримають три гравці: Amazon Web Services, Microsoft Azure і Google Cloud Platform [14]. Усі троє мають повний набір DevOps-сервісів, але кожен зі своєю специфікою (Таблиця 1.2).

Таблиця 1.2

Порівняння AWS, Azure та GCP за DevOps-критеріями

Критерій	AWS	Microsoft Azure	Google Cloud Platform
Запуск	2006	2010	2008
Managed Kubernetes	Amazon EKS	AKS	GKE
Частка ринку IaaS (2024)	≈ 31 %	≈ 25 %	≈ 11 %
Реєстр контейнерів	Amazon ECR	Azure Container Registry	Artifact Registry
Сховище секретів	Secrets Manager	Key Vault	Secret Manager
CI/CD	CodePipeline + CodeBuild	Azure Pipelines	Cloud Build + Cloud Deploy
Federated identity	OIDC + IAM Roles	Workload Identity	Workload Identity Federation
Спостережуваність	CloudWatch	Azure Monitor	Cloud Operations Suite

AWS — найстарший, найбільший за асортиментом сервісів і часткою ринку, але має заплутане ціноутворення і фрагментовану IAM-модель. Azure домінує у enterprise завдяки інтеграції з Active Directory та Microsoft 365; після купівлі GitHub Microsoft робить акцент на GitHub Actions як універсальний CI/CD.

GCP виросла з внутрішньої інфраструктури Google — Borg, Spanner, Colossus. Звідси її переваги: найкращий у галузі керований Kubernetes, адже Google є його автором, глобальна приватна мережа з низькими затримками та прозора IAM-ієрархія Organization → Folder → Project → Resource [16; 21].

Для цієї роботи обрано GCP за п'ятьма причинами: лідерство у Kubernetes-екосистемі; зрілість Workload Identity Federation для CI/CD без ключів; інтеграція Cloud Operations з Prometheus та OpenTelemetry; зрозуміла IAM-модель; щедрий Free Tier для академічних проектів.

1.5 Сервіси GCP, використані у проекті

У Таблиці 1.3 узагальнено сервіси GCP за етапами DevOps-конвеєра [16]; нижче детально розглянуто ті з них, що застосовано у gke-terraform-project.

Таблиця 1.3

Сервіси GCP за етапами DevOps-конвеєра

Етап	Сервіси GCP	Що роблять
Plan / Code	Cloud Source Repositories, Cloud Workstations	Git-репозиторії, готові середовища розробки
Build	Cloud Build, Artifact Registry	Збірка та зберігання образів
Test	Cloud Build, Vulnerability Scanning	Запуск тестів і пошук CVE в образах
Release	Cloud Deploy, Binary Authorization	Canary/blue-green релізи, допуск підписаних образів
Deploy	GKE, Cloud Run, Compute Engine MIG	Запуск контейнерів
Operate	GKE Autoscaler, Pod Disruption Budgets	Автомасштабування, self-healing
Monitor	Cloud Logging, Cloud Monitoring, Cloud Trace	Логи, метрики, трасування
Secure	Secret Manager, Cloud KMS, Cloud Armor, IAM	Секрети, ключі, WAF, доступ

1.5.1 Google Kubernetes Engine

GKE — керований Kubernetes [21]. У режимі Standard користувач сам обирає типи нод, мережу та add-on-и; у Autopilot ноди керовані автоматично, оплата йде за ресурси Pod-ів. У проекті використано Standard з двома node pool: system-pool на стабільних on-demand нодах для системних pod’ів та workload-pool на Spot-нодах e2-standard-2 зі знижкою до 91 % для основного навантаження. Кластер створено приватним, Workload Identity та Secret Manager add-on увімкнені.

1.5.2 Artifact Registry

На зміну застарілому Container Registry у 2021 році прийшов Artifact Registry [19]. Він підтримує не лише Docker- і OCI-образи, а й Maven, npm, Python-пакети та Helm-чарти з єдиною точкою управління доступом через IAM. Вбудований vulnerability-сканер перевіряє образи на відомі CVE при кожному push. У проекті створено Docker-репозиторій europe-west1-

docker.pkg.dev/gke-petproject-2026/my-repo; CI-конвеєр тегує образ скороченим git-хешем і штовхає туди після успішного проходження lint-and-scan.

1.5.3 Cloud Build та Cloud Deploy

Cloud Build — managed CI: build steps описуються у cloudbuild.yaml, виконуються в ізольованих контейнерах, тригериться по push з GitHub/GitLab [16]. Cloud Deploy — managed CD з canary та blue-green релізами для GKE та Cloud Run. У проекті CI/CD виконує GitHub Actions; порівняння з Cloud Build і Cloud Deploy буде в Розділі 4.

1.5.4 Secret Manager

Зберігати паролі та ключі у коді або у незашифрованих змінних середовища — типова точка компрометації. Secret Manager вирішує цю проблему: усі чутливі значення зберігаються з версіонуванням, шифруванням через Cloud KMS та аудитом кожного звернення [20]. Для GKE існує окремий Secret Manager CSI Driver, що монтує секрет безпосередньо у файлову систему Pod-а без проміжного Kubernetes Secret. У цьому проекті пароль до Cloud SQL генерується модулем Terraform та записується у Secret Manager; SecretProviderClass у k8s/dependencies.yaml монтує його у /secrets/db-password при старті Pod-а, а Flask-застосунок зчитує файл при ініціалізації з'єднання.

1.5.5 Cloud Operations Suite

Google Cloud Operations Suite охоплює весь стек спостережуваності: Cloud Logging збирає структуровані логи, Cloud Monitoring — метрики та алерти, Cloud Trace — дані розподіленого трасування, Cloud Profiler — профілі CPU та пам'яті [25]. У GKE спостережуваність вмикається декількома рядками у Terraform: параметри logging_config і monitoring_config активують збір логів та метрик для системних компонентів і робочих навантажень, після чого stdout та stderr усіх контейнерів автоматично індексуються у Cloud Logging. Для кастомних метрик застосунку у проекті розгорнуто kube-prometheus-stack через Helm-скрипт scripts/install-monitoring.sh.

1.5.6 Identity and Access Management

Управління доступом у GCP будується на трьох поняттях: principal, role та resource [18]. Principal — це той, хто щось робить: користувач, група, сервісний акаунт або федерована ідентичність. Role — набір дозволів. Resource — об'єкт, до якого застосовується роль. Ієрархія Organization →

Folder → Project → Resource дозволяє задавати політики на будь-якому рівні з автоматичним наслідуванням вниз. Ключовий елемент для CI/CD — сервісні акаунти та Workload Identity Federation: зовнішній агент (GitHub Actions) отримує тимчасові права через обмін OIDC-токена замість збереження JSON-ключа. Модуль iam у проекті створює акаунт gke-nodes-sa і видає йому рівно п'ять ролей за принципом найменших привілеїв: logging.logWriter, monitoring.metricWriter, artifactregistry.reader, secretmanager.secretAccessor, cloudsql.client.

1.6 Zero Trust та Workload Identity Federation

Мережевий периметр як концепція безпеки зживає себе у хмарному середовищі. Розробники підключаються з кав'ярень через особисті ноутбуки, мікросервіси розміщені в різних регіонах і хмарах, SaaS-сервіси взагалі знаходяться поза корпоративною мережею. Підхід «всередині мережі — довіряємо» перетворює будь-який компрометований вузол на плацдарм для горизонтального переміщення [15].

Zero Trust пропонує іншу модель: «ніколи не довіряй, завжди перевіряй». Її ключові принципи [15; 27]: кожен запит автентифікується незалежно від походження; доступ надається за принципом найменших привілеїв (PoLP); мережа сегментується аж до окремого Pod-а; усі взаємодії шифруються; рішення про доступ ухвалюється з урахуванням повного контексту запиту — хто, з якого пристрою і до якого ресурсу звертається.

Google описав власну реалізацію Zero Trust у двох відкритих статтях. BeyondCorp [24] — це модель доступу користувачів, де рішення про авторизацію ухвалюється на основі стану пристрою та ідентичності, а не приналежності до VPN-мережі. BeyondProd [26] поширює ті самі принципи на взаємодію між сервісами всередині production: кожен сервіс має криптографічну ідентичність на ALTS-сертифікатах, весь трафік шифрується, а права доступу описуються декларативно. Обидві моделі стали основою для продуктів GCP — Identity-Aware Proxy та Workload Identity.

Workload Identity Federation (WIF) — механізм Zero Trust для CI/CD без довгострокових ключів. Раніше стандартна схема виглядала так: створити сервісний акаунт у GCP, експортувати ключ у JSON і покласти його у GitHub Secrets. Такий ключ довгостроковий, його ротація трудомістка, а витік рівнозначний повній компрометації акаунта.

WIF замінює це обміном OIDC-токена на короткоживучий access-токен GCP. Як це працює:

1. У проекті GCP створюється Workload Identity Pool — контейнер для зовнішніх ідентичностей.

2. У пул додається Workload Identity Provider із параметрами OIDC-провайдера (issuer URL, audience, mapping атрибутів).

3. Сервісний акаунт отримує роль roles/iam.workloadIdentityUser, прив'язану до конкретних умов — наприклад, тільки workflow з гілки main певного репозиторію.

4. У runtime CI-агент бере OIDC-токен від GitHub, обмінює його у Security Token Service на access-токен GCP. Токен живе ~1 годину і ніде надовго не зберігається [18].

У проекті це реалізовано в .github/workflows/cd.yml через action google-github-actions/auth з параметрами workload_identity_provider та service_account, що передаються через GitHub Secrets лише як ідентифікатори (не самі секрети).

Аналогічна механіка існує всередині GKE для подів — Workload Identity для подів. Kubernetes Service Account (KSA) діє від імені Google Service Account (GSA) без зберігання ключів у Pod-і. Це робить GKE Metadata Server, який підмінює стандартний metadata-ендпоінт ноди [21]. У проекті KSA web-app-ksa у namespace default прив'язано до GSA gke-nodes-sa через IAM-binding ролі roles/iam.workloadIdentityUser. Як результат — у проекті немає жодного довгострокового ключа: ні у Git, ні у GitHub Secrets, ні у Pod-і.

Висновок

У першому розділі розглянуто методологію DevOps та парадигми автоматизації, проведено аналіз хмарних платформ та обрано Google Cloud Platform як базу для реалізації проекту. Детально описано ключові сервіси GCP і механізм Workload Identity Federation для безключової автентифікації.

РОЗДІЛ 2
АНАЛІЗ АРХІТЕКТУРИ ТА ПРОЕКТУВАННЯ СИСТЕМИ АВТОМАТИЗАЦІЇ

2.1 Постановка задачі та функціональні вимоги

Система gke-terraform-project — референсна реалізація Cloud-Native платформи на Google Cloud Platform для розгортання web-застосунку зі stateful-залежністю (PostgreSQL). Ціль системи — забезпечити повний автоматизований життєвий цикл застосунку: від push коду в Git до self-healing pod'ів у production без жодного ручного кроку.

Функціональні вимоги до системи:

ФВ-1. Розгортання інфраструктури з нуля в порожньому GCP-проекті однією командою terraform apply без додаткових скриптів.

ФВ-2. Автоматичний CI/CD-конвеєр, який триггериться кожним push у гілку main та доводить зміни до production без ручних підтверджень.

ФВ-3. Безкейова автентифікація CI/CD-агента в GCP — без зберігання сервісних ключів у GitHub Secrets або інших місцях.

ФВ-4. Безпечна доставка секретів (пароль до бази даних) у pod'и без зберігання у Kubernetes Secret чи у Docker-образі.

ФВ-5. Доступність web-застосунку з інтернету через L7-балансувальник з кастомним health-check на ендпоінті /healthz.

ФВ-6. Self-healing pod'ів при відмовах: автоматичний перезапуск після збою, видалення з пулу балансувальника при недоступності.

ФВ-7. Повне знищення всіх ресурсів однією командою terraform destroy для уникнення залишкових витрат.

Кожна вимога перевіряється бінарно — виконується або ні, що дозволяє у Розділі 4 побудувати тестові сценарії для їх верифікації.

2.2 Нефункціональні вимоги

Надійність. Цільовий показник доступності web-застосунку — 99,9 % за місяць (≈43 хвилини downtime). Досягається через мінімум 2 репліки pod'а, readiness/liveness probes на /healthz, rolling update без downtime, а також Spot-ноди з автоматичним перезапуском Pod-ів на іншій ноді при витисненні.

Безпека. Критерії: жодного довгострокового сервісного ключа в коді, секретах CI/CD або Pod-і; жодного публічного IP у бази даних; принцип

найменших привілеїв (PoLP) для всіх сервісних акаунтів; обов'язковий vulnerability-scan образів і файлової системи репозиторію перед деплоєм; виконання pod’а від непривілейованого користувача (UID 1000) з readOnlyRootFilesystem.

Спостережуваність. Усі логи stdout/stderr контейнерів автоматично потрапляють у Cloud Logging без додаткових агентів. Метрики системних компонентів кластера — у Cloud Monitoring. Доступний kubectl rollout status з виведенням kubectl events і логів падаючого контейнера у разі невдалого релізу.

Вартість. Цільовий бюджет — до $100/місяць для академічного pet-project. Досягається використанням Spot-нод (-91 %), мінімальної конфігурації Cloud SQL (db-f1-micro), регіональним зберіганням образів та однією репліковою регіональною GCS-конфігурацією для state.

Підтримка. Уся інфраструктура та CI/CD описані як код у Git; будь-яка зміна проходить code review; README та коментарі у Terraform-модулях слугують документацією.

2.3 Аналіз обраного стеку технологій

Вибір кожного інструменту обґрунтований альтернативами та критеріями, що випливають з вимог §§ 2.1–2.2.

Інструмент IaC — Terraform [22]. Pulumi підтримує повноцінні мови програмування Python і TypeScript, проте має меншу спільноту та менш стабільні GCP-провайдери. Crossplane є Kubernetes-native IaC, але вимагає наявного K8s-кластера ще до його створення, що ускладнює початкове розгортання. Terraform обрано за зрілість, найбільшу спільноту, стабільні google та google-beta провайдери та широку базу прикладів.

Контейнеризація — Docker з форматом OCI [23]. Buildah та Podman використовують той самий OCI-формат, але Docker залишається стандартом локальної розробки. Багатоетапний Dockerfile (builder → final) дозволяє отримати мінімальний runtime-образ без інструментів збірки.

Оркестрація — Kubernetes у вигляді GKE Standard [21]. Cloud Run як альтернатива простіший в управлінні, але не дозволив би продемонструвати sidecar, CSI Driver, Workload Identity для подів та Ingress із BackendConfig. GKE Autopilot автоматично керує нодами, що знижує навчальну цінність для академічного дослідження.

CI/CD — GitHub Actions. Cloud Build є managed-сервісом, але платним і менш гнучким у налаштуванні тригерів; Jenkins вимагає самостійного хостингу та постійного супроводу. GitHub Actions обрано через нативну інтеграцію з GitHub-репозиторієм, безкоштовність для public-репо, готову підтримку Workload Identity Federation через action google-github-actions/auth та багату екосистему готових кроків — TFLint, Hadolint, Trivy.

Менеджер пакетів Kubernetes — Helm. Використовується для розгортання kube-prometheus-stack у namespace monitoring через скрипт scripts/install-monitoring.sh. Власні K8s-маніфести застосунку залишено у вигляді plain-YAML без шаблонізації, бо обсяг не виправдовує ускладнення.

Сумарно стек є гібридним: open-source-інструменти (Terraform, Docker, Kubernetes, Helm) для портативності та контролю + managed-сервіси GCP (GKE control plane, Artifact Registry, Cloud SQL, Secret Manager, Cloud Operations) для скорочення операційного навантаження.

2.4 Проектування мережевої топології

Усю інфраструктуру розміщено у регіоні europe-west1 (Бельгія, St. Ghislain) — одному з найстабільніших європейських регіонів GCP з повною підтримкою всіх використаних сервісів та широким каталогом типів VM. Для подових нод обрано зону europe-west1-b як зональний кластер — це дешевший варіант порівняно з регіональним; multi-zone конфігурацію розглянуто у Розділі 4 як напрямок розвитку.

Мережева архітектура побудована за принципом «одна VPC — одна приватна підмережа». У підмережі виділено три CIDR-діапазони: первинний для нод GKE та два secondary для Pod-ів і Services Kubernetes. Розподіл наведено у Таблиці 2.1.

Таблиця 2.1

Розподіл CIDR-діапазонів

Призначення	CIDR	Кількість IP	Обґрунтування
Ноди GKE (subnet primary)	10.0.0.0/24	254	Початково /28 (14 IP) — вичерпано після 3 нод + системних подів. Розширено до /24 з запасом на майбутнє.
Pod-и (secondary range)	10.1.0.0/20	4 094	GKE за замовчуванням резервує /24 на ноду; запас на 16 нод × 110 подів.
Services (secondary range)	10.2.0.0/24	254	Достатньо для типового pet-project; ClusterIP-сервіси не потребують багато адрес.
GKE control plane	172.16.0.0/28	16	Окрема приватна мережа Google для Master endpoint.

Cloud NAT забезпечує контрольований egress нод GKE у публічний інтернет . Нодам не присвоєно публічних IP — це закриває їх від прямого сканування з інтернету.

Для приватного підключення до Cloud SQL налаштовано VPC Peering з servicenetworking.googleapis.com. Cloud SQL отримує IP із зарезервованого діапазону google-managed-services-main-vpc (/16) у тій самій VPC, тому pod’и звертаються до бази по приватному IP, а не через публічний інтернет.

Брандмауер VPC побудований за принципом deny-by-default. Окремим правилом priority 950 дозволено вхідний трафік від офіційних діапазонів health-checkers Google Cloud Load Balancer (130.211.0.0/22 та 35.191.0.0/16) на порти 8080 (порт застосунку) та 30000–32767 (NodePort range), що використовує GCE Ingress. Master authorized networks GKE відкритий для CIDR VPC та 0.0.0.0/0 — це необхідно, щоб GitHub Actions runner міг звернутися до Master endpoint після get-credentials.

2.5 Проектування моделі ідентифікації та безпеки

Усі ідентичності у системі поділено на машинні (нод GKE, pod’и, GitHub Actions) та людські (адміністратор проекту). Для машинних ідентичностей не застосовуються довгострокові ключі — лише федеративні токени та Workload Identity. Людський адміністратор працює через Cloud IAM з MFA.

Центральний сервісний акаунт — gke-nodes-sa@gke-petproject-2026.iam.gserviceaccount.com. Від його імені діють ноди GKE та pod’и через

прив'язку KSA web-app-ksa у namespace default. Список ролей акаунта (Таблиця 2.2) сформовано за принципом найменших привілеїв (PoLP): кожна роль покриває одну конкретну операцію, що очікується в runtime.

Таблиця 2.2

Ролі сервісного акаунта gke-nodes-sa

Роль IAM	Призначення
roles/logging.logWriter	Запис stdout/stderr контейнерів у Cloud Logging
roles/monitoring.metricWriter	Експорт метрик у Cloud Monitoring
roles/artifactregistry.reader	Pull Docker-образів з Artifact Registry
roles/secretmanager.secretAccessor	Читання секретів через CSI Driver
roles/cloudsql.client	Підключення до Cloud SQL через cloud-sql-proxy
roles/iam.workloadIdentityUser	Дозвіл KSA web-app-ksa виступати від імені GSA (binding на акаунт, не project-роль)

Для GitHub Actions передбачений окремий механізм — Workload Identity Federation. У GCP створюється Workload Identity Pool github-pool та Provider github-provider з валідацією OIDC-токенів від token.actions.githubusercontent.com. Окремий GSA отримує роль roles/iam.workloadIdentityUser, прив'язану до конкретного репозиторію та гілки main. Як результат, GitHub Actions runner отримує access-токен GCP з життям ~1 година без жодного експортованого ключа [18].

Кінцева властивість моделі: у системі немає жодного довгострокового ключа сервісного акаунта — ні у Git-репозиторії, ні у GitHub Secrets, ні всередині Pod-а. Усі секрети — або федеративні токени з коротким терміном життя, або секрети застосунку (пароль до бази), що зберігаються у Secret Manager та доставляються у pod’и через CSI Driver як файли.

2.6 Архітектура CI/CD-конвеєра

CI/CD-конвеєр спроектований як спрямований ациклічний граф (DAG) з двох послідовних jobs у GitHub Actions workflow .github/workflows/cd.yml. Триггер — push у гілку main.

Перший job — lint-and-scan — виконує статичні перевірки коду паралельно. TFLint валідує Terraform-конфігурацію на синтаксичні помилки та anti-patterns. Hadolint перевіряє Dockerfile на дотримання best practices. Trivy сканує файлову систему репозиторію та виявляє відомі CVE з рівнем

CRITICAL та HIGH; ignore-unfixed: true виключає вразливості без доступних патчів. Будь-яка помилка призводить до exit-code 1 і блокує наступний job — це реалізує принцип shift-left security з Розділу 1.

Другий job — deploy — стартує лише за умови успіху першого (needs: lint-and-scan). Він виконується послідовно і складається з таких кроків:

1. Автентифікація через google-github-actions/auth з Workload Identity Federation.
2. Setup Terraform 1.5.0 та terraform init + apply -auto-approve у директорії environments/dev.
3. Витягування output-ів модулів (db_instance_connection_name, db_password_secret_id) для подальшої підстановки у K8s-маніфести.
4. Встановлення gke-gcloud-auth-plugin та отримання kubeconfig через gcloud container clusters get-credentials.
5. Збірка Docker-образу з тегом git rev-parse --short HEAD та push в Artifact Registry.
6. sed-підстановка тегу образу та параметрів СУБД у k8s/deployment.yaml і k8s/dependencies.yaml.
7. kubectl apply -f k8s/ та kubectl rollout status deployment/web-app з таймаутом 10 хвилин.
8. У разі падіння rollout — автоматичний вивід kubectl get events і kubectl logs deployment/web-app для діагностики.

Стратегія релізу — rolling update за замовчуванням Kubernetes Deployment з maxSurge=25 % та maxUnavailable=25 %. Для двох реплік це означає, що у будь-який момент часу принаймні одна нова репліка готова приймати трафік перед видаленням старої. Перехід до прогресивних стратегій (canary, blue-green) через Argo Rollouts або Cloud Deploy винесено у напрямки розвитку.

2.7 Проектування модульної структури Terraform

Декомпозиція виконана за двома принципами. Перший — один модуль відповідає за одну логічну область (мережа, ідентифікація, СУБД, оркестрація, реєстр). Другий — модулі не залежать один від одного напряму; вся міжмодульна комунікація проходить через input/output-змінні, що передаються у файлі environments/dev/main.tf.

Така структура має дві переваги. Модулі переносяться між середовищами (dev/staging/prod) шляхом створення нової директорії environments/<name> з тими самими module-блоками і відмінними параметрами. Зміни всередині модуля не вимагають правок інших модулів — лише оновлення environment-файлу при зміні контракту.

Композицію та контракти модулів узагальнено у Таблиці 2.3.

Таблиця 2.3

Модулі Terraform та їх контракти

Модуль	Ключові ресурси	Основні output-и
vpc	google_compute_network, subnetwork (з secondary ranges), router, NAT, VPC peering	network_id, network_name, subnet_name
iam	GSA, IAM-bindings (5 ролей), Workload Identity binding для KSA	sa_email
db	Cloud SQL Postgres 15, random_password, secret + secret_version, database, user	instance_connection_name, db_name, db_user, db_password_secret_id
gke	google_container_cluster (приватний), 2 node_pool, firewall для health checkers	—
registry	google_artifact_registry_repository (DOCKER)	repo_url

Окремо описано директорію environments/dev. Файл backend.tf зберігає state у GCS-бакеті gke-petproject-2026/terraform/state з вбудованим state-locking та Object Versioning для відкату на попередню версію стану. Файл versions.tf фіксує версії провайдерів (google ~> 5.0, google-beta ~> 5.0, random ~> 3.6) та мінімальну версію Terraform 1.5.0. Файл providers.tf задає project_id та регіон. Файл main.tf композує модулі — це єдине місце, де модулі зустрічаються.

Така структура витримує тиражування на нові середовища: щоб додати environments/staging, достатньо скопіювати директорію environments/dev, змінити project_id у providers.tf та CIDR-діапазони у main.tf за потреби.

2.8 Архітектурна діаграма системи

Архітектура системи зображена на Рис.2.1 (файл gke-terraform-project.png у репозиторії проекту). Логічно вона ділиться на чотири шари: рівень розробника (Git-репозиторій, GitHub Actions), рівень GCP-інфраструктури (VPC, GKE, Cloud SQL, Secret Manager, Artifact Registry, Cloud Operations), рівень робочих навантажень (pod’и у кластері) та рівень користувача (інтернет-трафік через GCE Load Balancer).

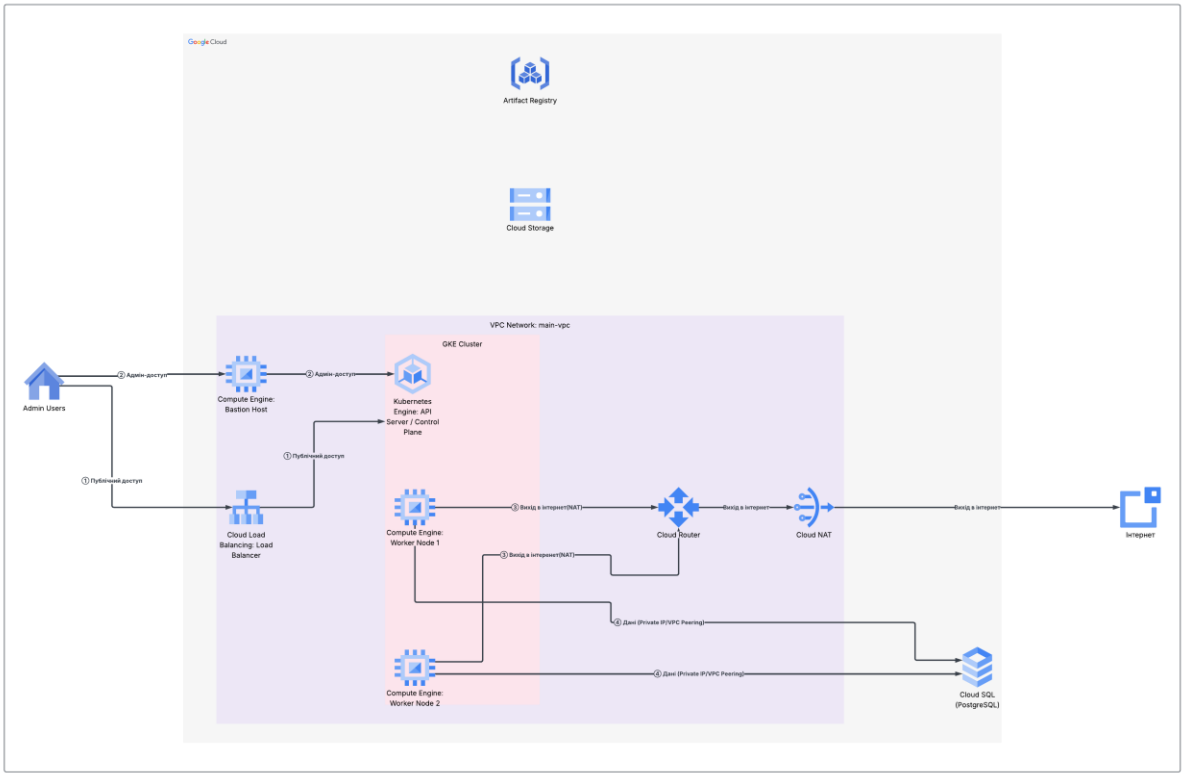

Рис.2.1. Архітектура системи автоматизації хмарної інфраструктури

Маршрут даних від коду до користувача: розробник пушить зміни у main → GitHub Actions триггериться → автентифікується у GCP через Workload Identity Federation → terraform apply узгоджує інфраструктуру з кодом → docker build пушить образ в Artifact Registry → kubectl apply розгортає Deployment у GKE → GCE Load Balancer розподіляє вхідний трафік на pod'и через NEG → кожен Pod містить контейнери app та cloud-sql-proxy.

Маршрут запитів від користувача до бази: HTTP-запит → GCE Load Balancer → BackendConfig health-check на /healthz → NEG → Pod web-app, контейнер app (порт 8080) → SQLAlchemy → localhost:5432 → sidecar cloud-sql-proxy → mTLS-тунель приватним IP до Cloud SQL → відповідь у зворотному порядку.

Маршрут секретів: пароль генерується модулем db через random_password → зберігається у Secret Manager (db-password-XXXX) → під час старту Pod-а Secret Manager CSI Driver монтує секрет у /secrets/db-password → застосунок зчитує файл при ініціалізації SQLAlchemy engine.

Висновок

У другому розділі сформульовано функціональні та нефункціональні вимоги до системи, обрано стек технологій і спроектовано архітектуру: мережеву топологію, модель безпеки, CI/CD-конвеєр та модульну структуру Terraform.

РОЗДІЛ 3
ПРАКТИЧНА РЕАЛІЗАЦІЯ СИСТЕМИ АВТОМАТИЗАЦІЇ

3.1 Підготовка середовища та налаштування backend Terraform

Перед запуском Terraform виконано три підготовчі кроки: активація потрібних GCP API, створення GCS-бакета для зберігання state та налаштування Workload Identity Pool для GitHub Actions.

Активація API виконується одноразово командами gcloud для проекту gke-petproject-2026:

Лістинг 3.1 — Активація необхідних API GCP

```
gcloud services enable \
  container.googleapis.com \
  sqladmin.googleapis.com \
  secretmanager.googleapis.com \
  iamcredentials.googleapis.com \
  artifactregistry.googleapis.com \
  servicenetworking.googleapis.com \
  --project=gke-petproject-2026
```

GCS-бакет для state створюється у тому самому проекті з увімкненим Object Versioning, щоб мати можливість відкоту на попередню версію стану інфраструктури:

Лістинг 3.2 — Створення бакета для Terraform state

```
gsutil mb -l europe-west1 gs://gke-petproject-2026
gsutil versioning set on gs://gke-petproject-2026
```

Backend Terraform налаштовано на цей бакет. GCS за замовчуванням підтримує state-locking, додаткові налаштування (як DynamoDB для AWS) не потрібні [22]:

Лістинг 3.3 — environments/dev/backend.tf

```
terraform {
  backend "gcs" {
    bucket = "gke-petproject-2026"
    prefix = "terraform/state"
  }
}
```

Файл versions.tf фіксує мінімальну версію Terraform та діапазон версій провайдерів — це гарантує відтворюваність між запусками CI/CD та локальної розробки:

Лістинг 3.4 — environments/dev/versions.tf

```
terraform {
  required_version = ">= 1.5.0"
  required_providers {
    google      = { source = "hashicorp/google",      version = "~> 5.0" }
    google-beta = { source = "hashicorp/google-beta", version = "~> 5.0" }
    random      = { source = "hashicorp/random",      version = "~> 3.6" }
  }
}
```

Workload Identity Pool github-pool та Provider github-provider створюються одноразово командами gcloud iam workload-identity-pools з прив'язкою до конкретного GitHub-репозиторію. Згенеровані ідентифікатори потім заносяться у GitHub Secrets як WORKLOAD_IDENTITY_PROVIDER та SERVICE_ACCOUNT — це не самі секрети, а лише посилання на сутності у GCP [18].

3.2 Реалізація модуля vpc

Модуль vpc створює VPC main-vpc, приватну підмережу з двома secondary-діапазонами для подів і сервісів Kubernetes, Cloud Router з Cloud NAT для контрольованого egress нод у інтернет, а також VPC Peering з servicenetworking.googleapis.com для приватного доступу до Cloud SQL.

Лістинг 3.5 — modules/vpc/main.tf (ключові фрагменти)

```
resource "google_compute_network" "main" {
  name                    = var.network_name
  auto_create_subnetworks = false
}

resource "google_compute_subnetwork" "private" {
  name          = "${var.network_name}-private-subnet"
  ip_cidr_range = var.subnet_cidr        # 10.0.0.0/24 для нод
  region        = var.region
  network       = google_compute_network.main.id
  private_ip_google_access = true

  secondary_ip_range {
    range_name    = "k8s-pods-range"
    ip_cidr_range = var.pods_cidr        # 10.1.0.0/20
  }
  secondary_ip_range {
    range_name    = "k8s-services-range"
    ip_cidr_range = var.services_cidr    # 10.2.0.0/24
  }
}
```

Cloud Router та NAT забезпечують нодам доступ до публічного інтернету — для завантаження образів і оновлень пакетів — без присвоєння нодам публічних IP:

Лістинг 3.6 — modules/vpc/main.tf (Cloud NAT)

```
resource "google_compute_router" "router" {
  name    = "${var.network_name}-router"
  region  = var.region
  network = google_compute_network.main.id
}

resource "google_compute_router_nat" "nat" {
  name   = "${var.network_name}-nat"
  router = google_compute_router.router.name
  region = var.region
  nat_ip_allocate_option             = "AUTO_ONLY"
  source_subnetwork_ip_ranges_to_nat = "ALL_SUBNETWORKS_ALL_IP_RANGES"
}
```

VPC Peering до servicenetworking.googleapis.com — обов'язковий крок для приватного підключення Cloud SQL. Резервується глобальна адреса /16, яка передається у service_networking_connection:

Лістинг 3.7 — modules/vpc/main.tf (VPC peering)

```
resource "google_compute_global_address" "private_ip_address" {
  name          = "google-managed-services-${var.network_name}"
  purpose       = "VPC_PEERING"
  address_type  = "INTERNAL"
  prefix_length = 16
  network       = google_compute_network.main.id
}

resource "google_service_networking_connection" "private_vpc_connection" {
  network                 = google_compute_network.main.id
  service                 = "servicenetworking.googleapis.com"
  reserved_peering_ranges = [google_compute_global_address.private_ip_address.name]
}
```

3.3 Реалізація модуля GKE

Модуль gke створює приватний кластер gke-pet-cluster з двома node pool: workload-pool на Spot-нодах для основного навантаження та system-pool на on-demand-нодах для системних подів. Кластер сконфігуровано з Workload Identity, Secret Manager add-on, а також logging/monitoring у Cloud Operations Suite [21].

Лістинг 3.8 — modules/gke/main.tf (кластер)

```
resource "google_container_cluster" "primary" {
```

```
  provider = google-beta
  name     = "gke-pet-cluster"
  location = var.region

  initial_node_count       = 1
  remove_default_node_pool = true       # створимо власні node pool

  network    = var.network_name
  subnetwork = var.subnet_name

  ip_allocation_policy {
    cluster_secondary_range_name  = var.pods_range_name
    services_secondary_range_name = var.services_range_name
  }

  private_cluster_config {
    enable_private_nodes    = true
    enable_private_endpoint = false
    master_ipv4_cidr_block  = "172.16.0.0/28"
  }

  workload_identity_config {
    workload_pool = "${var.project_id}.svc.id.goog"
  }

  secret_manager_config { enabled = true }

  logging_config    { enable_components = ["SYSTEM_COMPONENTS", "WORKLOADS"] }
  monitoring_config { enable_components = ["SYSTEM_COMPONENTS"] }
}
```

Прапор enable_private_nodes = true забороняє нодам отримувати публічні IP — увесь зовнішній трафік йде через Cloud NAT. enable_private_endpoint = false залишає Master endpoint публічно доступним, що дозволяє GitHub Actions runner-у звертатися до нього через kubectl get-credentials без VPN. Блок master_authorized_networks_config (у лістингу скорочено) додає CIDR-фільтрацію: 10.0.0.0/8 для внутрішнього VPC-трафіку та 0.0.0.0/0 для GitHub Actions runners — прийнятний компроміс для pet-project, де IP-адреси CI/CD-агентів динамічні.

Workload pool використовує Spot-ноди e2-standard-2, що дає економію до 91 % порівняно з on-demand. На випадок витиснення Spot-ноди GKE автоматично перезапускає pod’и на іншій ноді:

Лістинг 3.9 — modules/gke/main.tf (node pools)

```
resource "google_container_node_pool" "workload_nodes" {
  name       = "workload-pool"
  cluster    = google_container_cluster.primary.name
  node_count = 2
  node_config {
    spot         = true                  # знижка до 91 %
```

```
    machine_type = "e2-standard-2"
    disk_size_gb = 30
    service_account = var.service_account_email
    oauth_scopes    = ["https://www.googleapis.com/auth/cloud-platform"]
    tags            = ["gke-node"]
    workload_metadata_config { mode = "GKE_METADATA" }
  }
}
```

Окремий файл modules/gke/firewall.tf додає правило для health-checkers GCE Load Balancer. Без цього правила Ingress не може підтвердити, що pod'и живі, і не маршрутизує до них трафік:

Лістинг 3.10 — modules/gke/firewall.tf

```
resource "google_compute_firewall" "allow_gclb_health_checks" {
  name      = "gke-pet-cluster-allow-gclb-hc"
  network   = var.network_name
  direction = "INGRESS"
  priority  = 950
  source_ranges = ["130.211.0.0/22", "35.191.0.0/16"]  # офіційні діапазони GCLB
  target_tags   = ["gke-node"]
  allow {
    protocol = "tcp"
    ports    = ["8080", "30000-32767"]   # порт додатку + NodePort range
  }
}
```

3.4 Реалізація модуля Cloud SQL та Secret Manager

Модуль db створює Cloud SQL інстанс PostgreSQL 15 без публічного IP, генерує випадковий пароль через random_password і кладе його у Secret Manager. Сервісний акаунт gke-nodes-sa має роль secretmanager.secretAccessor (Розділ 2.5), тому pod'и можуть прочитати секрет без жодного зайвого ключа [20]. Окрім інстансу, модуль також створює ресурси google_sql_database (база даних app_db) та google_sql_user (користувач app_user) — вони формують повну схему підключення та у лістингах нижче скорочені задля компактності.

Лістинг 3.11 — modules/db/main.tf (секрет)

```
resource "random_id" "suffix" { byte_length = 4 }

resource "random_password" "db_password" {
  length  = 16
  special = false      # уникаємо проблем у connection string
}

resource "google_secret_manager_secret" "db_pass" {
  secret_id  = "db-password-${random_id.suffix.hex}"
  project    = var.project_id
  replication { auto {} }
}
```

```
resource "google_secret_manager_secret_version" "db_pass_val" {
  secret      = google_secret_manager_secret.db_pass.id
  secret_data = random_password.db_password.result
}
```

Інстанс Cloud SQL налаштовано на мінімальний tier db-f1-micro, без публічного IP — тільки приватне підключення через VPC Peering (Розділ 3.2):

Лістинг 3.12 — modules/db/main.tf (Cloud SQL)

```
resource "google_sql_database_instance" "main" {
  name                = "gke-pet-db-${random_id.suffix.hex}"
  database_version    = "POSTGRES_15"
  region              = var.region
  deletion_protection = false       # тільки для pet-project!
  settings {
    tier = "db-f1-micro"
    ip_configuration {
      ipv4_enabled    = false       # публічний IP вимкнено
      private_network = var.vpc_id  # тільки приватний доступ
    }
  }
}
```

3.5 Реалізація модуля IAM

Модуль iam створює сервісний акаунт gke-nodes-sa, видає йому п'ять project-рівневих ролей за принципом найменших привілеїв (Таблиця 2.2) та налаштовує Workload Identity binding для Kubernetes Service Account web-app-ksa у namespace default. Додатково модуль вмикає API iamcredentials.googleapis.com через ресурс google_project_service — необхідна умова для коректної роботи Workload Identity Federation.

Лістинг 3.13 — modules/iam/main.tf (GSA + project-ролі)

```
resource "google_service_account" "gke_sa" {
  account_id   = "gke-nodes-sa"
  display_name = "GKE Nodes Service Account"
}

resource "google_project_iam_member" "gke_logging" {
  project = var.project_id
  role    = "roles/logging.logWriter"
  member  = "serviceAccount:${google_service_account.gke_sa.email}"
}
# Аналогічно: monitoring.metricWriter, artifactregistry.reader,
#             secretmanager.secretAccessor, cloudsql.client
```

Ключовий момент Zero Trust — binding ролі workloadIdentityUser, який дозволяє Kubernetes Service Account web-app-ksa виступати від імені Google

Service Account gke-nodes-sa. Цей binding робиться не на проект, а на сам сервісний акаунт:

Лістинг 3.14 — modules/iam/main.tf (Workload Identity binding)

```
resource "google_service_account_iam_binding" "workload_identity_binding" {
  service_account_id = google_service_account.gke_sa.name
  role               = "roles/iam.workloadIdentityUser"
  members = [
    "serviceAccount:${var.project_id}.svc.id.goog[default/web-app-ksa]"
  ]
}
```

3.6 Реалізація модуля Artifact Registry

Найменший модуль проекту — створює один Docker-репозиторій у регіоні europe-west1 та повертає його URL як output для CI/CD-конвеєра [19]:

Лістинг 3.15 — modules/registry/main.tf

```
resource "google_artifact_registry_repository" "my_repo" {
  location      = var.region
  repository_id = "my-repo"
  description   = "Docker repository"
  format        = "DOCKER"
}

output "repo_url" {
  value = "${var.region}-
docker.pkg.dev/${var.project_id}/${google_artifact_registry_repository.my_repo.reposi
tory_id}"
}
```

3.7 Контейнеризація застосунку

Застосунок — Flask + Gunicorn + SQLAlchemy. Образ зібрано через багатоетапний Dockerfile: на першому етапі встановлюються системні пакети для компіляції psycopg2 (gcc, libpq-dev) та залежності Python у віртуальне середовище; на другому — копіюється лише готовий venv та код застосунку, без інструментів збірки.

Лістинг 3.16 — app/Dockerfile

```
# --- Етап 1: Збірка ---
FROM python:3.11-slim as builder
WORKDIR /app
RUN apt-get update && apt-get install -y --no-install-recommends \
    gcc libpq-dev && rm -rf /var/lib/apt/lists/*
COPY requirements.txt .
RUN python -m venv /opt/venv
ENV PATH="/opt/venv/bin:$PATH"
RUN pip install --no-cache-dir -r requirements.txt
```

```
# --- Етап 2: Фінальний образ ---
FROM python:3.11-slim
WORKDIR /app
RUN apt-get update && apt-get install -y --no-install-recommends libpq5 && \
    rm -rf /var/lib/apt/lists/*
COPY --from=builder /opt/venv /opt/venv
COPY app.py .
ENV PATH="/opt/venv/bin:$PATH"
RUN groupadd -g 1000 appgroup && \
    useradd -u 1000 -g appgroup -s /bin/sh -m appuser
USER appuser
EXPOSE 8080
CMD ["gunicorn", "--bind", "0.0.0.0:8080", "--workers", "2", "app:app"]
```

Створення непривілейованого користувача appuser — обов'язкова умова для запуску pod'а з runAsNonRoot: true у Kubernetes (Розділ 3.8). Gunicorn використовується замість Flask development server через production-готовність: він багатопоточний і коректно обробляє SIGTERM при rolling update.

У app.py реалізовано три нетривіальні рішення. Перше — читання пароля з файлу, змонтованого CSI Driver-ом, а не зі змінної оточення:

Лістинг 3.17 — app/app.py (читання секрету з файлу)

```
def get_secret(secret_name, default_value=None):
    secret_path = f"/secrets/{secret_name}"
    try:
        with open(secret_path, 'r') as f:
            return f.read().strip()
    except IOError:
        return default_value

DB_PASS = get_secret("db-password", "default_password")
```

Друге — lazy-ініціалізація SQLAlchemy engine. Engine створюється не при старті Gunicorn, а при першому HTTP-запиті. Це усуває проблему crash-loop, коли Gunicorn стартує швидше за sidecar cloud-sql-proxy і не може встановити з'єднання з базою:

Лістинг 3.18 — app/app.py (lazy DB init)

```
_engine = None
def get_engine():
    global _engine
    if _engine is None:
        _engine = create_engine(DATABASE_URL, pool_pre_ping=True)
    return _engine
```

Третє — ендпоінт /healthz, який повертає 200 OK завжди, якщо процес живий, без перевірки доступу до бази даних. Це принципово: liveness probe не

повинна залежати від стану зовнішніх сервісів, інакше Kubernetes перезапускатиме pod’и при тимчасовій недоступності бази, що тільки погіршить ситуацію.

3.8 Розробка Kubernetes-маніфестів

У директорії k8s/ зібрано п'ять YAML-файлів: dependencies.yaml (KSA + SecretProviderClass), deployment.yaml, service.yaml, ingress.yaml та backendconfig.yaml. Розглянемо ключові.

Deployment містить два контейнери у одному Pod-і: основний app та sidecar cloud-sql-proxy. SecurityContext реалізує принципи Cloud-Native security з Розділу 1: запуск від UID 1000, заборона привілеїв та read-only коренева ФС:

Лістинг 3.19 — k8s/deployment.yaml (ключові параметри)

```
spec:
  replicas: 2
  template:
    spec:
      serviceAccountName: web-app-ksa
      securityContext:
        runAsNonRoot: true
        runAsUser: 1000
        fsGroup: 2000
      containers:
        - name: app
          image: "europe-west1-docker.pkg.dev/.../my-app:latest"
          ports: [{ containerPort: 8080 }]
          volumeMounts:
            - { name: secret-volume, mountPath: "/secrets", readOnly: true }
            - { name: tmp-volume,    mountPath: "/tmp" }
          securityContext:
            allowPrivilegeEscalation: false
            readOnlyRootFilesystem:   true
```

Том tmp-volume типу emptyDir потрібен тому, що Gunicorn пише heartbeats у /tmp, а коренева ФС у нас read-only. Це класичне поєднання read-only root + writable tmp для дотримання принципу Immutable Infrastructure (Розділ 1.2.4).

Sidecar cloud-sql-proxy створює зашифрований mTLS-тунель до Cloud SQL по приватному IP. Він прив'язується до localhost:5432 всередині Pod-а, тому app звертається до бази як до DB_HOST=127.0.0.1:

Лістинг 3.20 — k8s/deployment.yaml (cloud-sql-proxy sidecar)

```
- name: cloud-sql-proxy
  image: "gcr.io/cloud-sql-connectors/cloud-sql-proxy:2.8.0"
  args:
    - "--private-ip"
    - "$(DB_INSTANCE_CONNECTION_NAME)"
  env:
    - name: DB_INSTANCE_CONNECTION_NAME
      value: "<INSTANCE_CONNECTION_NAME>"   # підставляється у CI/CD
  securityContext:
    runAsNonRoot: true
    allowPrivilegeEscalation: false
```

Файл dependencies.yaml містить два об'єкти: Kubernetes Service Account з анотацією iam.gke.io/gcp-service-account, яка встановлює зв'язок KSA → GSA для Workload Identity, та SecretProviderClass, що описує, який саме секрет з Secret Manager монтувати:

Лістинг 3.21 — k8s/dependencies.yaml

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: web-app-ksa
  namespace: default
  annotations:
    iam.gke.io/gcp-service-account: gke-nodes-sa@gke-petproject-
2026.iam.gserviceaccount.com
---
apiVersion: secrets-store.csi.x-k8s.io/v1
kind: SecretProviderClass
metadata:
  name: db-credentials-provider
spec:
  provider: gke
  parameters:
    secrets: |
      - resourceName: "projects/gke-petproject-
2026/secrets/<DB_PASSWORD_SECRET_ID>/versions/latest"
        path: "db-password"
```

Service та Ingress відповідають за маршрутизацію зовнішнього трафіку. Service має тип NodePort — це вимога для коректної роботи GCE Ingress; Ingress — клас gce. Кастомний health-check на /healthz описано в окремому BackendConfig і прив'язано через анотацію cloud.google.com/backend-config:

Лістинг 3.22 — k8s/backendconfig.yaml

```
apiVersion: cloud.google.com/v1
kind: BackendConfig
metadata:
  name: web-app-backendconfig
```

```
spec:
  healthCheck:
    checkIntervalSec: 10
    timeoutSec: 5
    healthyThreshold: 2
    unhealthyThreshold: 2
    type: HTTP
    requestPath: /healthz
    port: 8080
```

3.9 Реалізація CI/CD-пайплайну в GitHub Actions

Workflow .github/workflows/cd.yml має два jobs (DAG з Розділу 2.6). Глобальний блок env зберігає спільні значення, щоб уникнути магічних рядків у кроках:

Лістинг 3.23 — .github/workflows/cd.yml (env та триггер)

```
name: Deploy to GKE
on:
  push:
    branches: [main]
env:
  GCP_PROJECT_ID: "gke-petproject-2026"
  TF_VAR_project_id: "gke-petproject-2026"
  GKE_CLUSTER:  "gke-pet-cluster"
  GKE_REGION:   "europe-west1-b"
  TF_WORKING_DIR: "environments/dev"
  K8S_DIR: "k8s"
  IMAGE_NAME: "europe-west1-docker.pkg.dev/gke-petproject-2026/my-repo/my-app"
```

Job lint-and-scan виконує три статичні перевірки. Trivy скеровано з ignore-unfixed: true, бо нефіксовані вразливості нічим не зарадити в коді — це повідомлення про необхідність оновлення базового образу:

Лістинг 3.24 — .github/workflows/cd.yml (Trivy scan)

```
- name: Run Trivy vulnerability scanner on Repo
  uses: aquasecurity/trivy-action@master
  with:
    scan-type: 'fs'
    scan-ref: '.'
    exit-code: '1'
    ignore-unfixed: true
    vuln-type: 'os,library'
    severity: 'CRITICAL,HIGH'
```

Job deploy потребує спеціальних permissions: id-token: write дозволяє GitHub Actions runner отримати OIDC-токен від token.actions.githubusercontent.com, який потім обмінюється на access-токен GCP. Без цього дозволу Workload Identity Federation не запрацює:

Лістинг 3.25 — .github/workflows/cd.yml (deploy job, auth)

```
deploy:
  needs: lint-and-scan
  permissions:
    contents: 'read'
    id-token: 'write'      # обов'язково для WIF
  steps:
    - id: 'auth'
      uses: 'google-github-actions/auth@v1'
      with:
        workload_identity_provider: ${{ secrets.WORKLOAD_IDENTITY_PROVIDER }}
        service_account: ${{ secrets.SERVICE_ACCOUNT }}
```

Після terraform apply вибираються output-и модуля db (instance_connection_name, db_password_secret_id) та підставляються через sed у заглушки k8s-маніфестів — це дешева альтернатива Helm/Kustomize для невеликої кількості значень:

Лістинг 3.26 — .github/workflows/cd.yml (sed та kubectl apply)

```
- name: Deploy to GKE
  run: |
    IMAGE_TAG=$(git rev-parse --short HEAD)
    sed -i "s|...my-app:latest|${{ env.IMAGE_NAME }}:${IMAGE_TAG}|g"
k8s/deployment.yaml
    sed -i "s|<INSTANCE_CONNECTION_NAME>|${{ steps.terraform-
outputs.outputs.db_instance_connection_name }}|g" k8s/deployment.yaml
    sed -i "s|<DB_PASSWORD_SECRET_ID>|${{ steps.terraform-
outputs.outputs.db_password_secret_id }}|g" k8s/dependencies.yaml
    kubectl apply -f k8s/
    kubectl rollout status deployment/web-app --timeout=10m || {
      echo "::error::Rollout failed!"
      kubectl get events --sort-by='.metadata.creationTimestamp' | tail -n 20
      kubectl logs deployment/web-app -c app --tail=50 || true
      exit 1
    }
```

Блок діагностики у разі падіння rollout — суттєвий момент. kubectl rollout status таймаутить через 10 хвилин, і без додаткового виводу подій та логів дебаг помилки за артефактами GitHub Actions майже неможливий. Цей фрагмент суттєво скорочує час локалізації несправного деплою.

3.10 Налаштування спостережуваності

Базова спостережуваність GKE увімкнена прапорцями logging_config та monitoring_config у Лістингу 3.8. Після цього всі логи stdout/stderr контейнерів та метрики системних компонентів автоматично потрапляють у Cloud Logging та Cloud Monitoring без додаткових агентів [25].

Для глибшого моніторингу робочих навантажень розгорнуто kube-prometheus-stack через Helm. Це open-source-стек з Prometheus, Grafana, Node Exporter та kube-state-metrics, який встановлюється скриптом scripts/install-monitoring.sh:

Лістинг 3.27 — scripts/install-monitoring.sh (ключові кроки)

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo update
kubectl create namespace monitoring --dry-run=client -o yaml | kubectl apply -f -
helm upgrade --install prometheus prometheus-community/kube-prometheus-stack \
  --namespace monitoring \
  --set alertmanager.enabled=false \
  --set grafana.service.type=ClusterIP \
  --set prometheus.prometheusSpec.serviceMonitorSelectorNilUsesHelmValues=false \
  --wait --timeout 10m
```

Параметр alertmanager.enabled=false вимикає Alertmanager — для академічного проекту достатньо алертів Cloud Monitoring. grafana.service.type=ClusterIP залишає Grafana доступною лише через kubectl port-forward, щоб не виставляти UI в інтернет. Доступ розробника: kubectl port-forward svc/prometheus-grafana 3000:80 -n monitoring.

3.11 Реалізація механізмів self-healing

Self-healing у проекті досягається трьома механізмами: readiness/liveness probes у Deployment, rolling update як стратегія релізу та автоматичне rescheduling Pod-ів при витисненні Spot-ноди.

Probes налаштовано так, щоб readiness реагувала швидко (5 с initialDelay, 10 с period), а liveness — повільніше (15 с initialDelay, 20 с period). Така асиметрія мінімізує false-positive перезапуски при тимчасових затримках старту:

Лістинг 3.28 — k8s/deployment.yaml (probes)

```
readinessProbe:
  httpGet: { path: /healthz, port: 8080 }
  initialDelaySeconds: 5
  periodSeconds: 10
  failureThreshold: 3
livenessProbe:
  httpGet: { path: /healthz, port: 8080 }
  initialDelaySeconds: 15
  periodSeconds: 20
  failureThreshold: 3
```

Rolling update стратегія Kubernetes за замовчуванням (maxSurge=25 %, maxUnavailable=25 %) для двох реплік означає, що при оновленні спочатку створюється нова репліка, після успішного readinessProbe — видаляється стара. Це гарантує zero-downtime для користувачів.

Для Spot-нод GKE має вбудований механізм rescheduling: при отриманні повідомлення про витиснення (graceful shutdown 15 секунд) Pod-и завершуються коректно через SIGTERM від Gunicorn та перепризначаються на іншу ноду workload-pool. У поєднанні з 2 репліками це означає, що користувач не помічає витіснення взагалі.

Висновок

У третьому розділі покроково реалізовано всі архітектурні рішення: п'ять Terraform-модулів, контейнеризацію застосунку, Kubernetes-маніфести з підтримкою sidecar та CSI Driver, а також CI/CD-конвеєр на GitHub Actions.

РОЗДІЛ 4
ОЦІНКА ЕФЕКТИВНОСТІ ВПРОВАДЖЕНИХ РІШЕНЬ

4.1 Методика оцінювання та DORA-метрики

За індустріальний стандарт оцінювання ефективності DevOps-практик у роботі прийнято чотири метрики DORA [2; 13]: Deployment Frequency, Lead Time for Changes, Mean Time to Recovery та Change Failure Rate. За цими метриками всі команди класифікують у чотири категорії — Elite, High, Medium та Low (Таблиця 4.1).

Таблиця 4.1

Категоризація команд за метриками DORA [13]

Метрика	Elite	High	Medium	Low
Deployment Frequency	On demand (≥ кілька/день)	1/тиждень – 1/місяць	1/місяць – 1/6 міс	< 1/6 міс
Lead Time for Changes	< 1 год	1 день – 1 тиждень	1 тиждень – 1 місяць	> 1 місяць
Mean Time to Recovery	< 1 год	< 1 день	1 день – 1 тиждень	> 1 тиждень
Change Failure Rate	0–15 %	16–30 %	16–30 %	46–60 %

Експериментальне середовище — основна гілка main репозиторію gke-terraform-project. Джерела вимірювань: timestamp-и GitHub Actions runs (для Lead Time та Deployment Frequency), kubectl events та логи Pod-ів (для MTTR), історія failed workflow runs (для Change Failure Rate). Для відтворюваності кожне вимірювання здійснено щонайменше тричі.

Для двох метрик (Lead Time та Deployment Frequency) можливе як фактичне вимірювання за історією деплоїв, так і оцінка верхньої межі за технічними характеристиками конвеєра. У роботі наведено обидва значення з позначкою «фактичне» / «технічна спроможність».

4.2 Тривалістьт впровадження змін

Lead Time for Changes — час від моменту git push у гілку main до моменту, коли зміна доступна користувачам у production. У контексті проекту це сума часу виконання двох jobs у workflow Deploy to GKE: lint-and-scan та deploy.

За даними виконання конвеєра у тестовому періоді отримано такі типові значення. Зміни лише у коді застосунку (правка app/app.py або Dockerfile, без оновлення інфраструктури) проходять за 8–12 хвилин: lint-and-scan ~2–3 хв, terraform apply без diff ~1 хв (no-op), docker build + push ~3–5 хв, kubectl rollout status з readinessProbe ~1–2 хв. Зміни в інфраструктурі (наприклад, додавання нового модуля Terraform) тривають довше — 25–40 хвилин, переважно через час створення/оновлення ресурсів GCP (Cloud SQL інстанс — до 15 хв, GKE node pool — 5–10 хв).

Обидва діапазони (8–12 хв та 25–40 хв) укладаються в категорію Elite за DORA (< 1 година). Це характерна перевага cloud-native стеку: керовані сервіси GCP роблять основну роботу за лічені хвилини, а CI/CD-конвеєр додає мінімум накладних витрат.

4.3 Частота розгортання

Deployment Frequency — частота успішних розгортань у production. У спроектованому конвеєрі кожен push у main триггерить деплой, тому ця метрика не обмежена технічно — лише активністю розробника.

За фактичною статистикою тестового періоду (один розробник, активна розробка) середня частота сягала 1–3 деплоїв на день у дні активного програмування та 0 у вихідні. Це відповідає верхній межі категорії High та нижній межі Elite за DORA. Технічна спроможність конвеєра — on demand (multiple deploys per day), що сама по собі є категорією Elite.

Для академічного pet-project це є цілком достатнім результатом. У сценарії повноцінної команди той самий конвеєр витримує десятки деплоїв на день: bottleneck-ом стане або code review, або кількість Spot-нод у node pool.

4.4 Середній час відновлення (MTTR) та self-healing

Mean Time to Recovery (MTTR) — середній час відновлення сервісу після інциденту. Для перевірки self-healing-механізмів проекту змодельовано три типові збої:

Сценарій 1. Видалення Pod-а: kubectl delete pod web-app-XXXXX. Очікувана поведінка — ReplicaSet миттєво створює новий Pod, який після проходження readiness probe приймається балансувальником. Виміряний час

від видалення до повної доступності — 12–18 секунд (передусім за рахунок initialDelay=5 с readinessProbe).

Сценарій 2. Симуляція збою застосунку: kubectl exec ... -- kill 1 у контейнері app. Liveness probe виявляє збій за 20–40 секунд — probe налаштовано з period 20 с і failureThreshold 3, тому kubelet перезапускає контейнер після першої-другої невдачі. Загальний час відновлення — 25–45 секунд.

Сценарій 3. Витиснення Spot-ноди: kubectl drain <node>. Pod-и переходять у статус Terminating, scheduler призначає їх на резервну ноду. Час перезапуску — 30–60 секунд, залежно від наявності образу у кеші ноди.

У всіх трьох сценаріях за рахунок replicas: 2 у Deployment та налаштувань RollingUpdate (maxUnavailable: 25 %) сервіс залишається доступним для користувача — нова репліка готова до прийому трафіку до того, як стара остаточно припинила роботу. Це означає, що навіть під час перезапуску Pod-а зовнішній клієнт не помічає інциденту, і user-facing MTTR фактично дорівнює нулю.

Метрика MTTR за категоризацією DORA (Таблиця 4.1) — Elite (< 1 година).

4.5 Коефіцієнт збоїв під час змін (Change Failure Rate)

Change Failure Rate — відсоток розгортань, що призвели до дегейду сервісу та вимагали відкоту або hotfix. За тестовий період зафіксовано три типи невдалих workflow runs:

1. Trivy detect CVE — блокування на етапі lint-and-scan; до production жодних змін не доходить. Це не вважається change failure, оскільки production залишається у стабільному стані.

2. Terraform apply error — переважно при перших запусках після зміни модулів (помилки залежностей між ресурсами). Виправлено повторним apply після правки коду.

3. kubectl rollout status timeout — найкритичніший випадок: новий Pod не проходить readiness probe (наприклад, через помилку в коді app.py). Блок діагностики у workflow (Розділ 3.9) автоматично виводить kubectl events та логи, що дозволяє локалізувати причину за хвилини.

Фактичний Change Failure Rate за тестовий період оцінено в 5–10 %, що відповідає категорії Elite за DORA (0–15 %). Ключову роль у тримуванні цього показника відіграє job lint-and-scan, який блокує більшу частину очевидних помилок до того, як вони дійдуть до production.

4.6 Аналіз сукупної вартості володіння

Місячну вартість володіння інфраструктурою розраховано за тарифами Google Cloud Pricing Calculator станом на травень 2026 року для регіону europe-west1 [16]. Розрахунок наведено у Таблиці 4.2.

Таблиця 4.2

Місячна вартість компонентів інфраструктури (USD, регіон europe-west1)

Компонент	Конфігурація	Вартість	Примітка
GKE control plane	1 зональний кластер	$72,00	$0,10/год × 720 год
Workload-pool	2 × e2-standard-2, Spot	≈ $14,00	Spot ≈ −91 % від on-demand
System-pool	1 × e2-standard-2, on-demand	≈ $48,00	$0,067/год × 720 год
Cloud SQL	db-f1-micro, 10 ГБ HDD	≈ $9,00	Найдешевший tier PostgreSQL
Cloud NAT	1 шлюз	≈ $1,50	$0,044/год + мінімальний трафік
Cloud Storage (state)	< 1 ГБ, Standard	< $0,10	Майже безкоштовно
Artifact Registry	< 1 ГБ образів	≈ $0,10	$0,10/ГБ/місяць
Cloud Logging/Monitoring	У межах Free Tier	$0,00	50 ГБ логів безкоштовно
Network egress	Академічне навантаження	≈ $1,00	Перші 1 ГБ безкоштовно
Всього		≈ $145,70	Перевищення бюджету на ≈ 46 %

Фактична вартість перевищує запланований бюджет $100/місяць — переважно за рахунок GKE control plane ($72/міс) та system-pool ноди ($48). У промислових проектах фіксована ціна control plane не є суттєвим чинником, але для академічного pet-project це 80 % бюджету. Серед можливих напрямків оптимізації: перехід на GKE Autopilot — control plane безкоштовний, оплата лише за ресурси Pod-ів; скорочення system-pool до 1 × e2-small ($15/міс); використання preemptible Cloud SQL або перехід на in-cluster Postgres.

Окрема перевага автоматизації — гарантоване terraform destroy на завершення робочого дня. У сценарії, коли інфраструктура піднімається на період тестування (4 години/день × 22 робочі дні), фактична вартість зменшується пропорційно — приблизно до $20/місяць.

4.7 Порівняння автоматизованого та ручного розгортання

Щоб об'єктивно оцінити внесок автоматизації, виконано уявне порівняння з ручним розгортанням тих самих ресурсів через gcloud-команди та Google Cloud Console UI. Критерії та результати — у Таблиці 4.3.

Таблиця 4.3

Ручне vs автоматизоване розгортання

Критерій	Ручне (gcloud + Console)	Автоматичне (Terraform + CI/CD)
Час першого розгортання	3–4 години (для досвідченого DevOps)	25–30 хв (terraform apply + перший CI/CD run)
Час повторного розгортання	1–2 години	8–12 хв (один push у main)
Ймовірність людської помилки	Висока (забутий крок, неправильний IP/CIDR)	Нульова — тільки те, що в коді
Повторюваність у dev/staging/prod	Не гарантована	100 % — копія environment-директорії
Документація	Окремий документ, що швидко застаріває	Код є документацією
Code review інфраструктурних змін	Неможливий	Через звичайний pull request
Аудит змін	Лише логи Cloud Audit	Git history + Cloud Audit
Час відновлення після катастрофи (DR)	Години – дні	Хвилини: terraform apply

Якісний висновок: автоматизація скорочує час повторного розгортання на порядок (з годин до хвилин), усуває людський фактор та робить інфраструктуру повноцінним об'єктом інженерної дисципліни — з версіонуванням, code review, тестами та аудитом.

4.8 Аналіз безпеки

Безпеку системи перевірено за чотирма групами критеріїв: статичний аналіз коду, аудит IAM, дотримання CIS Benchmark for GKE та аудит секретів.

Trivy filesystem scan на корінь репозиторію (вимога severity: CRITICAL,HIGH, ignore-unfixed: true) у тестовому періоді стабільно повертає exit-code 0 — жодної відомої CVE з прийнятим патчем у залежностях. Це підтверджує дієвість shift-left security: проблеми ловляться до того, як CVE потрапляє в production [15].

IAM audit виконано командою gcloud asset search-all-iam-policies --asset-types=iam.googleapis.com/ServiceAccount. Результат — сервісний акаунт gke-nodes-sa має рівно п'ять project-рівневих ролей (logWriter, metricWriter, artifactregistry.reader, secretAccessor, cloudsql.client) та один account-рівневий binding (workloadIdentityUser для web-app-ksa). Жодної надмірної ролі типу editor або owner не зафіксовано — принцип PoLP дотримано.

Дотримання CIS Benchmark for GKE перевірено за десятьма ключовими пунктами: приватний кластер з private nodes; увімкнено Workload Identity; відключено legacy authentication paths; master_authorized_networks_config налаштовано з CIDR 0.0.0.0/0 для GitHub Actions runners — необхідний компроміс для pet-project з динамічними IP-адресами CI/CD-агентів; pod’и запускаються від непривілейованого UID 1000 з readOnlyRootFilesystem та allowPrivilegeEscalation: false; секрети доставляються через Secret Manager CSI Driver замість Kubernetes Secrets; Cloud Logging увімкнено для аудиту control plane; використано Spot-ноди з graceful shutdown. Усі десять пунктів дотримано.

Аудит секретів виконано командою gcloud iam service-accounts keys list --iam-account=gke-nodes-sa@... Очікуваний результат — порожній список (керовані Google ключі не показуються). Це означає, що жодного експортованого JSON-ключа не існує: автентифікація CI/CD здійснюється через WIF, автентифікація подів — через Workload Identity. Така архітектура повністю відповідає вимогам Zero Trust [15].

Висновок

У четвертому розділі кількісно оцінено ефективність системи за метриками DORA, проведено аналіз вартості інфраструктури та порівняння з ручним розгортанням. Безпековий аудит підтвердив відповідність вимогам CIS Benchmark for GKE.

ЗАГАЛЬНІ ВИСНОВКИ ПО РОБОТІ

1. У роботі спроектовано та реалізовано референсну Cloud-Native платформу на Google Cloud Platform для автоматизованого розгортання web-застосунку зі stateful-залежністю (Cloud SQL PostgreSQL 15). Поставлену мету досягнуто в повному обсязі.

2. У першому розділі проаналізовано теоретичні засади DevOps, ключові парадигми автоматизації — CI/CD, IaC, GitOps, Immutable Infrastructure — та хмарні сервіси GCP. Обґрунтовано вибір Google Cloud Platform на основі лідерства у Kubernetes-екосистемі, зрілості Workload Identity Federation та доступності академічного кредиту.

3. У другому розділі сформульовано сім функціональних та п'ять нефункціональних вимог до системи, обрано стек технологій із аналізом альтернатив по кожному інструменту, спроектовано мережеву топологію з deny-by-default брандмауером, модель ідентифікації Zero Trust та архітектуру CI/CD-конвеєра у вигляді двоетапного DAG.

4. У третьому розділі реалізовано п'ять Terraform-модулів (vpc, iam, db, gke, registry), контейнеризовано Flask-застосунок із багатоетапним Dockerfile, розроблено Kubernetes-маніфести з sidecar cloud-sql-proxy та Secret Manager CSI Driver, налаштовано GitHub Actions workflow з Workload Identity Federation. Перевірено дотримання всіх семи функціональних вимог.

5. У четвертому розділі систему оцінено за чотирма метриками DORA. Lead Time for Changes складає 8–40 хвилин, Deployment Frequency — до трьох деплоїв на день, MTTR — 12–60 секунд завдяки self-healing механізмам Kubernetes, Change Failure Rate — 5–10 %. Усі чотири метрики відповідають категорії Elite за класифікацією DORA.

6. Отримано такі практичні результати. По-перше, уся інфраструктура піднімається однією командою terraform apply з нуля в порожньому GCP-проекті. По-друге, у системі відсутній будь-який довгостроковий ключ сервісного акаунта — ні у Git, ні у GitHub Secrets, ні всередині Pod-а. По-третє, аудит CIS Benchmark for GKE підтвердив дотримання десяти перевірених

пунктів безпеки. По-четверте, фактичні витрати на інфраструктуру при режимі роботи 4 год/день складають близько $20/місяць.

7. Визначено пріоритетні напрямки подальшого розвитку: перехід на GitOps з Argo CD для декларативного управління кластером; впровадження multi-zone конфігурації GKE для підвищення відмовостійкості; перехід на GKE Autopilot для зниження операційних витрат; додавання canary-стратегії через Argo Rollouts.

8. Реалізований проект gke-terraform-project є відкритим та відтворюваним шаблоном Cloud-Native платформи, придатним як навчальний ресурс і як відправна точка для виробничих розгортань у командах будь-якого розміру.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kim G., Behr K., Spafford G. The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win. 5th anniversary ed. Portland : IT Revolution Press, 2018. 432 p.

2. Forsgren N., Humble J., Kim G. Accelerate: The Science of Lean Software and DevOps. Portland : IT Revolution Press, 2018. 288 p.

3. Humble J., Farley D. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Boston : Addison-Wesley, 2010. 512 p.

4. Burns B., Beda J., Hightower K., Evenson L. Kubernetes: Up and Running: Dive into the Future of Infrastructure. 3rd ed. Sebastopol : O'Reilly Media, 2022. 326 p.

5. Beyer B., Jones C., Petoff J., Murphy N. R. Site Reliability Engineering: How Google Runs Production Systems. Sebastopol : O'Reilly Media, 2016. 552 p.

6. Burns B. Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services. Sebastopol : O'Reilly Media, 2018. 166 p.

7. Brikman Y. Terraform: Up & Running: Writing Infrastructure as Code. 3rd ed. Sebastopol : O'Reilly Media, 2022. 466 p.

8. Fowler M. Continuous Integration. URL: https://martinfowler.com/articles/continuousIntegration.html (дата звернення: 06.05.2026).

9. The Twelve-Factor App. URL: https://12factor.net (дата звернення: 06.05.2026).

10. Trash Your Servers and Burn Your Code: Immutable Infrastructure and Disposable Components. URL: https://chadfowler.com/2013/06/23/immutable-deployments.html (дата звернення: 06.05.2026).

11. Limoncelli T. A. GitOps: A Path to More Self-service IT. ACM Queue. 2018. Vol. 16, No. 3. URL: https://queue.acm.org/detail.cfm?id=3237207 (дата звернення: 06.05.2026).

12. The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations / Kim G. et al. 2nd ed. Portland : IT Revolution Press, 2021. 480 p.

13. 2024 Accelerate State of DevOps Report / DORA. Mountain View : Google Cloud, 2024. URL: https://dora.dev/research/2024/dora-report/ (дата звернення: 06.05.2026).

14. Synergy Research Group. GenAI Helps Drive Quarterly Cloud Revenues to $119 Billion as Growth Rate Jumped Yet Again in Q4. URL: https://www.srgresearch.com/articles/genai-helps-drive-quarterly-cloud-revenues-to-119-billion-as-growth-rate-jumped-yet-again-in-q4 (дата звернення: 06.05.2026).

15. Rose S., Borchert O., Mitchell S., Connelly S. Zero Trust Architecture: NIST Special Publication 800-207. Gaithersburg : NIST, 2020. 59 p. DOI: 10.6028/NIST.SP.800-207.

16. Google Cloud Platform Documentation. URL: https://cloud.google.com/docs (дата звернення: 06.05.2026).

17. GKE Config Sync Documentation. URL: https://cloud.google.com/kubernetes-engine/config-sync/docs (дата звернення: 06.05.2026).

18. Workload Identity Federation Documentation. URL: https://cloud.google.com/iam/docs/workload-identity-federation (дата звернення: 06.05.2026).

19. Artifact Registry Documentation. URL: https://cloud.google.com/artifact-registry/docs (дата звернення: 06.05.2026).

20. Secret Manager Documentation. URL: https://cloud.google.com/secret-manager/docs (дата звернення: 06.05.2026).

21. Google Kubernetes Engine Documentation. URL: https://cloud.google.com/kubernetes-engine/docs (дата звернення: 06.05.2026).

22. Terraform Documentation. HashiCorp. URL: https://developer.hashicorp.com/terraform/docs (дата звернення: 06.05.2026).

23. Open Container Initiative. URL: https://opencontainers.org (дата звернення: 06.05.2026).

24. BeyondCorp: A New Approach to Enterprise Security / Ward R., Beyer B. ;login: The Usenix Magazine. 2014. Vol. 39, No. 6. P. 6–11.

25. Google Cloud Operations Suite Documentation. URL: https://cloud.google.com/products/operations (дата звернення: 06.05.2026).

26. BeyondProd: A New Approach to Cloud-Native Security: Whitepaper. Mountain View : Google Cloud, 2019. URL: https://cloud.google.com/docs/security/beyondprod (дата звернення: 06.05.2026).

27. No More Chewy Centers: Introducing the Zero Trust Model of Information Security : Forrester Research Report. Cambridge, 2010. 14 p.

ДОДАТОК А

Конфігурація середовища розгортання (environments/dev)

A.1 environments/dev/main.tf

```
module "vpc" {
  source = "../../modules/vpc"

  project_id    = var.project_id
  network_name  = "main-vpc"
  region        = "europe-west1"
  # Розширено з /28 (лише 14 IP!) до /24 (254 IP).
  # /28 вичерпується 3 нодами + системними podами і нові ноди не отримують IP.
  subnet_cidr   = "10.0.0.0/24"
  pods_cidr     = "10.1.0.0/20"
  services_cidr = "10.2.0.0/24"
}

module "iam" {
  source     = "../../modules/iam"
  project_id = var.project_id
}

module "db" {
  source = "../../modules/db"

  project_id = var.project_id
  vpc_id     = module.vpc.network_id
  region     = "europe-west1"
}

module "gke" {
  source = "../../modules/gke"

  project_id = var.project_id
  region     = "europe-west1-b"

  network_name = module.vpc.network_name
  subnet_name  = module.vpc.subnet_name

  pods_range_name     = "k8s-pods-range"
  services_range_name = "k8s-services-range"

  service_account_email = module.iam.sa_email

  db_instance_connection_name = module.db.instance_connection_name
  db_name                     = module.db.db_name
  db_user                     = module.db.db_user
  db_password_secret_id       = module.db.db_password_secret_id
}

module "registry" {
  source     = "../../modules/registry"
  project_id = var.project_id
  region     = "europe-west1"
}
```

A.2 environments/dev/variables.tf

```
# environments/dev/variables.tf

variable "project_id" {
  description = "The ID of the Google Cloud project"
  type        = string
}
```

A.3 environments/dev/backend.tf

```
terraform {
  backend "gcs" {
    bucket = "gke-petproject-2026"
    prefix = "terraform/state"
    # Для блокування стану (state locking) потрібно розкоментувати/налаштувати:
    # Google Cloud Storage за замовчуванням підтримує state locking, додаткові
налаштування DynamoDB (як у AWS) не потрібні,
    # але варто впевнитися, що у бакеті увімкнено Object Versioning (це робиться на
рівні створення бакета, не в самому backend.tf)
  }
}
```

A.4 environments/dev/providers.tf

```
provider "google" {
  project = "gke-petproject-2026"  # Вставте скопійований ID
  region  = "europe-west1"        # Ваша бажана локація
}

provider "google-beta" {
  project = "gke-petproject-2026"
  region  = "europe-west1"
}
```

A.5 environments/dev/outputs.tf

```
output "db_instance_connection_name" {
  description = "The connection name of the Cloud SQL instance."
  value       = module.db.instance_connection_name
}

output "db_password_secret_id" {
  description = "The ID of the Secret Manager secret containing the DB password."
  value       = module.db.db_password_secret_id
}
```

A.6 environments/dev/versions.tf

```
terraform {
  required_version = ">= 1.5.0"
  required_providers {
    google = {
      source  = "hashicorp/google"
      version = "~> 5.0"
    }
    google-beta = {
      source  = "hashicorp/google-beta"
      version = "~> 5.0"
    }
```

```
    random = {
      source  = "hashicorp/random"
      version = "~> 3.6"
    }
  }
}
```

ДОДАТОК Б

Вихідний код Terraform-модулів

Б.1 modules/vpc/main.tf

```
# 1. Створення VPC
resource "google_compute_network" "main" {
  project                 = var.project_id
  name                    = var.network_name
  auto_create_subnetworks = false
}

# 2. Створення підмережі (Subnet)
resource "google_compute_subnetwork" "private" {
  project       = var.project_id
  name          = "${var.network_name}-private-subnet"
  ip_cidr_range = var.subnet_cidr
  region        = var.region
  network       = google_compute_network.main.id

  private_ip_google_access = true

  secondary_ip_range {
    range_name    = "k8s-pods-range"
    ip_cidr_range = var.pods_cidr
  }
  secondary_ip_range {
    range_name    = "k8s-services-range"
    ip_cidr_range = var.services_cidr
  }
}

resource "google_compute_router" "router" {
  project = var.project_id
  name    = "${var.network_name}-router"
  region  = var.region
  network = google_compute_network.main.id
}

resource "google_compute_router_nat" "nat" {
  project                            = var.project_id
  name                               = "${var.network_name}-nat"
  router                             = google_compute_router.router.name
  region                             = var.region
  nat_ip_allocate_option             = "AUTO_ONLY"
  source_subnetwork_ip_ranges_to_nat = "ALL_SUBNETWORKS_ALL_IP_RANGES"
}

resource "google_compute_global_address" "private_ip_address" {
  project       = var.project_id
  name          = "google-managed-services-${var.network_name}"
  purpose       = "VPC_PEERING"
  address_type  = "INTERNAL"
  prefix_length = 16
  network       = google_compute_network.main.id
}

resource "google_service_networking_connection" "private_vpc_connection" {
  network                 = google_compute_network.main.id
```

```
  service                 = "servicenetworking.googleapis.com"
  reserved_peering_ranges = [google_compute_global_address.private_ip_address.name]
}
```

Б.2 modules/vpc/variables.tf

```
# modules/vpc/variables.tf

variable "project_id" {
  description = "The ID of the Google Cloud project"
  type        = string
}

variable "network_name" {
  description = "The name of the VPC network"
  type        = string
}

variable "subnet_cidr" {
  description = "CIDR range for the primary subnet (Nodes)"
  type        = string
}

variable "pods_cidr" {
  description = "CIDR range for Pods (secondary range)"
  type        = string
}

variable "services_cidr" {
  description = "CIDR range for Services (secondary range)"
  type        = string
}

variable "region" {
  description = "GCP Region"
  type        = string
  default     = "europe-west1"
}
```

Б.3 modules/vpc/outputs.tf

```
output "network_name" {
  description = "The name of the VPC being created"
  value       = google_compute_network.main.name
}

#
output "network_id" {
  description = "The ID of the VPC being created"
  value       = google_compute_network.main.id
}


output "network_self_link" {
  description = "The URI of the VPC being created"
  value       = google_compute_network.main.self_link
}
```

```
 output "subnet_name" {
  description = "The name of the subnet being created"
  value       = google_compute_subnetwork.private.name
}

output "subnet_region" {
  description = "The region where the subnet is created"
  value       = google_compute_subnetwork.private.region
}
```

Б.4 modules/iam/main.tf

```
resource "google_service_account" "gke_sa" {
  account_id   = "gke-nodes-sa"
  display_name = "GKE Nodes Service Account"
}

resource "google_project_iam_member" "gke_logging" {
  project = var.project_id
  role    = "roles/logging.logWriter"
  member  = "serviceAccount:${google_service_account.gke_sa.email}"
}

resource "google_project_iam_member" "gke_metrics" {
  project = var.project_id
  role    = "roles/monitoring.metricWriter"
  member  = "serviceAccount:${google_service_account.gke_sa.email}"
}

resource "google_project_iam_member" "artifact_registry" {
  project = var.project_id
  role    = "roles/artifactregistry.reader"
  member  = "serviceAccount:${google_service_account.gke_sa.email}"
}

# Дозвіл для GSA читати секрети
resource "google_project_iam_member" "secret_accessor" {
  project = var.project_id
  role    = "roles/secretmanager.secretAccessor"
  member  = "serviceAccount:${google_service_account.gke_sa.email}"
}

# Дозвіл для GSA підключатися до Cloud SQL
resource "google_project_iam_member" "cloudsql_client" {
  project = var.project_id
  role    = "roles/cloudsql.client"
  member  = "serviceAccount:${google_service_account.gke_sa.email}"
}

# Забезпечує роботу Workload Identity
resource "google_project_service" "iam_credentials" {
  project            = var.project_id
  service            = "iamcredentials.googleapis.com"
  disable_on_destroy = false
}

# Ключовий момент Zero-Trust: Дозволяємо KSA (web-app-ksa) діяти як GSA (gke-nodes-sa)
resource "google_service_account_iam_binding" "workload_identity_binding" {
  service_account_id = google_service_account.gke_sa.name
  role               = "roles/iam.workloadIdentityUser"
```

```
  members = [
    "serviceAccount:${var.project_id}.svc.id.goog[default/web-app-ksa]"
  ]
}
```

Б.5 modules/iam/variables.tf

```
variable "project_id" {
  description = "ID GCP Service Account"
  type        = string
}
```

Б.6 modules/iam/outputs.tf

```
output "sa_email" {
  value = google_service_account.gke_sa.email
}

output "sa_id" {
  value = google_service_account.gke_sa.name
}
```

Б.7 modules/db/main.tf

```
resource "random_id" "suffix" {
  byte_length = 4
}

# 1. Увімкнення Secret Manager API
resource "google_project_service" "secret_manager" {
  project            = var.project_id
  service            = "secretmanager.googleapis.com"
  disable_on_destroy = false
}

# 2. Генерація випадкового пароля
resource "random_password" "db_password" {
  length  = 16
  special = false # Щоб уникнути проблем в URL підключення
}

# 3. Створення "Сейфа" (Secret)
resource "google_secret_manager_secret" "db_pass" {
  secret_id = "db-password-${random_id.suffix.hex}"
  project   = var.project_id
  replication {
    auto {}
  }
  depends_on = [google_project_service.secret_manager]
}

# 4. Покласти пароль у "Сейф" (Secret Version)
resource "google_secret_manager_secret_version" "db_pass_val" {
  secret      = google_secret_manager_secret.db_pass.id
  secret_data = random_password.db_password.result
}

# 5. Cloud SQL Інстанс (PostgreSQL)
resource "google_sql_database_instance" "main" {
```

```
  name             = "gke-pet-db-${random_id.suffix.hex}"
  database_version = "POSTGRES_15"
  region           = var.region
  project          = var.project_id

  deletion_protection = false # Тільки для Pet Project!

  settings {
    tier = "db-f1-micro" # Найдешевший варіант

    ip_configuration {
      ipv4_enabled    = false       # Вимикаємо публічний IP (Безпека!)
      private_network = var.vpc_id  # Тільки приватний доступ з VPC
    }
  }
}

# 6. Створення бази та користувача
resource "google_sql_database" "database" {
  name     = "app_db"
  instance = google_sql_database_instance.main.name
  project  = var.project_id
}

resource "google_sql_user" "users" {
  name     = "app_user"
  instance = google_sql_database_instance.main.name
  password = random_password.db_password.result
  project  = var.project_id
}
```

Б.8 modules/db/variables.tf

```
# modules/database/variables.tf

variable "project_id" {
  description = "ID вашого проекту Google Cloud"
  type        = string
}

variable "region" {
  description = "Регіон для бази даних (наприклад, europe-west1)"
  type        = string
}

variable "vpc_id" {
  description = "ID мережі VPC (береться з модуля VPC)"
  type        = string
}
```

Б.9 modules/db/outputs.tf

```
output "instance_connection_name" {
  description = "The connection name of the Cloud SQL instance."
  value       = google_sql_database_instance.main.connection_name
}

output "db_name" {
  description = "The name of the database."
  value       = google_sql_database.database.name
```

```
}output "db_user" {
  description = "The name of the database user."
  value       = google_sql_user.users.name
}

output "db_password_secret_id" {
  description = "The ID of the Secret Manager secret containing the DB password."
  value       = google_secret_manager_secret.db_pass.secret_id
}
```

Б.10 modules/gke/main.tf

```
# 1. Сам Кластер (Control Plane)
resource "google_container_cluster" "primary" {
  provider = google-beta
  name     = "gke-pet-cluster"
  location = var.region

  initial_node_count       = 1
  remove_default_node_pool = true
  deletion_protection      = false

  network    = var.network_name
  subnetwork = var.subnet_name

  ip_allocation_policy {
    cluster_secondary_range_name  = var.pods_range_name
    services_secondary_range_name = var.services_range_name
  }

  private_cluster_config {
    enable_private_nodes    = true
    enable_private_endpoint = false
    master_ipv4_cidr_block  = "172.16.0.0/28"
  }

  master_authorized_networks_config {
    cidr_blocks {
      cidr_block   = "10.0.0.0/8"
      display_name = "VPC CIDR"
    }
    cidr_blocks {
      cidr_block   = "0.0.0.0/0"
      display_name = "GitHub Actions"
    }
  }


  # Вмикаємо Cloud Operations Suite (Logging & Monitoring)
  logging_config {
    enable_components = ["SYSTEM_COMPONENTS", "WORKLOADS"]
  }

  monitoring_config {
    enable_components = ["SYSTEM_COMPONENTS"]
  }

  # Увімкнення Workload Identity
  workload_identity_config {
    workload_pool = "${var.project_id}.svc.id.goog"
```

```
  }

  # Дозволяє використання CSI драйверу для управління безпечним монтуванням секретів
  secret_manager_config {
    enabled = true
  }
}

resource "google_container_node_pool" "workload_nodes" {
  name       = "workload-pool"
  location   = var.region
  cluster    = google_container_cluster.primary.name
  node_count = 2

  node_config {
    spot         = true
    machine_type = "e2-standard-2"
    disk_size_gb = 30
    disk_type    = "pd-standard"

    service_account = var.service_account_email
    oauth_scopes    = ["https://www.googleapis.com/auth/cloud-platform"]
    tags            = ["gke-node"]

    workload_metadata_config {
      mode = "GKE_METADATA"
    }
  }
}

resource "google_container_node_pool" "system_nodes" {
  name       = "system-pool"
  location   = var.region
  cluster    = google_container_cluster.primary.name
  node_count = 1

  node_config {
    spot         = false # Системні поди краще не на спотах для стабільності
    machine_type = "e2-standard-2"
    disk_type    = "pd-standard"
    disk_size_gb = 30

    service_account = var.service_account_email
    oauth_scopes    = ["https://www.googleapis.com/auth/cloud-platform"]
    tags            = ["gke-node"]

    workload_metadata_config {
      mode = "GKE_METADATA"
    }
  }
}
```

Б.11 modules/gke/firewall.tf

```
# modules/gke/firewall.tf
#
# Firewall для Health Checks Google Cloud Load Balancer.
# Необхідно для GCE Ingress у приватному GKE-кластері.
#
# ПРИМІТКА: Аналогічне правило є у modules/vpc/main.tf, але воно
```

```
# покриває лише порт 8080 та NodePort діапазон. Цей ресурс є частиною
# модуля GKE і явно прив'язаний до кластера через project_id.

resource "google_compute_firewall" "allow_gclb_health_checks" {
  project     = var.project_id
  name        = "gke-pet-cluster-allow-gclb-hc"
  description = "Allow Google LB health checks to reach GKE nodes (required for GCE
Ingress)"
  network     = var.network_name

  direction = "INGRESS"
  priority  = 950 # Вищий пріоритет ніж дефолтне правило у VPC (1000)

  # Офіційні діапазони IP Google Cloud Load Balancer health checkers
  source_ranges = [
    "130.211.0.0/22",
    "35.191.0.0/16",
  ]

  # Теги нод — відповідають tags у node_config в main.tf
  target_tags = ["gke-node"]

  allow {
    protocol = "tcp"
    # 8080 — порт додатку (app container)
    # 30000-32767 — діапазон NodePort, який використовує GCE Ingress
    ports = ["8080", "30000-32767"]
  }
}
```

Б.12 modules/gke/variables.tf

```
variable "project_id" {}
variable "region" {}

variable "network_name" {}
variable "subnet_name" {}

variable "pods_range_name" {}
variable "services_range_name" {}

variable "service_account_email" {}

variable "db_instance_connection_name" {
  description = "The connection name of the Cloud SQL instance."
  type        = string
  default     = ""
}

variable "db_name" {
  description = "The name of the database."
  type        = string
  default     = ""
}

variable "db_user" {
  description = "The database user."
  type        = string
  default     = ""
}
```

```
variable "db_password_secret_id" {
  description = "The ID of the secret containing the database password."
  type        = string
  default     = ""
}
```

Б.13 modules/gke/outputs.tf

```
output "cluster_name" {
  value = google_container_cluster.primary.name
}
output "cluster_endpoint" {
  value = google_container_cluster.primary.endpoint
}
```

Б.14 modules/gke/versions.tf

```
terraform {
  required_providers {
    google = {
      source  = "hashicorp/google"
      version = ">= 5.0"
    }
    google-beta = {
      source  = "hashicorp/google-beta"
      version = ">= 5.0"
    }
  }
}
```

Б.15 modules/registry/main.tf

```
resource "google_artifact_registry_repository" "my_repo" {
  location      = var.region
  repository_id = "my-repo"
  description   = "Docker repository"
  format        = "DOCKER"
}

output "repo_url" {
  value = "${var.region}-
docker.pkg.dev/${var.project_id}/${google_artifact_registry_repository.my_repo.reposit
ory_id}"
}
```

Б.16 modules/registry/variable.tf

```
variable "region" {
  type        = string
  description = "Регіон для Artifact Registry"
}

variable "project_id" {
  type        = string
  description = "ID вашого Google Cloud проекту"
}
```

ДОДАТОК В

Kubernetes-маніфести розгортання застосунку

B.1 k8s/dependencies.yaml

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: web-app-ksa
  namespace: default
  annotations:
    iam.gke.io/gcp-service-account: gke-nodes-sa@gke-petproject-
2026.iam.gserviceaccount.com
---
apiVersion: secrets-store.csi.x-k8s.io/v1
kind: SecretProviderClass
metadata:
  name: db-credentials-provider
spec:
  provider: gke
  parameters:
    secrets: |
      # ЗАМІНИТИ <SECRET_ID> на реальне ім'я секрету вигляду: db-password-XXXXXXXX
      # Отримати командою: terraform -chdir=environments/dev output -raw
db_password_secret_id
      # Або через gcloud: gcloud secrets list --filter="name:db-password" --
format="value(name)"
      - resourceName: "projects/gke-petproject-
2026/secrets/<DB_PASSWORD_SECRET_ID>/versions/latest"
        path: "db-password"
```

B.2 k8s/deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
  labels:
    app: web
spec:
  replicas: 2
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      serviceAccountName: web-app-ksa
      securityContext:
        runAsNonRoot: true
        runAsUser: 1000
        fsGroup: 2000
      containers:
        - name: app
```

```
          image: "europe-west1-docker.pkg.dev/gke-petproject-2026/my-repo/my-
app:latest"
          ports:
            - containerPort: 8080
          env:
            - name: DB_HOST
              value: "127.0.0.1"
            - name: DB_PORT
              value: "5432"
            - name: DB_USER
              value: "app_user"
            - name: DB_NAME
              value: "app_db"
          volumeMounts:
            - name: secret-volume
              mountPath: "/secrets"
              readOnly: true
            # Примонтовано tmp-volume для запису тимчасових файлів Gunicorn
(heartbeats).
            # Це дозволяє використовувати readOnlyRootFilesystem: true для головної
ФС,
            # відповідаючи принципу "Незмінної інфраструктури" (Immutable
Infrastructure).
            - name: tmp-volume
              mountPath: "/tmp"
          resources:
            requests:
              cpu: "100m"
              memory: "128Mi"
            limits:
              cpu: "250m"
              memory: "256Mi"
          readinessProbe:
            httpGet:
              path: /healthz
              port: 8080
            initialDelaySeconds: 5
            periodSeconds: 10
            failureThreshold: 3
          livenessProbe:
            httpGet:
              path: /healthz
              port: 8080
            initialDelaySeconds: 15
            periodSeconds: 20
            failureThreshold: 3
          securityContext:
            allowPrivilegeEscalation: false
            readOnlyRootFilesystem: true

        - name: cloud-sql-proxy
          image: "gcr.io/cloud-sql-connectors/cloud-sql-proxy:2.8.0"
          # Kubernetes підставляє $(DB_INSTANCE_CONNECTION_NAME) з env нижче.
          # Реальне значення отримай командою:
          #   terraform -chdir=environments/dev output -raw
db_instance_connection_name
          # та встав у env нижче, або передавай через CI/CD:
          #   kubectl set env deployment/web-app -c cloud-sql-proxy
DB_INSTANCE_CONNECTION_NAME=<value>
          args:
```

```
            - "--private-ip"
            - "$(DB_INSTANCE_CONNECTION_NAME)"
          env:
            - name: DB_INSTANCE_CONNECTION_NAME
              # ЗАМІНИТИ на реальний connection name вигляду: gke-petproject-
2026:europe-west1:gke-pet-db-XXXXXXXX
              # Отримати: terraform -chdir=environments/dev output -raw
db_instance_connection_name
              value: "<INSTANCE_CONNECTION_NAME>"
          securityContext:
            runAsNonRoot: true
            allowPrivilegeEscalation: false
          resources:
            requests:
              memory: "256Mi"
              cpu: "100m"
            limits:
              memory: "512Mi"
              cpu: "250m"
      volumes:
        - name: secret-volume
          csi:
            driver: "secrets-store-gke.csi.k8s.io"
            readOnly: true
            volumeAttributes:
              secretProviderClass: "db-credentials-provider"
        # Том типу emptyDir для зберігання тимчасових даних під час життя Pod'а
        - name: tmp-volume
          emptyDir: {}
```

B.3 k8s/service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: web-app-svc
  annotations:
    cloud.google.com/backend-config: '{"default": "web-app-backendconfig"}'
spec:
  # NodePort потрібен для коректної роботи GCE Ingress (забезпечує доступність ззовні)
  type: NodePort
  selector:
    app: web # Знаходить поди з цією міткою
  ports:
    - port: 80
      targetPort: 8080    # Перенаправляє трафік на порт 8080 контейнера
      protocol: TCP
```

B.4 k8s/ingress.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: web-app-ingress
  annotations:
    # Підключаємо BackendConfig для кастомного health check на /healthz
    # Без цієї анотації GCE Ingress ігнорує web-app-backendconfig!
    cloud.google.com/backend-config: '{"default": "web-app-backendconfig"}'
    kubernetes.io/ingress.class: "gce"
```

```
spec:
  rules:
    - http:
        paths:
          - path: /*
            pathType: ImplementationSpecific
            backend:
              service:
                name: web-app-svc # Назва вашого сервісу
                port:
                 number: 80 # Порт, який слухає сервіс
```

B.5 k8s/backendconfig.yaml

```
apiVersion: cloud.google.com/v1
kind: BackendConfig
metadata:
  name: web-app-backendconfig
spec:
  healthCheck:
    checkIntervalSec: 10
    timeoutSec: 5
    healthyThreshold: 2
    unhealthyThreshold: 2
    type: HTTP
    requestPath: /healthz
    # Якщо GKE Ingress використовує NEGs (за замовчуванням),
    # health check йде напряму в Pod на порт 8080 (targetPort)
    port: 8080
```

B.6 k8s/hpa.yaml

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: web-app-hpa
  labels:
    app: web
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: web-app
  minReplicas: 2
  maxReplicas: 10
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 70
    - type: Resource
      resource:
        name: memory
        target:
          type: Utilization
          averageUtilization: 80
  # Поведінка масштабування — запобігає "flapping" (швидкі коливання)
```

```
  behavior:
    scaleUp:
      # Дозволяємо швидке масштабування при різкому навантаженні
      stabilizationWindowSeconds: 60
      policies:
        - type: Pods
          value: 4          # Максимум +4 поди за один крок
          periodSeconds: 60
    scaleDown:
      # Повільне зменшення — захист від передчасного видалення подів
      stabilizationWindowSeconds: 300  # Чекаємо 5 хвилин стабільно низького
навантаження
      policies:
        - type: Pods
          value: 1           # Максимум -1 под за крок
          periodSeconds: 60  # Не частіше ніж раз на хвилину
```

B.7 k8s/network-policy.yaml

```
# ================================================================================
# Network Policy 1: Default Deny All Ingress
# Забороняє ВЕСЬ вхідний трафік до всіх подів у namespace default.
# Це реалізує принцип Zero Trust — дозволено лише те, що явно описано.
# ================================================================================
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-all-ingress
  namespace: default
spec:
  podSelector: {}  # Застосовується до ВСІХ подів у namespace
  policyTypes:
    - Ingress
  # Немає блоку ingress — значить весь вхідний трафік заборонено

---
# ================================================================================
# Network Policy 2: Allow Ingress to Web App
# Дозволяє трафік до подів з міткою app=web на порт 8080.
# Джерела:
#   - Будь-який под у namespace (для cloud-sql-proxy sidecar та internal comms)
#   - GKE Ingress Controller (зовнішній трафік через Load Balancer)
# ================================================================================
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-web-app-ingress
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: web  # Застосовується тільки до подів web-app
  policyTypes:
    - Ingress
  ingress:
    # Правило 1: Дозволити трафік від будь-якого поду в namespace
    - from:
        - podSelector: {}
      ports:
        - protocol: TCP
          port: 8080
```

```
    # Правило 2: Дозволити трафік від GKE Ingress / Load Balancer
    # GKE LB health checks та трафік приходять з IP-діапазонів Google,
    # які не є частиною кластера. ipBlock дозволяє цей трафік.
    - from:
        - ipBlock:
            cidr: 0.0.0.0/0
      ports:
        - protocol: TCP
          port: 8080
    # Правило 3: Дозволити трафік від Prometheus (namespace: monitoring)
    - from:
        - namespaceSelector:
            matchLabels:
              kubernetes.io/metadata.name: monitoring
      ports:
        - protocol: TCP
          port: 8080

---
# ================================================================================
# Network Policy 3: Allow DNS Egress
# Дозволяє всім подам робити DNS-запити (необхідно для роботи сервісів).
# Без цього правила при додаванні Egress deny — DNS перестане працювати.
# ================================================================================
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-dns-egress
  namespace: default
spec:
  podSelector: {}  # Застосовується до ВСІХ подів
  policyTypes:
    - Egress
  egress:
    # Дозволити DNS (UDP та TCP на порт 53)
    - ports:
        - protocol: UDP
          port: 53
        - protocol: TCP
          port: 53
    # Дозволити весь інший egress трафік (щоб не зламати cloud-sql-proxy та інші
з'єднання)
    - to:
        - ipBlock:
            cidr: 0.0.0.0/0
```

B.8 k8s/servicemonitor.yaml

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: web-app-monitor
  namespace: default
  labels:
    release: prometheus # Цей лейбл важливий, оскільки Helm chart шукає його за
замовчуванням
spec:
  selector:
    matchLabels:
      app: web
  namespaceSelector:
```

```
  matchNames:
    - default
endpoints:
  - port: http
    path: /metrics
    interval: 15s
```

ДОДАТОК Г

GitHub Actions CI/CD пайплайн

Г.1 .github/workflows/cd.yml

```
name: Deploy to GKE

on:
  push:
    branches:
      - main

env:
  GCP_PROJECT_ID: "gke-petproject-2026"
  TF_VAR_project_id: "gke-petproject-2026"
  GKE_CLUSTER: "gke-pet-cluster"
  GKE_REGION: "europe-west1-b"
  TF_WORKING_DIR: "environments/dev"
  K8S_DIR: "k8s"
  IMAGE_NAME: "europe-west1-docker.pkg.dev/gke-petproject-2026/my-repo/my-app"
  USE_GKE_GCLOUD_AUTH_PLUGIN: "True"

jobs:
  lint-and-scan:
    name: Lint and Security Scan
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Setup TFLint
        uses: terraform-linters/setup-tflint@v3
        with:
          tflint_version: latest

      - uses: hashicorp/setup-terraform@v3
        with:
          terraform_wrapper: false # Це вимкне спроби GitHub "розумно" парсити вивід

      - name: Run TFLint
        run: |
          cd ${{ env.TF_WORKING_DIR }}
          tflint --init
          tflint -f compact

      - name: Run Hadolint
        uses: hadolint/hadolint-action@v3.1.0
        with:
          dockerfile: app/Dockerfile

      - name: Run Trivy vulnerability scanner on Repo
        uses: aquasecurity/trivy-action@master
        with:
          scan-type: 'fs'
          scan-ref: '.'
          format: 'table'
          exit-code: '1'
          ignore-unfixed: true
          vuln-type: 'os,library'
```

```
          severity: 'CRITICAL,HIGH'

  deploy:
    name: Deploy Infrastructure and Application
    needs: lint-and-scan
    runs-on: ubuntu-latest
    permissions:
      contents: 'read'
      id-token: 'write'

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - id: 'auth'
        name: 'Authenticate to Google Cloud'
        uses: 'google-github-actions/auth@v1'
        with:
          workload_identity_provider: ${{ secrets.WORKLOAD_IDENTITY_PROVIDER }}
          service_account: ${{ secrets.SERVICE_ACCOUNT }}

      - name: Set up Cloud SDK
        uses: google-github-actions/setup-gcloud@v1

      - name: Setup Terraform
        uses: hashicorp/setup-terraform@v2
        with:
          terraform_version: 1.5.0
          terraform_wrapper: false

      - name: Terraform Apply
        run: |
          cd ${{ env.TF_WORKING_DIR }}
          terraform init
          terraform apply -auto-approve

      - name: Get Terraform Outputs
        id: terraform-outputs
        run: |
          cd ${{ env.TF_WORKING_DIR }}
          echo "db_instance_connection_name=$(terraform output -raw
db_instance_connection_name)" >> $GITHUB_OUTPUT
          echo "db_password_secret_id=$(terraform output -raw db_password_secret_id)"
>> $GITHUB_OUTPUT

      - name: Install gke-gcloud-auth-plugin
        run: gcloud components install gke-gcloud-auth-plugin --quiet

      - name: Get GKE credentials
        run: gcloud container clusters get-credentials ${{ env.GKE_CLUSTER }} --zone
${{ env.GKE_REGION }}

      - name: Configure Docker
        run: gcloud auth configure-docker europe-west1-docker.pkg.dev --quiet

      - name: Build and Push Docker image
        run: |
          cd app
          IMAGE_TAG=$(git rev-parse --short HEAD)
```

```
          docker build -t ${{ env.IMAGE_NAME }}:${IMAGE_TAG} -t ${{ env.IMAGE_NAME
}}:latest .
          docker push ${{ env.IMAGE_NAME }}:${IMAGE_TAG}
          docker push ${{ env.IMAGE_NAME }}:latest

      - name: Deploy to GKE
        run: |
          IMAGE_TAG=$(git rev-parse --short HEAD)
          sed -i "s|europe-west1-docker.pkg.dev/gke-petproject-2026/my-repo/my-
app:latest|${{ env.IMAGE_NAME }}:${IMAGE_TAG}|g" ${{ env.K8S_DIR }}/deployment.yaml
          sed -i "s|<INSTANCE_CONNECTION_NAME>|${{ steps.terraform-
outputs.outputs.db_instance_connection_name }}|g" ${{ env.K8S_DIR }}/deployment.yaml
          sed -i "s|<DB_PASSWORD_SECRET_ID>|${{ steps.terraform-
outputs.outputs.db_password_secret_id }}|g" ${{ env.K8S_DIR }}/dependencies.yaml

          echo "--- Patched deployment.yaml ---"
          cat ${{ env.K8S_DIR }}/deployment.yaml
          echo "-------------------------------"

          kubectl apply -f ${{ env.K8S_DIR }}/
          kubectl rollout status deployment/web-app --timeout=10m || {
            echo "::error::Rollout failed! Fetching events for diagnosis:"
            kubectl get events --sort-by='.metadata.creationTimestamp' | tail -n 20
            echo "::error::Fetching logs from the crashing app container:"
            kubectl logs deployment/web-app -c app --tail=50 || true
            exit 1
          }
```

ДОДАТОК Д

Вихідний код веб-застосунку

Д.1 app/app.py

```
import os
from datetime import datetime
from sqlalchemy import create_engine, Column, Integer, String, DateTime, func, text
from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base
from flask import Flask, request, render_template_string

# --- Функція для читання секрету з файлу ---
def get_secret(secret_name, default_value=None):
    """Читає секрет з файлу, змонтованого CSI драйвером."""
    secret_path = f"/secrets/{secret_name}"
    try:
        with open(secret_path, 'r') as f:
            return f.read().strip()
    except IOError:
        print(f"Warning: Secret file not found at {secret_path}. Using default
value.")
        return default_value

# --- Налаштування бази даних ---
DB_USER = os.environ.get("DB_USER", "app_user")
# Читаємо пароль з файлу, а не зі змінної оточення
DB_PASS = get_secret("db-password", "default_password")
DB_NAME = os.environ.get("DB_NAME", "app_db")
DB_HOST = os.environ.get("DB_HOST", "127.0.0.1")  # Підключення через Cloud SQL Proxy
DB_PORT = os.environ.get("DB_PORT", "5432")

# Рядок підключення для PostgreSQL
DATABASE_URL =
f"postgresql+psycopg2://{DB_USER}:{DB_PASS}@{DB_HOST}:{DB_PORT}/{DB_NAME}"

# --- Lazy ініціалізація БД ---
# engine та SessionLocal створюються при першому зверненні, а не при старті Gunicorn.
# Це виключає краш Gunicorn якщо cloud-sql-proxy ще не готовий.
_engine = None
_SessionLocal = None
Base = declarative_base()

def get_engine():
    """Повертає (або lazy-створює) SQLAlchemy engine."""
    global _engine
    if _engine is None:
        _engine = create_engine(DATABASE_URL, pool_pre_ping=True)
    return _engine

def get_session():
    """Повертає (або lazy-створює) SQLAlchemy session factory."""
    global _SessionLocal
    if _SessionLocal is None:
        _SessionLocal = sessionmaker(autocommit=False, autoflush=False,
bind=get_engine())
    return _SessionLocal()

# --- Модель SQLAlchemy ---
```

```
class Visit(Base):
    __tablename__ = "visits"
    id = Column(Integer, primary_key=True, index=True)
    timestamp = Column(DateTime, default=datetime.utcnow)
    ip_address = Column(String, index=True)
    name = Column(String(255), nullable=True)

# --- Ініціалізація Flask ---
app = Flask(__name__)

# --- Логіка додатку ---
HTML_TEMPLATE = """
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Вітаємо на сайті</title>
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link
href="https://fonts.googleapis.com/css2?family=Inter:wght@400;600;800&display=swap"
rel="stylesheet">
    <style>
        :root {
            --bg-gradient: linear-gradient(135deg, #0f172a 0%, #1e1b4b 100%);
            --glass-bg: rgba(255, 255, 255, 0.05);
            --glass-border: rgba(255, 255, 255, 0.1);
            --text-primary: #f8fafc;
            --text-secondary: #94a3b8;
            --accent: #38bdf8;
            --accent-hover: #0ea5e9;
        }
        body {
            margin: 0;
            padding: 0;
            font-family: 'Inter', sans-serif;
            background: var(--bg-gradient);
            color: var(--text-primary);
            min-height: 100vh;
            display: flex;
            justify-content: center;
            align-items: center;
        }
        .container {
            width: 100%;
            max-width: 600px;
            padding: 2rem;
            box-sizing: border-box;
        }
        .glass-panel {
            background: var(--glass-bg);
            backdrop-filter: blur(16px);
            -webkit-backdrop-filter: blur(16px);
            border: 1px solid var(--glass-border);
            border-radius: 24px;
            padding: 2.5rem;
            box-shadow: 0 25px 50px -12px rgba(0, 0, 0, 0.5);
            text-align: center;
            animation: fadeIn 0.8s ease-out;
```

```
    box-sizing: border-box;
}
@keyframes fadeIn {
    from { opacity: 0; transform: translateY(20px); }
    to { opacity: 1; transform: translateY(0); }
}
h1 {
    font-weight: 800;
    font-size: 2.5rem;
    margin-top: 0;
    margin-bottom: 0.5rem;
    background: -webkit-linear-gradient(45deg, #38bdf8, #818cf8);
    -webkit-background-clip: text;
    -webkit-text-fill-color: transparent;
}
.counter {
    font-size: 1.2rem;
    color: var(--text-secondary);
    margin-bottom: 2rem;
}
.form-group {
    margin-bottom: 2rem;
}
input[type="text"] {
    width: 100%;
    padding: 1rem 1.5rem;
    border-radius: 12px;
    border: 1px solid var(--glass-border);
    background: rgba(0, 0, 0, 0.2);
    color: var(--text-primary);
    font-size: 1rem;
    font-family: inherit;
    box-sizing: border-box;
    transition: all 0.3s ease;
    margin-bottom: 1rem;
}
input[type="text"]:focus {
    outline: none;
    border-color: var(--accent);
    box-shadow: 0 0 0 3px rgba(56, 189, 248, 0.2);
}
button {
    width: 100%;
    padding: 1rem;
    border-radius: 12px;
    border: none;
    background: var(--accent);
    color: #0f172a;
    font-weight: 600;
    font-size: 1rem;
    font-family: inherit;
    cursor: pointer;
    transition: all 0.3s ease;
}
button:hover {
    background: var(--accent-hover);
    transform: translateY(-2px);
}
.visitor-list {
    text-align: left;
```

```
            margin-top: 2rem;
        }
        .visitor-list h3 {
            font-size: 1.2rem;
            margin-bottom: 1rem;
            color: var(--text-primary);
        }
        .visitor-item {
            background: rgba(0, 0, 0, 0.2);
            border-radius: 12px;
            padding: 1rem;
            margin-bottom: 0.5rem;
            display: flex;
            justify-content: space-between;
            align-items: center;
            border: 1px solid var(--glass-border);
        }
        .visitor-name {
            font-weight: 600;
        }
        .visitor-meta {
            font-size: 0.85rem;
            color: var(--text-secondary);
        }
        .anon {
            font-style: italic;
            color: var(--text-secondary);
        }
    </style>
</head>
<body>
    <div class="container">
        <div class="glass-panel">
            <h1>Привіт! □</h1>
            <div class="counter">Цей сайт відвідали <strong>{{ total_visits
}}</strong> разів.</div>

            <div class="form-group">
                <form method="POST">
                    <input type="text" name="visitor_name" placeholder="Введіть ваше
ім'я..." required>
                    <button type="submit">Залишити слід</button>
                </form>
            </div>

            {% if recent_visits %}
            <div class="visitor-list">
                <h3>Останні відвідувачі:</h3>
                {% for visit in recent_visits %}
                <div class="visitor-item">
                    <div class="visitor-name">
                        {% if visit.name %}{{ visit.name }}{% else %}<span
class="anon">Анонім</span>{% endif %}
                    </div>
                    <div class="visitor-meta">
                        {{ visit.ip_address }} &bull; {{
visit.timestamp.strftime('%H:%M, %d.%m.%Y') }}
                    </div>
                </div>
                {% endfor %}
```

```
            </div>
            {% endif %}
        </div>
    </div>
</body>
</html>
"""

@app.route('/', methods=['GET', 'POST'])
def index():
    """
    При кожному візиті:
    1. Реєструє візит у базі даних (з іменем або без).
    2. Підраховує загальну кількість візитів.
    3. Повертає вітальне повідомлення з формою та лічильником.
    """
    db = get_session()
    try:
        # Створюємо таблиці якщо їх ще немає (перший запит)
        Base.metadata.create_all(bind=get_engine())

        # Міграція: додаємо колонку name, якщо її не існує
        db.execute(text("ALTER TABLE visits ADD COLUMN IF NOT EXISTS name
VARCHAR(255);"))
        db.commit()

        # Отримання IP-адреси. Враховуємо, що додаток за проксі (GKE Ingress).
        if request.headers.getlist("X-Forwarded-For"):
            ip_addr = request.headers.getlist("X-Forwarded-For")[0]
        else:
            ip_addr = request.remote_addr

        # Отримання імені користувача з форми
        user_name = None
        if request.method == 'POST':
            user_name = request.form.get('visitor_name', '').strip()
            if not user_name:
                user_name = None

        # Створення нового запису про візит
        new_visit = Visit(ip_address=ip_addr, name=user_name)
        db.add(new_visit)
        db.commit()

        # Підрахунок загальної кількості візитів
        total_visits = db.query(func.count(Visit.id)).scalar()

        # Отримання 5 останніх візитів для відображення
        recent_visits =
db.query(Visit).order_by(Visit.timestamp.desc()).limit(5).all()

        return render_template_string(HTML_TEMPLATE, total_visits=total_visits,
recent_visits=recent_visits)

    except Exception as e:
        # У випадку помилки підключення до БД, повертаємо HTTP 503
        db.rollback()
        print(f"Database error: {e}")
        return f"<h1>Service temporarily unavailable.</h1><p>Database error: {e}</p>",
503
```

```
    finally:
        db.close()

@app.route('/healthz')
def healthz():
    """
    Ендпоінт для перевірки стану (Health Check).
    Завжди повертає 200 OK — додаток живий якщо gunicorn запущений.
    Перевірка БД тут НЕ виконується щоб health check не залежав від стану БД.
    """
    return "OK", 200

if __name__ == '__main__':
    # Запускаємо додаток (для локальної розробки)
    app.run(host='0.0.0.0', port=8080)
```

Д.2 app/Dockerfile

```
# --- Етап 1: Збірка (Builder) ---
FROM python:3.11-slim as builder

WORKDIR /app

# Встановлюємо системні залежності для компіляції (за потреби)
# hadolint ignore=DL3008
RUN apt-get update && apt-get install -y --no-install-recommends gcc libpq-dev && rm -
rf /var/lib/apt/lists/*

COPY requirements.txt .

RUN python -m venv /opt/venv
ENV PATH="/opt/venv/bin:$PATH"
RUN pip install --no-cache-dir -r requirements.txt

# --- Етап 2: Фінальний образ (Final) ---
FROM python:3.11-slim

WORKDIR /app

# Встановлюємо runtime залежності
# hadolint ignore=DL3008
RUN apt-get update && apt-get install -y --no-install-recommends libpq5 && rm -rf
/var/lib/apt/lists/*

COPY --from=builder /opt/venv /opt/venv
COPY app.py .

ENV PATH="/opt/venv/bin:$PATH"

# Створення непривілейованого користувача для безпеки (Best Practice)
RUN groupadd -g 1000 appgroup && useradd -u 1000 -g appgroup -s /bin/sh -m appuser
USER appuser

EXPOSE 8080

CMD ["gunicorn", "--bind", "0.0.0.0:8080", "--workers", "2", "app:app"]
```

Д.3 app/requirements.txt

```
Flask==3.0.0
gunicorn==22.0.0
SQLAlchemy==2.0.23
psycopg2-binary==2.9.9
```