

Міністерство освіти і науки України
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки

ПОЯСНЮВАЛЬНА ЗАПИСКА

до випускної кваліфікаційної роботи бакалавра
на тему: «Проект хмарної інфраструктури вебзастосу-
нку на базі рішень Docker, Kubernetes
та сервісів AWS»

Виконав: студент 4-го курсу групи КІ-21-1
спеціальності 123 «Комп'ютерна інженерія»

 Микола МІЩЕНЧУК

Керівник: доцент кафедри комп'ютерної інженерії та кі-
бербезпеки, кандидат технічних наук,

 Олексій ШЕЛУХА

Рецензент: завідувач кафедри комп'ютерної інженерії та
кібербезпеки, кандидат технічних наук, доцент

 Андрій ЄФІМЕНКО

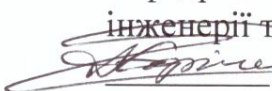
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки
Спеціальність 123 «Комп'ютерна інженерія»
Освітня програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ

Завідувач

кафедри комп'ютерної

інженерії та кібербезпеки

 Андрій ЄФІМЕНКО

3 березня 2025 р.

ЗАВДАННЯ

на випускну кваліфікаційну роботу бакалавра

Студент: Міщенко Микола Миколайович

Тема роботи: «Проект хмарної інфраструктури вебзастосунку на базі рішень
Docker, Kubernetes та сервісів AWS»

затверджена Наказом університету від 3 березня 2025 р. №79/с

Термін здачі студентом закінченої роботи 9 червня 2025 р.

Вихідні дані роботи (зазначається предмет і об'єкт дослідження):

- хмарна інфраструктура, зокрема сервіси AWS, контейнеризовані вебзастосунки
 - методи та засоби контейнеризації, оркестрації, моніторингу та автоматизації функціонування хмарної інфраструктури
- Обладнання та програмне забезпечення:

1. Amazon Web Services
2. Docker
3. Kubernetes
4. GitHub Actions

Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	В.В. Миколайчук	05.03.2025 	05.03.2025 

Студент

Керівник

 Микола МІЩЕНЧУК

 Олексій ШЕЛУХА

Календарний план

№ з/п	Назви етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка завдання	03.03.2025	Виконано
2	Пошук, огляд та аналіз інформації про AWS, Docker та Kubernetes	24.03.2025	Виконано
3	Проектування архітектури хмарної інфраструктури	07.04.2025	Виконано
4	Підготовка тестових веб-застосунків, створення Dockerfile та налаштування docker-compose	14.04.2025	Виконано
5	Налаштування VPC	28.04.2025	Виконано
6	Створення Kubernetes-кластера, конфігурація RBAC	12.05.2025	Виконано
7	Налаштування інтеграції з AWS-сервісами (RDS, S3, ECR)	19.05.2025	Виконано
8	Налаштування CI/CD через GitHub Actions	26.05.2025	Виконано
9	Написання пояснювальної записки	09.06.2025	Виконано

Студент



Микола МІЩЕНЧУК

Керівник



Олексій ШЕЛУХА

АНОТАЦІЯ

Пояснювальна записка містить: 85 сторінок текстової частини, 65 рисунків та 15 джерел.

Метою роботи є дослідження та проектування хмарної інфраструктури для сучасного вебзастосунку з використанням контейнеризації на базі Docker, оркестрації за допомогою Kubernetes та сервісів AWS.

У першому розділі даної кваліфікаційної роботи проведено аналіз основних теоретичних підходів до побудови хмарних інфраструктур, досліджено переваги використання Docker і Kubernetes для автоматизації розгортання та масштабування застосунків.

У другому розділі розглянуто особливості проектування мережі для хмарної інфраструктури, налаштування взаємодії різних компонентів через сучасні хмарні сервіси, а також впровадження автоматизації процесу розгортання додатку.

У третьому розділі розроблено та впроваджено хмарну інфраструктуру, яка забезпечує гнучкість, масштабованість та відмовостійкість вебзастосунку.

Ключові слова: ХМАРНА ІНФРАСТРУКТУРА, ОРКЕСТРАЦІЯ, ВЕБЗАСТОСУНОК, КОНТЕЙНЕРИЗАЦІЯ, БЕЗПЕКА, МАСШТАБУВАННЯ, CI/CD, DEVOPS.

					ККІК.КР.Б – 123 – 11 – ПЗ			
Зак.	Арк.	№ докум.	Підпис	Дата	Проект хмарної інфраструктури вебзастосунку на базі рішень Docker, Kubernetes та сервісів AWS	Літ.	Арк.	Аркушів
Розроб.		Міценчук М.М.		23.06.25				
Керівник		Шелуха О.О.		25.06.25			4	85
Реценз.		Єфіменко А.А.		26.06.25		ФІКТ, Гр. КІ-21-1		
Н. контр.		Шелуха О.О.		25.06.25				
Затверд.		Єфіменко А.А.		25.06.25				

ANNOTATION

The qualification work contains 85 pages, 65 illustrations, a list of 15 references.

The aim of this work is to research and design a cloud infrastructure for a modern web application using containerization based on Docker, orchestration with Kubernetes, and AWS services.

The first section of this qualification work analyzes the main theoretical approaches to building cloud infrastructures, and examines the advantages of using Docker and Kubernetes for automating deployment and scaling of applications.

The second section considers the specifics of designing the network for the cloud infrastructure, setting up interaction between various components through modern cloud services, as well as implementing automation of the application deployment process.

The third section presents the development and implementation of a cloud infrastructure that ensures flexibility, scalability, and fault tolerance of the web application.

Keywords: CLOUD INFRASTRUCTURE, CI/CD, KUBERNETES, DOCKER, AWS, ORCHESTRATION, WEB APPLICATION, CONTAINERIZATION, SECURITY, SCALABILITY, DEVOPS.

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				5
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

ВСТУП.....	8
РОЗДІЛ 1 ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО ТЕХНОЛОГІЇ ПОБУДОВИ ХМАРНОЇ ІНФРАСТРУКТУРИ.....	9
1.1 Сучасні підходи до побудови хмарних інфраструктур вебзастосунків.....	9
1.2 Контейнеризація додатків за допомогою Docker	12
1.3 Оркестрація контейнерів із використанням Kubernetes	16
1.4 Аналіз інфраструктурних сервісів Amazon Web Services.	20
1.5 Аналіз інструментів безперервної інтеграція та доставки	23
Висновки до першого розділу	25
РОЗДІЛ 2 ПРОЄКТУВАННЯ ХМАРНОЇ ІНФРАСТРУКТУРИ ДЛЯ ВЕБЗАСТОСУНКУ.....	27
2.1 Аналіз вимог до системи	27
2.2 Проєктування загальної архітектури хмарної інфраструктури	29
2.3 Проєктування топології мережі	31
2.4 Проєктування кластеру Kubernetes.....	34
Висновки до розділу 2.....	35
РОЗДІЛ 3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ ЕФЕКТИВНОСТІ ХМАРНОЇ ІНФРАСТРУКТУРИ.....	37
3.1 Налаштування Docker	37
3.2 Створення та налаштування мережі	40
3.3 Створення EKS кластеру	45
3.4 Створення інших сервісів AWS	55
3.5 Налаштування Ingress Controller та тестування роботи системи.....	59
3.6 Налаштування CI/CD	63
Висновки до розділу 3.....	66
ВИСНОВКИ	68
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ.....	69
ДОДАТКИ.....	71

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AWS – Amazon Web Services

IaC – Infrastructure as Code

K8s – Kubernetes

VPC – Virtual Private Cloud

EKS – Elastic Kubernetes Service

RDS – Relational Database Service

S3 – Simple Storage Service

IAM – Identity and Access Management

CI/CD – Continuous Integration / Continuous Deployment

DNS – Domain Name System

RBAC – Role-Based Access Control

NAT – Network Address Translation

ALB – Application Load Balancer

ELB – Elastic Load Balancer

AZ – Availability Zone

API – Application Programming Interface

CLI – Command Line Interface

VPN – Virtual Private Network

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

YAML – YAML Ain't Markup Language

IP – Internet Protocol

CIDR – Classless Inter-Domain Routing

HPA – Horizontal Pod Autoscaler

SG – Security Group

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				7
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми. Сучасні вебзастосунки вимагають високої доступності, надійності та здатності швидко адаптуватися до змін навантаження. Традиційні підходи до побудови інфраструктури не завжди можуть забезпечити ці характеристики, що призводить до простоїв, втрат даних та неефективного використання ресурсів. Використання хмарних технологій на базі Docker, Kubernetes і AWS дозволяє побудувати автоматизовану та високодоступну інфраструктуру, яка легко масштабується та витримує високі навантаження.

Метою кваліфікаційної роботи є створення та реалізація проєкту хмарної інфраструктури з використанням рішень Docker, Kubernetes та сервісів AWS, що забезпечить високу доступність, автоматичне масштабування та надійність вебзастосунків.

Визначена мета обумовлює наступні завдання:

- Аналіз та порівняння сучасних технологій для побудови хмарної інфраструктури вебзастосунків.
- Проєктування архітектури віртуальної приватної хмари (VPC).
- Розгортання Kubernetes-кластера за допомогою AWS EKS, налаштування RBAC та управління доступом.
- Налаштування автоматичного розгортання додатків із використанням інструментів CI/CD.

Об'єктом дослідження є процес побудови, розгортання та керування хмарною інфраструктурою контейнеризованих вебзастосунків.

Предметом дослідження є технології та методи контейнеризації, оркестрації, інфраструктурні сервіси AWS, а також підходи до організації безпеки, моніторингу та автоматизації процесів CI/CD у хмарному середовищі.

У процесі виконання кваліфікаційної роботи було розроблено проєкт хмарної інфраструктури на базі Amazon Web Services з використанням Docker і Kubernetes, що забезпечує автоматичне масштабування, високу доступність та безпеку вебзастосунків, а також автоматизоване розгортання додатків за допомогою системи безперервної інтеграції та доставки.

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

РОЗДІЛ 1 ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО ТЕХНОЛОГІЇ ПОБУДОВИ ХМАРНОЇ ІНФРАСТРУКТУРИ

1.1 Сучасні підходи до побудови хмарних інфраструктур вебзастосунків

На сучасному етапі розвитку інформаційних технологій особливу актуальність набуває перехід від традиційної архітектури вебзастосунків до хмарних рішень. Використання традиційних методів, таких як виділені фізичні сервери або прості віртуальні машини, все частіше демонструє свої недоліки, зокрема обмеженість у масштабуванні, недостатню відмовостійкість, складність в управлінні ресурсами та високу вартість підтримки інфраструктури.

На зміну традиційним підходам приходять хмарні технології, які забезпечують високу доступність, гнучкість та масштабованість. Хмарні технології мають багато переваг, які призводять до їх широкого використання в різних сферах.

Таблиця 1.1

Порівняння традиційних і хмарних інфраструктур

Критерій	Традиційна інфраструктура	Хмарна інфраструктура
Початкові витрати	Високі (обладнання, ліцензії)	Мінімальні (оплата за фактичне використання)
Операційні витрати	Постійно високі (обслуговування, енергоспоживання, персонал)	Гнучкі, залежно від обсягу споживаних ресурсів
Масштабованість	Обмежена, потребує закупівлі додаткового обладнання	Майже необмежена, здійснюється за кілька хвилин
Доступність	Залежить від резервування та власних ресурсів	Гарантується провайдером (SLA, multi-AZ)
Відмовостійкість	Потребує додаткових рішень та інвестицій	Реалізується вбудованими механізмами хмари

Основними перевагами хмарних рішень є здатність швидко адаптуватися до змін навантаження, знижені витрати на експлуатацію інфраструктури, автоматичне резервне копіювання та можливість використання ресурсів за моделлю «pay-as-you-go». Ця модель являє собою спосіб розрахунку лише за ті ресурси, які були реально використані. На відміну від традиційних моделей із фіксованою абонплатою, тут оплата здійснюється за конкретні обчислювальні потужності, обсяги зберігання даних, мережевий трафік та інші ресурси залежно від їх фактичного споживання. Це дозволяє ефективно керувати витратами і уникнути переплат за невикористані ресурси.

Самообслуговування та автоматизація також на високому рівні, користувачі можуть самостійно керувати ресурсами, запускати віртуальні машини, налаштовувати сервіси та автоматизувати багато процесів, що прискорює розгортання та управління інфраструктурою. Будь який користувач може отримувати доступ до своїх даних та ресурсів з будь-якого місця, де є Інтернет, що полегшує роботу на віддалених робочих місцях та сприяє спільній роботі. Великі хмарні постачальники, такі як Amazon Web Services, Microsoft Azure, та Google Cloud, зазвичай вкладають значні зусилля в забезпечення безпеки даних, включаючи шифрування, автентифікацію та інші заходи.

Незважаючи на численні переваги, хмарні сервіси також мають певні недоліки, які можуть суттєво вплинути на роботу бізнесу та взаємодію з користувачами. Одним із ключових моментів є залежність роботи сервісів від стабільного підключення до мережі Інтернет. За відсутності такого підключення можливе виникнення проблем із доступом до важливих даних та сервісів.

Окрім цього, використання хмарних рішень часто викликає занепокоєння стосовно приватності та безпеки інформації. Зберігання конфіденційних даних на серверах сторонніх постачальників може призводити до недовіри з боку користувачів. Хоча провайдери хмарних послуг докладають значних зусиль для забезпечення високого рівня безпеки, ризики витоків інформації та кібератак не можуть бути повністю виключені.

		Міценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		10

Ще одним фактором, що обмежує використання хмарних сервісів, є часткова втрата контролю над інфраструктурою. Обчислювальні ресурси, певні налаштування та аспекти безпеки керуються не самими користувачами, а провайдерами, що може бути небажаним для деяких компаній із високими вимогами до контролю.

З точки зору економічної ефективності, хмарні рішення часто позиціонуються як більш вигідні, однак це не завжди є абсолютною істиною. Витрати на використання хмарних послуг можуть зростати в довгостроковій перспективі або через неочікувані обставини, наприклад, різке збільшення використання ресурсів або зміну тарифних планів постачальників.

Ще одним суттєвим недоліком є обмеження гнучкості налаштування сервісів та інфраструктури. Користувачі не завжди можуть повною мірою адаптувати сервіси під свої специфічні потреби, оскільки постачальники пропонують стандартні налаштування, що не завжди відповідають конкретним бізнес-вимогам.

Також варто зазначити, що компанії, використовуючи хмарні послуги, не завжди мають повний контроль над місцезнаходженням власних даних, оскільки постачальники можуть розміщувати їх у дата-центрах, розташованих у різних країнах, що може викликати додаткові юридичні питання.

Таким чином, хоча більшість із зазначених проблем можна суттєво зменшити за допомогою правильного планування та ретельного вибору постачальника хмарних послуг, вони мають бути ретельно розглянуті та враховані перед впровадженням хмарних технологій.

Серед сучасних підходів до побудови хмарних інфраструктур вебзастосунків особливо виділяються такі технології та методи:

- **Контейнеризація:** технологія, що дозволяє упаковувати застосунки разом із залежностями в ізольовані контейнери, забезпечуючи стандартизоване середовище для розгортання і запуску застосунків у будь-якому оточенні.
- **Оркестрація контейнерів:** дозволяє автоматизувати процеси управління, масштабування та забезпечення відмовостійкості контейнерних застосунків, а також значно спрощує адміністрування розподілених систем.

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		11

- **Безсерверні обчислення:** модель, при якій розробники не займаються безпосереднім адмініструванням серверів, а використовують керовані сервіси, що забезпечують запуск і виконання коду без прямого управління ресурсами інфраструктури.
- **Автоматизація розгортання (CI/CD):** забезпечує швидку і безпечну доставку змін в оточення користувачів завдяки автоматичним процесам збирання, тестування і розгортання коду.
- **Інфраструктура як код:** концепція, що дозволяє описувати та керувати інфраструктурою з використанням декларативних конфігураційних файлів, що значно знижує ймовірність помилок і спрощує масштабування.
- **Мікросервісна архітектура:** розбиття великого монолітного застосунку на незалежні сервіси, які можуть бути розгорнуті, масштабовані і оновлені окремо один від одного.
- **Хмарні платформи:** сервіси таких провайдерів дозволяють швидко створювати, масштабувати та управляти ресурсами і сервісами без необхідності обслуговувати фізичні сервери.

Вибір конкретного підходу залежить від вимог і особливостей вебзастосунку, бюджету проєкту, бажаного рівня безпеки, а також необхідної гнучкості й масштабованості. У межах цієї роботи було обрано модель використання контейнеризації, оркестрації, CI/CD та хмарних сервісів AWS як найбільш сучасну, гнучку і надійну основу для реалізації хмарної інфраструктури вебзастосунку.

Таким чином, розглянуті підходи дозволяють вирішити широкий спектр задач: забезпечення високої доступності, автоматичного масштабування, зниження операційних витрат, а також підвищення надійності та безпеки вебзастосунків у хмарному середовищі.

1.2 Контейнеризація додатків за допомогою Docker

В останній час контейнеризація стала невід’ємною частиною розробки та розгортання сучасних вебзастосунків. Основною технологією, яка здобула найбільшу популярність у цій сфері, є Docker. Контейнери дозволяють ізолювати застосунки

		Міценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		12

разом із усіма їхніми залежностями, створюючи самодостатнє середовище для запуску незалежно від операційної системи чи конфігурації хосту.

Головною перевагою Docker є можливість швидко і легко створювати, пакувати та переносити застосунки. Для цього використовуються так звані образи Docker.

Docker-образи - це основа для створення Docker-контейнерів. Образ є шаблоном лише для читання, який містить весь необхідний для запуску застосунку код, бібліотеки, залежності та конфігураційні файли. Образ формується з кількох шарів, причому кожен шар відповідає окремій інструкції у Dockerfile. Ці шари накладаються один на одного й у сукупності утворюють цілісний Docker-образ.

Однією з ключових переваг Docker-образів є їх портативність. Після створення образу його можна легко переносити та запускати на будь-якому сервері чи комп'ютері з встановленим Docker, незалежно від операційної системи чи особливостей інфраструктури. Це забезпечує однакову роботу застосунків у різних середовищах.

Docker-образи зберігаються у спеціальних реєстрах, таких як Docker Hub або приватні корпоративні реєстри. Завдяки цьому розробники можуть зручно обмінюватися образами, співпрацювати й організовувати спільне використання.

Dockerfile - це текстовий файл, у якому записано інструкції для автоматизації процесу створення Docker-образу. Він слугує своєрідною схемою, за якою будується образ із усім необхідним: програмним кодом застосунку, залежностями та середовищем виконання. Dockerfile має спеціальний формат і декларативний синтаксис, що дає змогу послідовно описати кроки для побудови образу. Кожна інструкція в цьому файлі створює окремий шар, що підвищує ефективність збирання та поширення Docker-образів. [\[1\]](#)

		Міценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		13

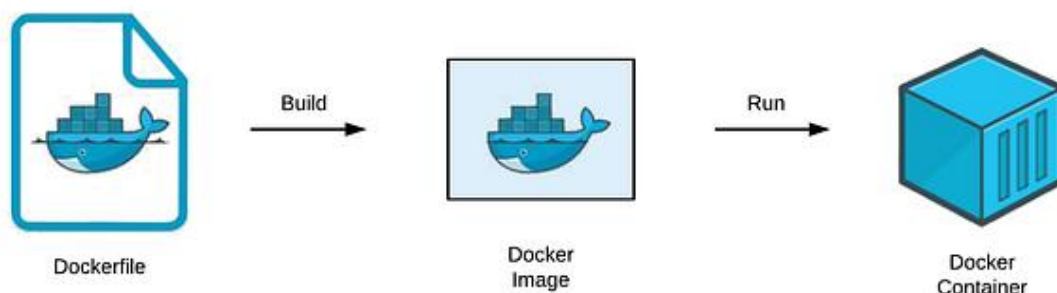


Рисунок 1.1 – Схема створення контейнера

Контейнери, на відміну від класичних віртуальних машин, не містять власного ядра операційної системи. Вони використовують ядро хост-машини, що дозволяє запускати значно більше ізольованих середовищ на тих самих фізичних ресурсах. Це знижує навантаження, підвищує швидкість запуску та скорочує витрати на обслуговування інфраструктури. [2]

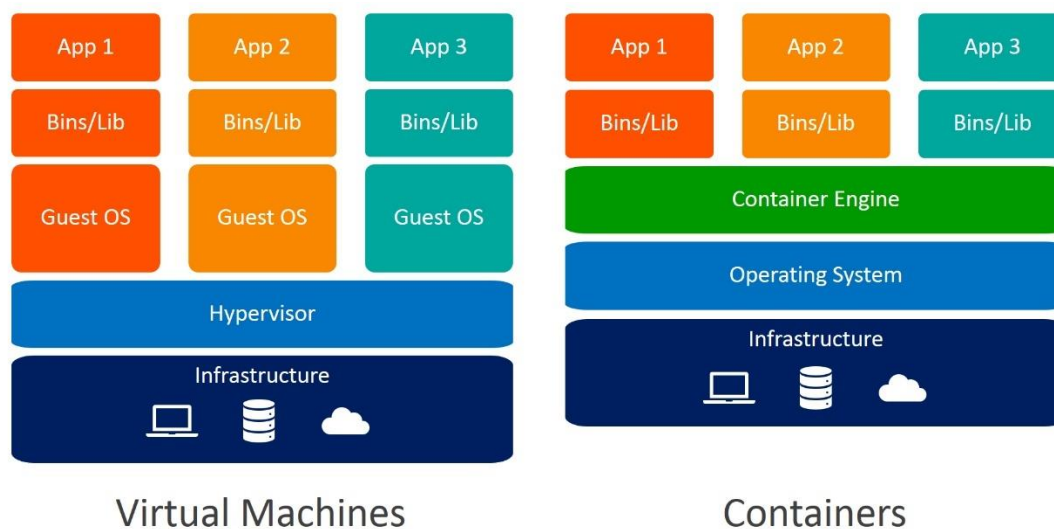


Рисунок 1.2 – Порівняння структури віртуальної машини та контейнера

Контейнеризація також полегшує підтримку мікросервісної архітектури, де кожен сервіс може бути розгорнутий у власному ізольованому середовищі й оновлюватись незалежно від інших компонентів системи.

У сучасних хмарних інфраструктурах, таких як AWS, контейнери Docker стають стандартом для розгортання та масштабування застосунків. Сервіси на зразок Amazon Elastic Kubernetes Service або Amazon Elastic Container Service використовують Docker-контейнери як базову одиницю обчислень.

Таблиця 1.2

Порівняння контейнерів та віртуальних машин

Характеристика	Контейнери	Віртуальні машини
Архітектура	Використовують ядро хост-ОС, ізольовані на рівні процесів	Включають гіпервізор і повну гостьову ОС
Ресурси	Легковагі, займають мало місця	Більш ресурсоємні
Швидкість запуску	Дуже швидкий запуск	Відносно повільний запуск
Ізоляція	Менш повна (на рівні ОС)	Повна (на рівні гостьової ОС)
Портативність	Висока, однаковий образ на різних хостах	Вимагає сумісності гостьових ОС
Управління	Просте масштабування, автоматизація	Більш складне, потребує адміністрування
Масштабованість	Висока, легко додаються/видаляються	Обмежена через ресурсоємність
Оновлення	Можна оновлювати окремі контейнери	Оновлення всієї ВМ або сервісу
Вартість	Дешевше в експлуатації	Дорожче в обслуговуванні

Таким чином, Docker став універсальним стандартом для побудови сучасних розподілених систем, забезпечуючи простоту, портативність та ефективність при роботі як з локальними, так і з хмарними інфраструктурами.

1.3 Оркестрація контейнерів із використанням Kubernetes

Зі зростанням складності сучасних вебзастосунків і переходом до мікросервісної архітектури виникла необхідність у потужних інструментах для автоматизованого управління великою кількістю контейнерів. Саме таку роль виконує Kubernetes - одна з найпопулярніших платформ оркестрації контейнерів у світі.

Kubernetes - це відкрита система для автоматизації розгортання, масштабування й управління контейнеризованими застосунками. Її головною перевагою є можливість централізовано керувати великою кількістю контейнерів у кластері, забезпечуючи їхню високу доступність, відмовостійкість і масштабованість. [7]

Серцем кластера Kubernetes є головний вузол (Master Node). Саме він виконує роль контрольної площини, управляючи станом усього кластера. На Master Node покладається прийняття глобальних рішень: розподіл нових Pod-ів по вузлах, моніторинг стану вузлів і контейнерів, масштабування застосунків у відповідь на зміни навантаження.

До складу Master Node входять основні компоненти:

- API Server - центральний компонент для управління кластером, через який користувачі та інші частини системи взаємодіють із Kubernetes за допомогою API.
- etcd - розподілене сховище типу “ключ-значення”, де зберігаються всі налаштування й актуальний стан кластера.
- Controller Manager - відповідає за роботу контролерів, які відстежують стан кластера через API Server і автоматично підтримують бажаний стан. Наприклад, контролер ReplicaSet стежить, щоб завжди працювала задана кількість копій Pod-ів.
- Scheduler - визначає, на яких робочих вузлах будуть запускатися нові Pod-и, виходячи з доступності ресурсів і поточного завантаження, забезпечуючи оптимальний розподіл навантаження між нодами.

Worker Nodes - це машини, на яких запускаються й виконуються контейнери. Вони формують так звану “data plane” - саме тут відбувається виконання основних

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				16
Змн.	Арк.	№ докум.	Підпис	Дата		

робочих навантажень кластера. Кожен робочий вузол містить кілька важливих компонентів:

- Kubelet - це агент, який працює на кожному Worker Node і взаємодіє з Master Node. Kubelet слідкує, щоб усі контейнери, визначені у специфікаціях Pod-ів, були запущені й знаходились у здоровому стані.
- Container Runtime - програмне забезпечення для безпосереднього запуску контейнерів. Kubernetes підтримує різні системи виконання, такі як Docker або containerd. Container Runtime відповідає за завантаження образів контейнерів і їх запуск на вузлі.
- Kube Proxy - компонент, що забезпечує мережеву взаємодію всередині кластера. Kube Proxy налаштовує маршрутизацію трафіку для сервісів і відповідає за балансування навантаження між Pod-ами.

Головний вузол і робочі вузли взаємодіють між собою через API Server Kubernetes. Саме через API Server користувачі та інші компоненти можуть управляти кластером. Наприклад, коли користувач розгортає новий застосунок, його конфігурація надсилається до API Server, який зберігає цю інформацію в etcd. Controller Manager постійно відстежує стан кластера через API Server. Якщо поточний стан відрізняється від бажаного (наприклад, якийсь pod не запущений), Controller Manager автоматично вживає заходів для відновлення необхідного стану. Під час створення нового pod-a Scheduler обирає відповідний Worker Node з урахуванням доступності ресурсів та інших параметрів. API Server сповіщає вибраний вузол, і Kubelet на ньому запускає контейнер. Worker Nodes регулярно надсилають інформацію про стан pod-ів назад до Master Node за допомогою Kubelet. Завдяки цьому головний вузол завжди має актуальні дані про весь кластер і може оперативно реагувати на будь-які зміни.

У Kubernetes існує ціла низка об'єктів, які дозволяють автоматизувати розгортання, управління та масштабування контейнеризованих застосунків. Одним із ключових об'єктів є Deployment - це високорівневий ресурс, який спрощує керування життєвим циклом Pod-ів. Використовуючи Deployment, можна забезпечити

		Міценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		17

автоматичне самовідновлення застосунків, швидке масштабування, оновлення без простоїв та відкат до попередніх стабільних версій у разі помилок. За лаштунками Deployment працює з об'єктом ReplicaSet, який безпосередньо створює, видаляє та підтримує необхідну кількість Pod-ів у кластері.

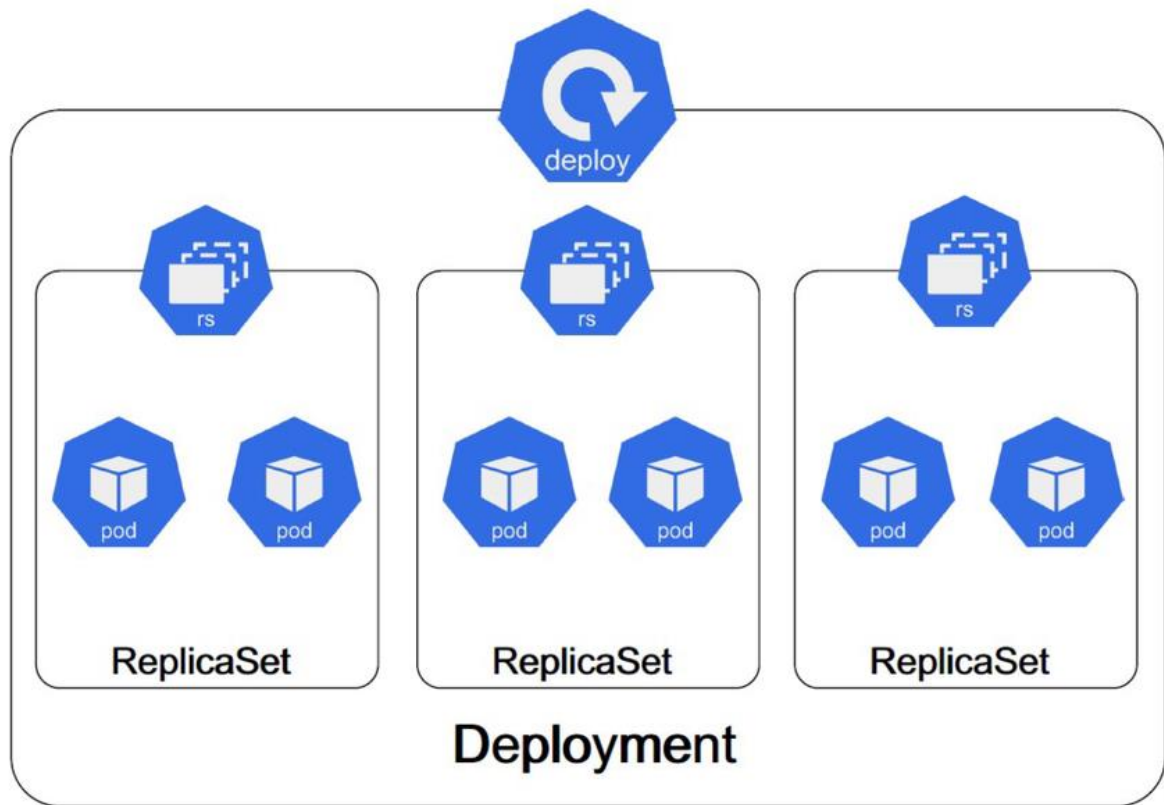


Рисунок 1.3 – Зв'язок між Deployment, ReplicaSet і Pod

Ще одним важливим об'єктом Kubernetes є Service. Він забезпечує надійну і стабільну мережеву взаємодію для застосунків, які працюють у кластері. Service надає постійне DNS-ім'я та віртуальну IP-адресу для групи Pod-ів, що динамічно змінюються під час роботи. Завдяки цьому трафік автоматично балансується між усіма доступними Pod-ами. Окрім цього, Service дозволяє відкривати застосунок для зовнішнього світу та інтегрувати його з хмарними балансувальниками навантаження.

Для організації доступу до декількох застосунків через одну точку входу використовується об'єкт Ingress. Ingress - це набір правил маршрутизації вхідного HTTP/HTTPS-трафіку до відповідних сервісів усередині кластера. Він підтримує

маршрутизацію на основі доменних імен та шляхів URL, дозволяє реалізовувати безпечне з'єднання і централізоване управління правилами доступу.

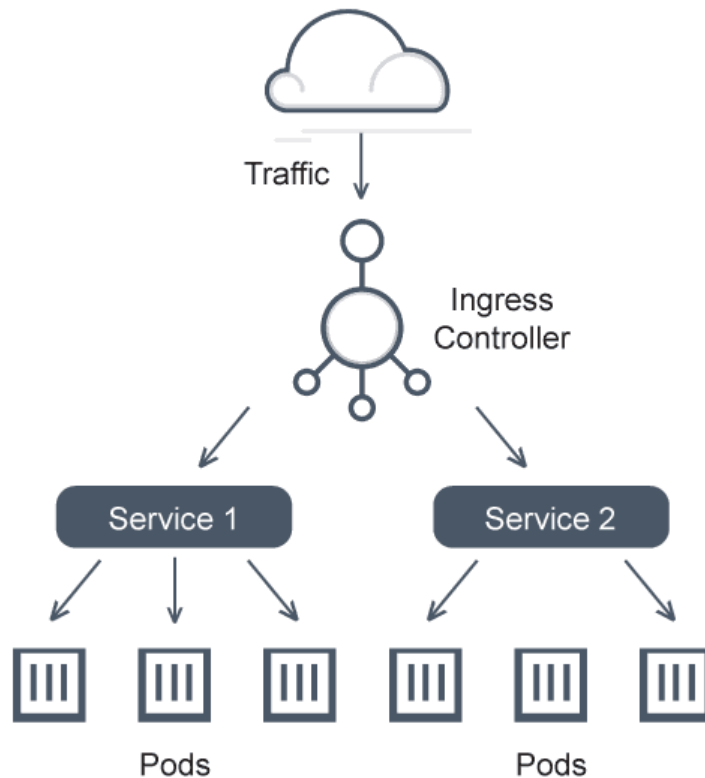


Рисунок 1.4 – Приклад роботи Ingress контроллера

Для зберігання налаштувань застосунків у Kubernetes використовують ConfigMap та Secret. ConfigMap дозволяє передавати застосунку різноманітні конфігураційні параметри або навіть цілі конфігураційні файли у вигляді ключ-значення. Secret служить для зберігання конфіденційної інформації - паролів, токенів, ключів доступу. Обидва типи можна підключати до Pod-ів як змінні середовища або файли. [8]

У сучасних хмарних середовищах, таких як Amazon Web Services, Google Cloud чи Microsoft Azure, Kubernetes часто надається у вигляді керованих сервісів, що дозволяє зосередитись на розгортанні застосунків, а не на обслуговуванні самої платформи.

Таким чином, Kubernetes є ключовим елементом сучасної хмарної інфраструктури, що забезпечує автоматизацію, масштабування та надійність розподілених застосунків.

		Міценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				19
Змн.	Арк.	№ докум.	Підпис	Дата		

1.4 Аналіз інфраструктурних сервісів Amazon Web Services.

Amazon Web Services - це найбільший провайдер хмарних послуг у світі, який пропонує широкий спектр послуг для компаній будь-якого розміру, починаючи від стартапів до великих корпорацій. AWS надає доступ до інфраструктури, що дозволяє клієнтам здійснювати обчислення, зберігання даних, мережі, розгортання програмного забезпечення, аналіз даних, штучний інтелект та багато іншого, без необхідності володіти власною фізичною інфраструктурою.

AWS пропонує гнучкі та масштабовані рішення, які дозволяють клієнтам швидко адаптуватися до змін у потребах бізнесу, збільшувати або зменшувати потужність залежно від навантаження. Крім того, AWS пропонує ряд інструментів для автоматизації, моніторингу та безпеки, що дозволяє клієнтам забезпечувати високу доступність своїх додатків та даних. Компанії можуть використовувати послуги AWS для будь-яких цілей, від розгортання веб-сайтів і мобільних додатків до обчислень великого обсягу даних та розвитку штучного інтелекту. [4]

Amazon Elastic Kubernetes Service (EKS) є повністю керованим сервісом від AWS, який суттєво спрощує запуск, управління та масштабування Kubernetes-кластерів у хмарному середовищі. Завдяки EKS більшість рутинних операцій, пов'язаних із розгортанням та підтримкою Kubernetes, автоматизовані, а сам користувач отримує у своє розпорядження готовий і безпечний кластер практично «з коробки». Управління контрольними вузлами здійснюється самим AWS: це забезпечує надійність, автоматичне оновлення, моніторинг та захист ядра кластера без участі адміністратора.

EKS відзначається високою інтеграцією з іншими сервісами AWS, що дозволяє організувати ефективну взаємодію між додатками, базами даних, системами моніторингу, управління доступом та мережею. За допомогою цього сервісу можна гнучко налаштовувати мережеву інфраструктуру, зокрема, працювати з VPC, підмережами, маршрутними таблицями та групами безпеки для забезпечення ізоляції й контролю доступу.

У EKS легко додавати чи видаляти робочі вузли, як на основі EC2-інстансів, так і використовуючи серверлес-рішення на кшталт Fargate. Для налаштування

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		20

рівнів доступу до ресурсів застосовуються механізми IAM та RBAC, які гарантують чіткий розподіл прав для користувачів і сервісів. Розгортання додатків виконується стандартними інструментами Kubernetes: YAML-маніфестами, Helm-чартами чи CI/CD-процесами.

Завдяки тісній інтеграції з CloudWatch та іншими моніторинговими рішеннями, адміністратор може відстежувати стан кластера, отримувати сповіщення про збої та керувати масштабуванням. Оновлення та підтримка кластера також здійснюються без зупинки сервісу, що гарантує відсутність простоїв для користувачів.

Amazon Elastic Container Registry (ECR) - це керований хмарний реєстр для зберігання, управління та обміну Docker-образами в AWS. Його можна уявити як приватний аналог Docker Hub, який інтегрується безпосередньо з іншими сервісами Amazon, наприклад EKS, ECS, Lambda, а також з CI/CD-системами. Завдяки ECR розробник може зручно зберігати свої контейнерні образи в безпечному приватному сховищі, організовувати доступ для команд і автоматизувати деплоймент у Kubernetes або інші сервіси AWS. Образи завантажуються у репозиторій ECR із локальної машини або CI/CD-процесу, а потім використовуються для запуску контейнерів у хмарній інфраструктурі.

ECR повністю управляється AWS: він автоматично масштабується, забезпечує шифрування збережених даних, підтримує контроль доступу через IAM-політики, а також може перевіряти образи на наявність вразливостей. [\[15\]](#)



Рисунок 1.5 – Amazon Elastic Container Registry

		Міценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		21

Amazon Relational Database Service (RDS) - це керований хмарний сервіс для розгортання, адміністрування й масштабування реляційних баз даних у середовищі AWS. Завдяки RDS ти можеш дуже швидко розгорнути будь яку SQL базу даних без необхідності самостійно налаштовувати сервер, слідкувати за оновленнями чи резервним копіюванням. Всю рутину - інсталяцію, автоматичне резервне копіювання, оновлення, моніторинг стану, реплікацію й навіть відновлення після збоїв - Amazon RDS бере на себе. Ти просто створюєш екземпляр бази через консоль AWS або інфраструктурний код, а далі підключаєш свій застосунок до наданої адреси й користуєшся повноцінною SQL-базою.

Однією з ключових переваг RDS є можливість масштабування ресурсів у кількості кліків - можна збільшити об'єм пам'яті, потужність процесора або розмір диску навіть на вже існуючій базі без зупинки роботи застосунку.

RDS підтримує функції реплікації для масштабування читання, автоматичне резервне копіювання з можливістю відновлення за конкретний момент часу, а також високу доступність завдяки Multi-AZ.

Сервіс інтегрується з іншими службами AWS, зокрема з Kubernetes (EKS) - додатки можуть безпечно підключатися до бази через внутрішню приватну мережу, використовуючи секрети та параметри зберігання. [\[11\]](#)

Amazon Simple Storage Service (S3) - це об'єктне хмарне сховище, яке дозволяє зберігати та керувати будь-якими файлами: від зображень, відео та документів до бекапів, логів і статичних сайтів. S3 розроблено для високої надійності й масштабованості, тому ти можеш зберігати в ньому фактично необмежену кількість даних, а доступ до файлів можливий із будь-якої точки світу через інтернет або приватну мережу AWS.

Головна фішка S3 - простота роботи. Ти створюєш так званий бакет - це щось на кшталт папки чи контейнера, куди можна завантажувати, зберігати та організовувати свої файли. Кожен об'єкт у бакеті має унікальне ім'я та може супроводжуватися додатковими метаданими.

		Міценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		22

S3 підтримує різні режими доступу до файлів: можна зробити об'єкти повністю публічними, доступними лише певним користувачам або сервісам, або ж обмежити доступ через IAM-політики чи “presigned URLs”. Усе це дозволяє тонко контролювати, хто, коли і як може працювати з твоїми даними.

Важливою перевагою є підтримка версіонування - можна зберігати історію змін кожного файлу, що спрощує відновлення даних у разі помилкового видалення чи змін. Також S3 автоматично реплікує дані по кількох дата-центрах для надійності та зберігає їх у зашифрованому вигляді, якщо потрібно. [\[12\]](#)

1.5 Аналіз інструментів безперервної інтеграція та доставки

В умовах сучасної розробки програмного забезпечення безперервна інтеграція та доставка (CI/CD) стали стандартом для команд, що прагнуть швидко та якісно впроваджувати зміни в застосунках. Головна ідея CI полягає в тому, що зміни від різних розробників регулярно інтегруються в спільний репозиторій, де автоматично перевіряються на наявність помилок. Це дає змогу уникати конфліктів, підвищує якість коду й дозволяє оперативно знаходити та виправляти баги ще до виходу релізу.



Рисунок 1.6 – CI/CD pipeline

		Міценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		23

Порівняння популярних CI/CD-інструментів

Критерій	GitHub Actions	GitLab CI/CD	Jenkins
Тип розгортання	SaaS (хмара), частково self-hosted	SaaS (хмара) та self-hosted	Self-hosted, може бути хмарним
Інтеграція з Git	Глибока (GitHub)	Глибока (GitLab)	Підтримка Git, SVN, Mercurial
Конфігурація	YAML (workflow)	YAML (.gitlab-ci.yml)	Groovy (Jenkinsfile), UI
Масштабованість	Автоматична у хмарі, або свої runners	Автоматична або свої runners	Вимагає ручного налаштування
Плагіни/розширення	Marketplace, обмежено	Велика кількість інтеграцій	Дуже багато (Jenkins Plugins)
Моніторинг	Вбудований, базова аналітика	Вбудовані дашборди, аналітика	Вимагає окремих плагінів
Підтримка Docker/K8s	Пряма підтримка	Пряма підтримка	Пряма підтримка через плагіни

Коли розробник створює нову гілку або відправляє pull request, CI-система автоматично запускає низку перевірок: лінтинг коду, unit-тести, збірку контейнерного образу, перевірку залежностей і за потреби, додатковий аудит безпеки.

Після успішного проходження тестів створюється новий Docker-образ, який завантажується у реєстр. Це гарантує, що саме цей, перевірений, артефакт буде використовуватися під час розгортання в середовищі розробки, тестування або продакшн.

Далі в гру вступає Continuous Delivery (CD): зміни автоматично розгортаються у відповідне середовище, наприклад, у Kubernetes-кластер на AWS EKS. Процес деплою теж повністю автоматизований: система викликає kubectl або helm, застосовує новий маніфест, виконує rolling update, моніторить стан pod-ів та в разі невдачі може автоматично виконати rollback на попередню стабільну версію.

CI/CD-процеси тісно інтегруються з системами моніторингу йннн, наприклад, відправляють повідомлення у Slack чи інші месенджери про результат виконання пайплайнів, а також можуть збирати метрики для подальшого аналізу. Завдяки сучасним CI/CD-інструментам компанії можуть розділити процес деплою за середовищами (develop, staging, production), захищати продакшн через перевірки та ручне підтвердження релізу, а також швидко масштабувати пайплайни під потреби різних команд і проєктів.

У результаті впровадження CI/CD прискорюється життєвий цикл застосунків, підвищується якість і стабільність продукту, зменшуються ризики людських помилок, а сам процес релізу стає прозорим, відслідковуваним і керованим. Для хмарних і контейнерних інфраструктур такі підходи стали невід’ємною частиною розробки.

Висновки до першого розділу

Отже, у першому розділі було розглянуто основні теоретичні відомості побудови сучасної хмарної інфраструктури для вебзастосунків. Проведено аналіз розвитку хмарних технологій і порівняно традиційні та хмарні підходи до організації інфраструктури, що дозволило визначити ключові переваги використання хмарних сервісів у сучасних ІТ-проєктах.

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		25

Було детально досліджено концепцію контейнеризації на базі Docker, яка забезпечує ізоляцію, портативність і спрощує розгортання застосунків у різних середовищах. Окрема увага приділена оркестрації контейнерів за допомогою Kubernetes - як ефективному інструменту для автоматизації, масштабування та забезпечення відмовостійкості застосунків.

Розглянуто можливості основних сервісів AWS, таких як EKS, ECR, RDS та S3, що забезпечують гнучкість, безпеку й простоту в адмініструванні хмарної інфраструктури. Описано сучасні підходи до безперервної інтеграції та доставки, які значно пришвидшують і спрощують процес релізу та оновлення програмного забезпечення.

Отже, результати теоретичного аналізу цього розділу створюють надійну основу для подальшого проектування та практичної реалізації хмарної інфраструктури вебзастосунку на базі Docker, Kubernetes і сервісів AWS.

		Міценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		26

РОЗДІЛ 2 ПРОЄКТУВАННЯ ХМАРНОЇ ІНФРАСТРУКТУРИ ДЛЯ ВЕБЗА- СТОСУНКУ

2.1 Аналіз вимог до системи

Початковий етап проєктування хмарної інфраструктури для сучасного вебзастосунку передбачає детальний аналіз вимог, які визначають основний вектор розвитку майбутнього рішення. Цей процес допомагає не лише узгодити очікування всіх зацікавлених сторін, але й уникнути типових помилок, пов'язаних з недостатньою увагою до специфіки бізнесу, ролей користувачів, вимог до надійності, масштабованості та безпеки. Власне, саме якісний аналіз вимог формує підґрунтя для подальшого вибору архітектури, технологій та стратегій автоматизації.

Система, що проєктується, повинна підтримувати широкий спектр сценаріїв: від базового керування користувачами та їхньою автентифікацією до зберігання, обробки й захисту різноманітних типів даних. Кожен користувач взаємодіє із застосунком через інтуїтивний веб-інтерфейс, де має змогу створювати обліковий запис, авторизуватися, керувати персональними даними, переглядати та редагувати інформацію, завантажувати файли та використовувати додатковий функціонал залежно від призначеної ролі. Окрім кінцевих користувачів, у системі передбачено адміністративну панель, через яку адміністратор отримує розширені права: перегляд журналів активності, управління іншими обліковими записами, налаштування доступу до окремих модулів чи сервісів, контроль за станом інфраструктури.

Для реалізації можливостей масштабування та ефективної інтеграції із зовнішніми сервісами, а також з метою організації резервного копіювання, аналітики й моніторингу, у системі активно використовуються сервісні акаунти. Вони можуть діяти у вигляді автономних процесів або взаємодіяти із застосунком через API. Подібний підхід забезпечує не лише гнучкість у побудові нових бізнес-процесів, але й закладає основу для подальшого розширення функціоналу.

Важливо, щоб вебзастосунок демонстрував стабільну роботу незалежно від поточного навантаження, часу доби чи географічного розташування користувача. Для цього особливу увагу приділяється питанням продуктивності й доступності.

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		27

Всі основні операції - реєстрація, логін, виконання запитів, збереження даних - повинні виконуватися швидко, без затримок і з гарантованою якістю сервісу. Надійна робота системи має зберігатися навіть у випадку збоїв окремих компонентів або аварій в інфраструктурі хмарного провайдера. Саме тому у вимогах передбачено високу доступність, підтримку автоматичного відновлення, та масштабування.

Серед найважливіших нефункціональних вимог - забезпечення високого рівня інформаційної безпеки. Захист персональних даних користувачів, шифрування під час зберігання та передавання даних, гнучкий контроль доступу до всіх сервісів і компонентів, а також регулярний аудит дій користувачів - усе це повинно стати невід'ємною частиною архітектури. При цьому для адміністраторів та розробників важливо забезпечити простий і прозорий процес оновлення системи, що включає автоматизоване тестування, деплой та моніторинг через сучасні CI/CD-процеси.

Важливим етапом аналізу є також оцінка типових та пікових навантажень на систему. При проектуванні враховується можливість різкого зростання кількості користувачів, особливо під час маркетингових кампаній або при запуску нових сервісів. Завдяки використанню хмарної інфраструктури, а саме автоматичного балансування навантаження, масштабування обчислювальних потужностей, а також гнучкої організації баз даних та файлового сховища, система готова до таких сценаріїв розвитку.

З технічної точки зору, доцільним є впровадження детального журналювання та моніторингу, що дозволяє оперативно виявляти й усувати проблеми, а також аналізувати поведінку користувачів і ефективність роботи системи. Дані моніторингу можуть бути використані для подальшої оптимізації ресурсів і зниження витрат.

Ще одним важливим аспектом аналізу вимог виступає визначення основних ризиків. До них належить можливість втрати доступу до інфраструктури внаслідок збоїв у роботі хмарного провайдера, ризики компрометації даних через помилки в налаштуваннях, неконтрольоване зростання витрат через надмірне масштабування, а також людський фактор у процесах деплою чи оновлення системи. Розуміння цих викликів дозволяє підготуватися до них заздалегідь, визначити стратегії резервування, автоматичного відновлення та реагування на інциденти.

		Міценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		28

2.2 Проєктування загальної архітектури хмарної інфраструктури

На основі аналізу вимог до майбутньої системи формується цілісне бачення архітектури хмарної інфраструктури, яка має забезпечити надійність, масштабованість та ефективність роботи вебзастосунку. Архітектурне рішення має відповідати сучасним підходам до розгортання й обслуговування хмарних сервісів, а також враховувати специфіку застосованих технологій - Docker, Kubernetes та інструментів AWS.

Основна ідея побудови архітектури полягає у розділенні інфраструктури на окремі логічні компоненти, кожен із яких виконує певну роль у забезпеченні роботи системи. Кластер Kubernetes стає центральним ядром, у якому розгортаються й масштабуються всі основні сервіси: бекенд- і фронтенд-додатки, допоміжні сервіси, автоматизовані процеси. За рахунок мікросервісної організації кожен компонент може розвиватися незалежно, що полегшує масштабування та підтримку.

Важливим елементом інфраструктури виступає віртуальна приватна хмара (VPC), яка дозволяє створити ізольоване, захищене середовище для всіх ресурсів проєкту. Усередині VPC формуються підмережі з чітким розподілом ролей: публічні підмережі слугують для доступу іззовні та розміщення балансувальників навантаження, приватні підмережі - для основних застосунків і сервісів, а окремі ізольовані сегменти - для бази даних і резервного зберігання даних. Такий підхід мінімізує ризики несанкціонованого доступу та підвищує безпеку. [\[5\]](#)

Кластер Kubernetes інтегрується із мережею через окремі групи безпеки, які визначають, які порти й протоколи відкриті для внутрішньої та зовнішньої взаємодії. Всі додатки та сервіси, що працюють у кластері, комунікують із зовнішнім світом виключно через балансувальник навантаження. Це дозволяє розподіляти трафік, захищати додатки від DDoS-атак та масштабувати систему автоматично залежно від поточного навантаження.

Зберігання структурованих даних відбувається у керованій реляційній базі даних, яка розташована в окремому приватному сегменті мережі та недоступна на пряму ззовні. Для збереження файлів, резервних копій та статичних ресурсів використовується сервіс Amazon S3.

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		29

Процеси автоматизації, зокрема збірка Docker-образів, запуск тестів, деплой нових версій додатків, інтегруються через сервіс GitHub Actions. Готові образи зберігаються у приватному реєстрі Amazon ECR, а розгортання виконується за допомогою стандартних засобів Kubernetes через автоматизовані пайплайни.

Контроль доступу до ресурсів і управління ролями здійснюється через інтеграцію Kubernetes із AWS IAM. Це дозволяє забезпечити чітке розмежування прав для різних користувачів, сервісних акаунтів і процесів автоматизації, знижуючи ризики випадкового або зловмисного втручання в роботу інфраструктури.

При виборі інструментів для CI/CD головними критеріями були: гнучкість налаштування, зручність інтеграції з Git-репозиторіями, підтримка роботи з контейнерами та Kubernetes, масштабованість, безпека, а також популярність та підтримка спільнотою. Аналіз різних платформ показав, що оптимальним вибором для даного проєкту є GitHub Actions. Цей сервіс має нативну інтеграцію з GitHub, дозволяє запускати пайплайни при кожному коміті чи pull request'і, підтримує роботу з Docker, AWS CLI та kubectl, має багатий вибір готових екшенів та прикладів.

Важливою перевагою є й те, що GitHub Actions дозволяє розгорнути багатостадійні пайплайни: з окремими джобами для збірки, тестування, статичного аналізу, створення Docker-образів, завантаження їх у приватний реєстр, а також деплою до Kubernetes-кластера через kubectl.

Ще одна сильна сторона - простота налаштування доступу до секретів через Encrypted secrets, що дає змогу безпечно зберігати токени, ключі й паролі для підключення до AWS або Kubernetes.

Таким чином, проєктування загальної архітектури інфраструктури передбачає побудову багаторівневої, гнучкої й безпечної системи, де кожен компонент працює у власному ізольованому середовищі, а взаємодія між ними ретельно контролюється й автоматизується за допомогою сучасних хмарних сервісів. Це дозволяє забезпечити високу надійність, масштабованість та ефективність роботи всього вебзастосунку навіть у складних і динамічних умовах сучасного IT-ринку.

		Міценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		30

2.3 Проєктування топології мережі

Архітектура мережі - ключова складова безпечної, масштабованої й стабільної хмарної інфраструктури для вебзастосунку. Для цього проєкту було спроектовано ізольовану VPC з чіткою структурою підмереж, що дозволяє оптимально розподілити ресурси, організувати контроль доступу та ефективно використовувати хмарні сервіси.

При проєктуванні VPC для Kubernetes-кластера надзвичайно важливо враховувати обсяг необхідного IP-простору не лише для робочих нод, а й для компонентів Control Plane. Кожна підмережа, у якій розміщується Control Plane, повинна містити щонайменше 16 вільних IP-адрес. Ці адреси потрібні для створення Amazon Elastic Network Interfaces, які забезпечують доступ до API-сервера кластера та інших служб керування.

Під час визначення CIDR-блоків для підмереж робочих нод потрібно забезпечити наявність достатньої кількості IP-адрес для всіх нод, а також для всіх контейнерів, які можуть бути розгорнуті у майбутньому. Недостатній розмір підмережі може призвести до ситуацій, коли розгортання нових подів або сервісів стане неможливим через брак IP-адрес.

Базова IP-адресація обирається з приватних діапазонів згідно зі стандартом RFC1918. Основний CIDR-блок для VPC - 10.0.0.0/22, який дає змогу отримати до 1024 унікальних приватних IP-адрес у межах однієї віртуальної мережі. Це рішення оптимізоване під потреби середнього або великого проєкту: обсяг адрес достатній для кластеру Kubernetes, EC2-ресурсів, сервісів моніторингу та можливого резерву під майбутнє розширення. Додатковий CIDR-блок - 192.168.0.0/23 - виділено під ізольовані сервіси, такі як бази даних чи специфічне сховище, що фізично розділяє мережевий трафік критичних компонентів від основної інфраструктури.

VPC поділена на декілька логічних підмереж із чітко визначеними ролями.

Публічні підмережі (10.0.0.0/24, 10.0.1.0/24) призначені для розміщення компонентів, які повинні бути доступними ззовні - наприклад, балансувальників нава-

		Міценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		31

нтаження чи шлюзів NAT. Завдяки цьому вхідний трафік з інтернету або із зовнішніх мереж потрапляє спочатку на спеціальні керовані компоненти, які забезпечують контрольований доступ до внутрішніх ресурсів.

Приватні підмережі (10.0.2.0/24, 10.0.3.0/24) слугують для розміщення основних застосунків, робочих вузлів Kubernetes-кластера, бекенд-сервісів, допоміжних систем. Ці підмережі не мають прямого виходу в Інтернет, усі вихідні запити здійснюються через NAT Gateway, що забезпечує додатковий рівень безпеки: жоден сервіс із приватної зони не буде напряму доступний із зовнішнього світу.

Окремий сегмент мережі виділено для бази даних (192.0.0.0/24, 192.0.1.0/24), яку розміщено в спеціальних ізольованих підмережах. Це дозволяє забезпечити максимальну ізоляцію даних: база не має прямого виходу в Інтернет, а доступ до неї відкритий лише з приватних підмереж, де працюють застосунки-клієнти.

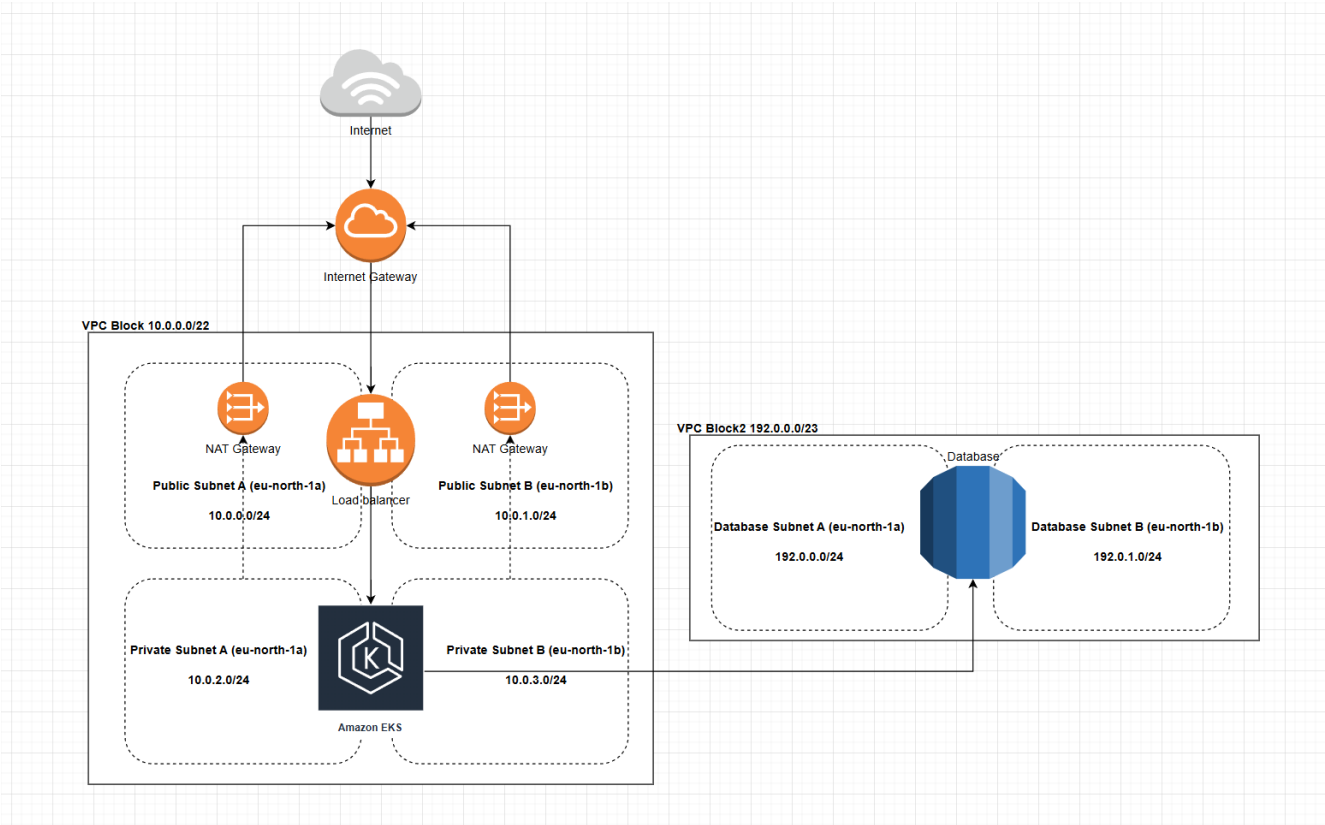


Рисунок 2.1 – Схема мережі

Таблиця підмереж

Підмережа	CIDR	Availability Zone	Тип
Public Subnet A	10.0.10.0/24	eu-north-1a	Публічна
Public Subnet B	10.0.20.0/24	eu-north-1b	Публічна
Private Subnet A	10.0.11.0/24	eu-north-1a	Приватна
Private Subnet B	10.0.21.0/24	eu-north-1b	Приватна
Database Subnet A	192.0.0.0/24	eu-north-1a	Ізольована
Database Subnet B	192.0.1.0/24	eu-north-1b	Ізольована

Правильне функціонування мережі забезпечується шляхом створення і конфігурації окремих маршрутних таблиць для кожного типу підмереж. Для публічних підмереж створена окрема маршрутна таблиця, яка містить запис для маршруту всього трафіку за межі VPC (0.0.0.0/0) через інтернет-шлюз. Саме завдяки цьому ресурси в публічних підмережах можуть приймати й надсилати трафік в Інтернет.

Приватні підмережі підключені до окремих маршрутних таблиць, у яких записано маршрут всього зовнішнього трафіку через NAT Gateway. Тобто, усі сервіси у приватних підмережах можуть ініціювати вихідні з'єднання, але не приймати запити напряму з Інтернету. Це дозволяє контролювати безпеку, не обмежуючи функціонал додатків.

Ізольовані підмережі для бази даних мають власну маршрутну таблицю, що дозволяє доступ лише з приватної частини інфраструктури - ніякого зовнішнього виходу чи вхідного трафіку тут не передбачено.

Для підвищення відмовостійкості й географічної розподіленості ресурси дублюються у двох Availability Zones. Це означає, що у кожній зоні створено по одній публічній, приватній та ізольованій підмережі. Такий підхід дозволяє забезпечити безперервну роботу сервісу навіть у разі аварії одного з дата-центрів.

Структура мережі побудована так, щоб у майбутньому можна було легко додати нові підмережі чи розширити CIDR-блоки для розміщення додаткових ресур-

сів. Використання ізольованих приватних і спеціалізованих підмереж суттєво знижує ризики компрометації даних, а налаштування маршрутних таблиць і шлюзів дозволяє ефективно контролювати трафік у межах інфраструктури.

2.4 Проєктування кластеру Kubernetes

Після створення базової мережевої інфраструктури наступним кроком є детальне проєктування Kubernetes-кластера, що виконує роль основного середовища для розгортання, масштабування й обслуговування вебзастосунків.

Оскільки ми працюємо з AWS, то для цього проєкту використовується Amazon EKS, оскільки він спрощує адміністрування, автоматизує оновлення й патчі, чудово інтегрується з іншими AWS-сервісами, а також гарантує високу доступність і відмовостійкість. Використання керованого сервісу знижує ризик помилок у налаштуванні кластера, а також суттєво економить час і ресурси.

Кластер складається з control plane - керуючі ноди, які повністю управляються AWS та worker nodes - робочі вузли, EC2-інстанси, де безпосередньо виконуються поди із застосунками)

Control plane відповідає за оркестрацію, моніторинг стану, автоматичне відновлення, планування подів і взаємодію з API, тоді як worker nodes - за виконання реального навантаження.

Для підвищення відмовостійкості й балансування навантаження, worker nodes розподілені між двома Availability Zones (eu-north-1a та eu-north-1b).

Для досягнення максимальної ізоляції, структурованості та безпеки у межах одного Kubernetes-кластера впроваджується поділ ресурсів за namespaces. Кожен namespace являє собою логічно відокремлене середовище, у якому розміщуються свої об'єкти - деплойменти, сервіси, конфігурації, секрети. Такий підхід дозволяє уникати ситуацій, коли розгортання нового функціоналу або тестування призводить до збоїв у продакшені. Кожен етап життєвого циклу застосунку - розробка, тестування, продакшн - отримує власний namespace із окремими налаштуваннями та змінними середовища.

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		34

Namespaces

Namespace	Призначення	Основні ресурси	Основні користувачі
dev	Середовище для розробки та початкового тестування нових функцій і фіч	Deployment, Service, HPA, ConfigMap, Secret	Розробники, Тестувальники, CI/CD
staging	Тестове середовище, максимально наближене до продакшн, тут проходить фінальна перевірка релізів перед продакшн.	Deployment, Service, HPA, ConfigMap, Secret	тестувальники, CI/CD
prod	Основне робоче середовище, у якому обробляються реальні запити користувачів і зберігаються дані.	Deployment, Service, HPA, ConfigMap, Secret	Кінцеві користувачі, адміністратори, CI/CD

Висновки до розділу 2

У другому розділі було виконано повний цикл проектування хмарної інфраструктури для сучасного вебзастосунку із використанням технологій Docker і Kubernetes у середовищі AWS. Розглянуто підхід до побудови ізольованої мережі Virtual Private Cloud, який передбачає поділ мережі на окремі підмережі відповідно до ролей та функцій - публічні, приватні та ізольовані під зберігання баз даних. Докладно обґрунтовано вибір CIDR-блоків для кожної підмережі з урахуванням можливого масштабування, оптимального використання адресного простору та дотримання вимог до безпеки й ізоляції компонентів.

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		35

У процесі проєктування детально розглянуто принципи побудови маршрутних таблиць та налаштування маршрутів для організації взаємодії між підмережами, забезпечення виходу в Інтернет через Internet Gateway і NAT Gateway, а також ізоляції критичних компонентів від зовнішніх мереж. Окрема увага приділена використанню кількох зон доступності для розміщення підмереж та основних компонентів інфраструктури, що гарантує підвищену відмовостійкість та можливість безперервної роботи навіть у разі відмови одного з дата-центрів.

Архітектура Kubernetes-кластера спроектована з урахуванням найкращих практик: описано взаємодію між control plane та робочими вузлами, що розміщуються в окремих приватних підмережах. Для ізоляції життєвих циклів додатків і мінімізації ризику впроваджено структурування ресурсів кластера за допомогою namespaces (dev, staging, prod). Це дозволяє ефективно організувати робочі процеси, забезпечити гнучкість у керуванні та пришвидшити впровадження нових рішень.

Враховуючи всі ці рішення, сформовано фундамент для побудови масштабованої, надійної, безпечної та зручної для адміністрування інфраструктури. Усі розглянуті архітектурні підходи та обґрунтування вибору компонентів закладають основу для подальшої практичної реалізації, автоматизації та підтримки життєвого циклу вебзастосунку у хмарному середовищі.

		Міценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		36

РОЗДІЛ 3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ ЕФЕКТИВНОСТІ ХМАРНОЇ ІНФРАСТРУКТУРИ

3.1 Налаштування Docker

Першим кроком стало створення окремих Dockerfile для кожного сервісу. Кожен Dockerfile задає базовий образ, визначає робочу директорію, копіює необхідні файли проекту, встановлює залежності, відкриває потрібні порти та прописує команду запуску.

```
apps > be-admin > Dockerfile > ...
1  FROM node:18-alpine
2
3  WORKDIR /app
4
5  COPY package*.json ./
6
7  RUN npm install
8
9  COPY . .
10
11 EXPOSE 3000
12
13 CMD ["npm", "run", "start"]
```

Рисунок 3.1 – Dockerfile для be-admin

Перед тим як перейти до побудови повноцінної хмарної інфраструктури в AWS, важливим кроком є організація зручного та ізольованого середовища для локальної розробки та тестування. Для вирішення цього завдання у проєкті використовується Docker Compose.

Docker Compose - це офіційний інструмент для роботи з багатоконтейнерними додатками у середовищі Docker, що дозволяє описувати всі необхідні сервіси, мережі, змінні середовища та залежності у вигляді єдиного YAML-файлу. Завдяки цьому можна розгорнути всю інфраструктуру проєкту на своєму комп'ютері однією командою, не переймаючись про сумісність бібліотек чи ручне налаштування кожного компонента. [\[3\]](#)

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		37

Повний код файлу docker-compose.yml наведено у додатку, а нижче розглянемо його основні частини та принцип роботи.

Цей файл дозволяє підняти одночасно всі частини майбутньої хмарної системи локально, зберігаючи зв'язки між компонентами. Кожен додаток будується окремо зі своєї папки, має відкритий порт, власну команду запуску та змінні середовища для зручної інтеграції. База даних s запускається зі стандартного Docker-образу та використовує зовнішній том, щоб дані зберігалися навіть після перезапуску контейнера.

Для фронтенд і бекенд-сервісів додатково налаштований механізм автоматичного перезапуску, що значно прискорює розробку - зміни в коді одразу відображаються в контейнері без ручного втручання.

```

  > Run Service
  be-admin:
    build: ./apps/be-admin
    ports:
      - "3001:3000"
    depends_on:
      - postgres
    environment:
      - DB_DIALECT=postgres
      - DB_USERNAME=dev
      - DB_PASSWORD=dev
      - DB_DATABASE=food_mr_db
      - DB_HOST=postgres
      - DB_PORT=5432
    command: npm run start:dev
    networks:
      - food-mr-network
    develop:
      watch:
        - action: sync
          path: ./apps/be-admin/src
          target: /app
        - action: rebuild
          path: ./apps/be-admin/package.json

```

Рисунок 3.2 – Налаштування be-admin в docker-compose.yml

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		38

Цей підхід дозволяє розробникам запускати локально повноцінний “мікро-кластер” додатків, максимально наближений до хмарної інфраструктури. Це спрощує тестування інтеграції та пришвидшує цикл розробки.

Для запуску файлу використовують команду docker-compose up.

```
PS E:\Work\Projects\any-teller\food-mn-app> docker compose up --build --watch
time="2025-06-22T21:41:07+03:00" level=warning msg="E:\Work\Projects\any-teller\food-mn-app\docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion"
[*] Running 0/2 client Building
- Service be-client Building
[*] Building 2.1s (17/18)
[*] Running 1/2s (18/19)
[*] Building 2.4s (18/19)
✓ Service be-client Built
-> exporting config sha256:296083df75f5f0d5ded3e50c4c1cf34b304407784f819dc3a499e67ff4d8c5fe
[*] Building 2.5s (19/20)
[*] Building 4.7s (38/39)
[*] Running 3/4client 4/6 RUN npm install
-> CACHED [fe-client 4/6] RUN npm install
-> [fe-client 5/6] COPY . .
✓ Service be-client Built
[*] Building 5.6s (38/39)
[*] Running 3/4client 4/6 RUN npm install
-> CACHED [fe-client 4/6] RUN npm install
✓ Service be-client Built
-> [fe-client 5/6] COPY . .
[*] Building 6.3s (38/39)
[*] Running 3/4internal load .dockerignore
-> [fe-admin internal] load .dockerignore
-> -> transferring context: 86B
✓ Service be-client Built
[*] Running 3/46s (38/39)
[*] Building 7.2s (38/39)
-> [fe-admin internal] load .dockerignore
✓ Service be-client Built
-> -> transferring context: 86B
[*] Running 3/46s (38/39)
[*] Building 7.8s (38/39)
```

Рисунок 3.3 – Запуск docker compose

Після чого бачимо що усі додатки були успішно запечені на відповідних портах

```
[6:47:49 PM] Starting compilation in watch mode...
be-client-1 | fe-client-1 | ▲ Next.js 15.2.4 (Turbopack)
fe-client-1 | | - Local: http://localhost:3000
fe-client-1 | | - Network: http://172.19.0.5:3000
fe-client-1 | | ✓ Starting...
fe-admin-1 | fe-admin-1 | ▲ Next.js 15.2.4 (Turbopack)
fe-admin-1 | | - Local: http://localhost:3000
fe-admin-1 | | - Network: http://172.19.0.6:3000
fe-admin-1 | | ✓ Starting...
fe-client-1 | | ✓ Ready in 1784ms
fe-admin-1 | | ✓ Ready in 1742ms
be-client-1 | [6:47:52 PM] Found 0 errors. Watching for file changes.
be-admin-1 | [6:47:53 PM] Found 0 errors. Watching for file changes.
be-client-1 | [Nest] 30 - 06/22/2025, 6:47:53 PM LOG [NestFactory] Starting Nest application...
be-client-1 | [Nest] 30 - 06/22/2025, 6:47:53 PM LOG [InstanceLoader] AppModule dependencies initialized +9ms
be-client-1 | [Nest] 30 - 06/22/2025, 6:47:53 PM LOG [RoutesResolver] AppController {}: +6ms
be-client-1 | [Nest] 30 - 06/22/2025, 6:47:53 PM LOG [RouterExplorer] Mapped {/, GET} route +2ms
be-client-1 | [Nest] 30 - 06/22/2025, 6:47:53 PM LOG [NestApplication] Nest application successfully started +3ms
be-admin-1 | [Nest] 29 - 06/22/2025, 6:47:53 PM LOG [NestFactory] Starting Nest application...
be-admin-1 | [Nest] 29 - 06/22/2025, 6:47:53 PM LOG [InstanceLoader] ConfigHostModule dependencies initialized +85ms
be-admin-1 | [Nest] 29 - 06/22/2025, 6:47:53 PM LOG [InstanceLoader] AppModule dependencies initialized +0ms
be-admin-1 | [Nest] 29 - 06/22/2025, 6:47:53 PM LOG [InstanceLoader] ConfigModule dependencies initialized +0ms
be-admin-1 | [Nest] 29 - 06/22/2025, 6:47:53 PM LOG [InstanceLoader] ConfigModule dependencies initialized +0ms
be-admin-1 | Executing (default): SELECT table name FROM information schema.tables WHERE table schema = 'public' AND table name = 'users'
be-admin-1 | Executing (default): SELECT i.relname AS name, ix.indisprimary AS primary, ix.indisunique AS unique, ix.indkey AS indkey, array_agg(a.attname) as column_indexes, array_agg(a.attrelid) AS attrelids FROM pg.indexes ix JOIN pg.indexes i ON i.indexrelid = ix.indexrelid AND a.attrelid = i.attrelid AND a.attname = i.attname GROUP BY i.relname, ix.indisprimary, ix.indisunique, ix.indkey
be-admin-1 | [Nest] 29 - 06/22/2025, 6:47:53 PM LOG [InstanceLoader] DatabaseModule dependencies initialized +100ms
be-admin-1 | [Nest] 29 - 06/22/2025, 6:47:53 PM LOG [RoutesResolver] AppController {}: +6ms
be-admin-1 | [Nest] 29 - 06/22/2025, 6:47:53 PM LOG [RouterExplorer] Mapped {/, GET} route +3ms
be-admin-1 | [Nest] 29 - 06/22/2025, 6:47:53 PM LOG [RouterExplorer] Mapped {/create-user, GET} route +1ms
be-admin-1 | [Nest] 29 - 06/22/2025, 6:47:53 PM LOG [RouterExplorer] Mapped {/get-user, GET} route +0ms
be-admin-1 | [Nest] 29 - 06/22/2025, 6:47:53 PM LOG [NestApplication] Nest application successfully started +3ms
```

Рисунок 3.4 – Результат запуску docker compose

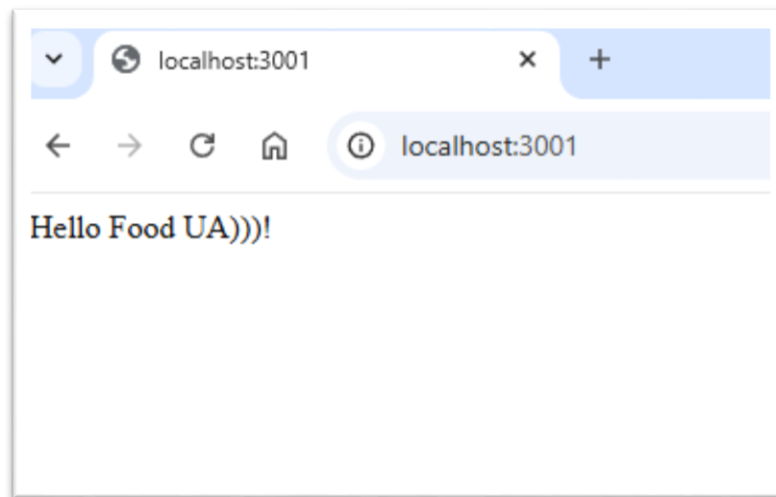


Рисунок 3.5 – Результат запуску docker compose

3.2 Створення та налаштування мережі

У сучасних хмарних проєктах ручне налаштування інфраструктури вважається неефективним та схильним до помилок. Саме тому для побудови складних і структурованих мереж все частіше використовують підхід “Infrastructure as Code” (IaC) - тобто опис усіх необхідних компонентів у вигляді коду чи конфігураційних файлів. В екосистемі AWS основним інструментом для реалізації такого підходу є CloudFormation.

CloudFormation - це сервіс, який дозволяє створювати й управляти всіма ресурсами AWS через єдиний шаблон у форматі YAML або JSON. По суті, CloudFormation - це “архітектор” і “виконавець”, який за один запуск автоматично розгортає всю інфраструктуру згідно з описаними інструкціями. Це дає можливість забезпечити автоматизацію, повторюваність і контроль над усіма ресурсами, а також значно пришвидшує процес впровадження нових середовищ.

На відміну від ручного створення кожного ресурсу через AWS Console, підхід із CloudFormation повністю позбавляє необхідності створювати все вручну - всі дії виконуються на основі чітко визначеного коду, що дозволяє мінімізувати людський фактор та забезпечити прозорість для всієї команди.

		Міценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		40

Ще однією важливою перевагою CloudFormation є легкість відтворення або видалення інфраструктури: за допомогою шаблону можна підняти ідентичну мережу у будь-якому регіоні AWS, а також швидко видалити всі ресурси одним кліком, що особливо корисно для тимчасових або експериментальних проєктів.

Усі шаблони CloudFormation можна зберігати у системах контролю версій, що спрощує командну роботу, відстеження змін та історію розгортань.

CloudFormation є сервісом AWS, підтримує найновіші можливості хмарної платформи, інтегрується з іншими інструментами IaC, такими як Terraform, і забезпечує повну відповідність best practices DevOps. [6]

Для налаштування VPC у межах даного проєкту був підготовлений окремий CloudFormation-шаблон, який дозволяє автоматизовано створити всю необхідну мережеву інфраструктуру, яку вже було спроектовано. Доцільно розглянути його основні блоки більш детально, щоб зрозуміти, як саме формується логічна структура мережі, її ізоляція й взаємозв'язки компонентів.

Блок Metadata служить для організації логічних груп параметрів у зручному інтерфейсі CloudFormation, щоб було простіше вводити значення під час створення стека.

```
4 Metadata:
5   AWS::CloudFormation::Interface:
6     ParameterGroups:
7       - Label:
8         default: "Network Configuration"
9         Parameters:
10          - Environment
11          - VPCBlock
12          - VPCBlock2
13       - Label:
14         default: "Subnets CIDR Blocks"
15         Parameters:
16          - PublicSubnetACIDR
17          - PublicSubnetBCIDR
18          - PrivateSubnetACIDR
19          - PrivateSubnetBCIDR
20          - DatabaseSubnetACIDR
21          - DatabaseSubnetBCIDR
22
```

Рисунок 3.6 – Метадата

		Міценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		41

В шаблоні оголошуються параметри, що дозволяють швидко адаптувати розгортання під різні середовища чи регіони AWS. Тут визначаються такі змінні, як назва середовища, CIDR-блоки для основної й додаткової мережі, а також CIDR для кожної підмережі. Такий підхід забезпечує гнучкість при масштабуванні інфраструктури, а також гарантує можливість повторного використання шаблону для тестових, девелоперських або продуктивних оточень.

```

23 Parameters:
24   Environment:
25     Type: String
26     Default: "foodua-vpc"
27
28   VPCBlock:
29     Type: String
30     Default: "10.0.0.0/22"
31
32   VPCBlock2:
33     Type: String
34     Default: "192.168.0.0/23"
35
36   PublicSubnetACIDR:
37     Type: String
38     Default: "10.0.0.0/24"
39     Description: "Public Subnet-A CIDR"
40

```

Рисунок 3.7 – Параметри шаблону

Основну частину шаблону складають блоки ресурсів, які відповідають за створення усіх мережевих компонентів. Описується створення самої VPC із заданими параметрами, підключення додаткового CIDR-блоку, а також організація шести підмереж: двох публічних, двох приватних і двох ізольованих під базу даних. Підмережі розподілені по різних зонах доступності, що дозволяє досягти високої відмовостійкості інфраструктури навіть у випадку виходу з ладу одного з дата-цен-

трів. Далі в шаблоні створюються Internet Gateway для забезпечення виходу в Інтернет та NAT Gateway для приватних підмереж, що дозволяє захистити внутрішні сервіси від прямого доступу, але водночас надати їм можливість отримувати оновлення чи взаємодіяти з зовнішніми ресурсами.

Завдяки використанню окремих маршрутних таблиць для кожного типу підмереж та їх асоціації з відповідними підмережами, досягається чітке розмежування потоків трафіку та ізоляція критичних компонентів, таких як база даних.

```

66 Resources:
67   VPC:
68     Type: AWS::EC2::VPC
69     Properties:
70       CidrBlock: !Ref VPCBlock
71       EnableDnsSupport: true
72       EnableDnsHostnames: true
73       Tags:
74         - Key: Name
75           Value: !Ref Environment
76
77   VPCCidrBlock2:
78     Type: "AWS::EC2::VPCCidrBlock"
79     DependsOn: VPC
80     DeletionPolicy: Delete
81     Properties:
82       CidrBlock: !Ref VPCBlock2
83       VpcId: !Ref VPC
84
85   GatewayInternet:
86     Type: "AWS::EC2::InternetGateway"
87     Properties:
88       Tags:
89         - Key: Name
90           Value: !Ref Environment
91
92   GatewayAttachmentInternet:
93     Type: "AWS::EC2::VPCGatewayAttachment"
94     Properties:
95       VpcId: !Ref VPC
96       InternetGatewayId: !Ref GatewayInternet
97

```

Рисунок 3.8 – Ресурси

Завершується шаблон визначенням вихідних даних - тобто ідентифікаторів створених ресурсів, які в подальшому використовуються для налаштування наступних етапів інфраструктури: підключення Kubernetes-кластера, створення RDS або розгортання сервісів.

```

304 Outputs:
305   VPC:
306     Description: ID for the VPC
307     Value: !Ref VPC
308     Export:
309       Name: !Join ["-", [!Ref "Environment", "VPC"]]
310
311   VPCBlock1:
312     Description: Primary CIDR block for the VPC
313     Value: !GetAtt VPC.CidrBlock
314     Export:
315       Name: !Join ["-", [!Ref "Environment", "CIDR1"]]
316
317   VPCBlock2:
318     Description: Secondary CIDR block for the VPC
319     Value: !Ref VPCBlock2
320     Export:
321       Name: !Join ["-", [!Ref "Environment", "CIDR2"]]
322
323   PublicA:
324     Description: ID for Public Subnet A
325     Value: !Ref PublicSubnetA
326     Export:
327       Name: !Join ["-", [!Ref "Environment", "PublicSubnetA"]]
328

```

Рисунок 3.9 – Вихідні дані

Для запуску створення стека достатньо виконати команду `aws cloudformation create-stack`

```

PS E:\Work\Projects\any-teller\food-mn-app> aws cloudformation create-stack --stack-name foodua-vpc --template-body file://./aws/foodua-vpc.yaml
{
  "StackId": "arn:aws:cloudformation:eu-north-1:332149187237:stack/foodua-vpc/231ca5c0-4faa-11f0-848e-0a051f9933cf"
}
❖ PS E:\Work\Projects\any-teller\food-mn-app>

```

Рисунок 3.10 – Створення VPC

Subnets (6) Info						
Find subnets by attribute or tag						
VPC: vpc-0bf62eb6a63bdc8b7 X Clear filters						
☐	Name	Subnet ID	State	VPC	Block Public...	IPv4 CIDR
☐	foodua-vpc-Database-B	subnet-04c99b423c4ea266b	Available	vpc-0bf62eb6a63bdc8b7 food...	Off	192.0.1.0/24
☐	foodua-vpc-Public-A	subnet-0e83143b8e372611b	Available	vpc-0bf62eb6a63bdc8b7 food...	Off	10.0.0.0/24
☐	foodua-vpc-Public-B	subnet-04a09b3da19a46e86	Available	vpc-0bf62eb6a63bdc8b7 food...	Off	10.0.1.0/24
☐	foodua-vpc-Database-A	subnet-0972ffb099cfb1a3b	Available	vpc-0bf62eb6a63bdc8b7 food...	Off	192.0.0.0/24
☐	foodua-vpc-Private-B	subnet-0ce3940912b1c1e26	Available	vpc-0bf62eb6a63bdc8b7 food...	Off	10.0.3.0/24
☐	foodua-vpc-Private-A	subnet-00122983533e9632e	Available	vpc-0bf62eb6a63bdc8b7 food...	Off	10.0.2.0/24

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		44

Рисунок 3.11 – Створені підмережі

Name	NAT gateway ID	Connectivity...	State	State message	Primary public I...	Primary private I...	Primary network...	VPC
NATGateway-B	nat-07c4277e0349b5c5e	Public	Available	–	51.20.242.50	10.0.1.234	eni-04295c9b0fc639...	vpc-0bf62eb6a63bdc8b7 / foo...
NATGateway-A	nat-0316402335bc7f897	Public	Available	–	16.16.113.52	10.0.0.88	eni-0c218eb246082...	vpc-0bf62eb6a63bdc8b7 / foo...

Рисунок 3.11 – Створені NAT шлюзи

3.3 Створення EKS кластеру

Для розгортання Kubernetes-кластера у підготовленій VPC використовується інструмент eksctl, що підтримує декларативний підхід на основі окремого YAML-файлу конфігурації. Це дозволяє чітко визначити всі параметри кластера, обрану VPC та сабнети, а також описати кілька node group - наприклад, для front-end і back-end робочих нод. [\[9\]](#) [\[10\]](#)

```

aws > eks-config.yaml
1  apiVersion: eksctl.io/v1alpha5
2  kind: ClusterConfig
3
4  metadata:
5    name: foodua-cluster
6    region: eu-north-1
7
8  vpc:
9    id: "vpc-0bf62eb6a63bdc8b7"
10   subnets:
11     public:
12       eu-north-1a: { id: subnet-0e83143b8e372611b }
13       eu-north-1b: { id: subnet-04a09b3da19a46e86 }
14     private:
15       eu-north-1a: { id: subnet-00122983533e9632e }
16       eu-north-1b: { id: subnet-0ce3940912b1c1e26 }
17
18  nodeGroups:
19    - name: fe-group
20      instanceType: t3.micro
21      desiredCapacity: 3
22      minSize: 1
23      maxSize: 3
24      privateNetworking: true
25
26    - name: be-group
27      instanceType: t3.micro
28      desiredCapacity: 3
29      minSize: 1
30      maxSize: 4
31      privateNetworking: true
32

```

Рисунок 3.12 – Файл конфігурації EKS кластеру

Для запуску створення кластера на основі даного файлу використовується команда `eksctl create cluster`

```

PS E:\Work\Projects\any-teller\food-mr-app> eksctl create cluster -f ./aws/eks-config.yaml
2025-06-23 00:44:21 [i] eksctl version 0.210.0
2025-06-23 00:44:21 [i] using region eu-north-1
2025-06-23 00:44:21 [!] Amazon EKS will no longer publish EKS-optimized Amazon Linux 2 (AL2) AMIs after November 26th, 2025. Additionally, Kubernetes version 1.32 is the last version for
rocket based AMIs. The default AMI family when creating clusters and nodegroups in Eksctl will be changed to AL2023 in the future.
2025-06-23 00:44:22 [✓] using existing VPC (vpc-0bf62eb6a63bdc8b7) and subnets (private:map[eu-north-1a:{subnet-00122983533e9632e eu-north-1a 10.0.2.0/24 0 } eu-north-1b:{subnet-0ce3940912b1c1e26
rth-1b:{subnet-04a09b3da19a46e86 eu-north-1b 10.0.1.0/24 0 }])
2025-06-23 00:44:22 [!] custom VPC/subnets will be used; if resulting cluster doesn't function as expected, make sure to review the configuration of VPC/subnets
2025-06-23 00:44:22 [i] nodegroup "fe-group" will use "ami-0300974632851ca92" [AmazonLinux2/1.32]
2025-06-23 00:44:22 [i] nodegroup "be-group" will use "ami-0300974632851ca92" [AmazonLinux2/1.32]
2025-06-23 00:44:23 [i] using Kubernetes version 1.32
2025-06-23 00:44:23 [i] creating EKS cluster "foodua-cluster" in "eu-north-1" region with un-managed nodes
2025-06-23 00:44:23 [i] 2 nodegroups (be-group, fe-group) were included (based on the include/exclude rules)
2025-06-23 00:44:23 [i] will create a CloudFormation stack for cluster itself and 2 nodegroup stack(s)
2025-06-23 00:44:23 [i] if you encounter any issues, check CloudFormation console or try 'eksctl utils describe-stacks --region=eu-north-1 --cluster=foodua-cluster'
2025-06-23 00:44:23 [i] Kubernetes API endpoint access will use default of {publicAccess=true, privateAccess=false} for cluster "foodua-cluster" in "eu-north-1"
2025-06-23 00:44:23 [i] CloudWatch logging will not be enabled for cluster "foodua-cluster" in "eu-north-1"
2025-06-23 00:44:23 [i] you can enable it with 'eksctl utils update-cluster-logging --enable-types={SPECIFY-YOUR-LOG-TYPES-HERE (e.g. all)} --region=eu-north-1 --cluster=foodua-cluster'
2025-06-23 00:44:23 [i] default addons vpc-cni, kube-proxy, coredns, metrics-server were not specified, will install them as EKS addons
2025-06-23 00:44:23 [i]
%2 sequential tasks: { create cluster control plane "foodua-cluster",

```

Рисунок 3.13 – Створення EKS кластеру

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		46

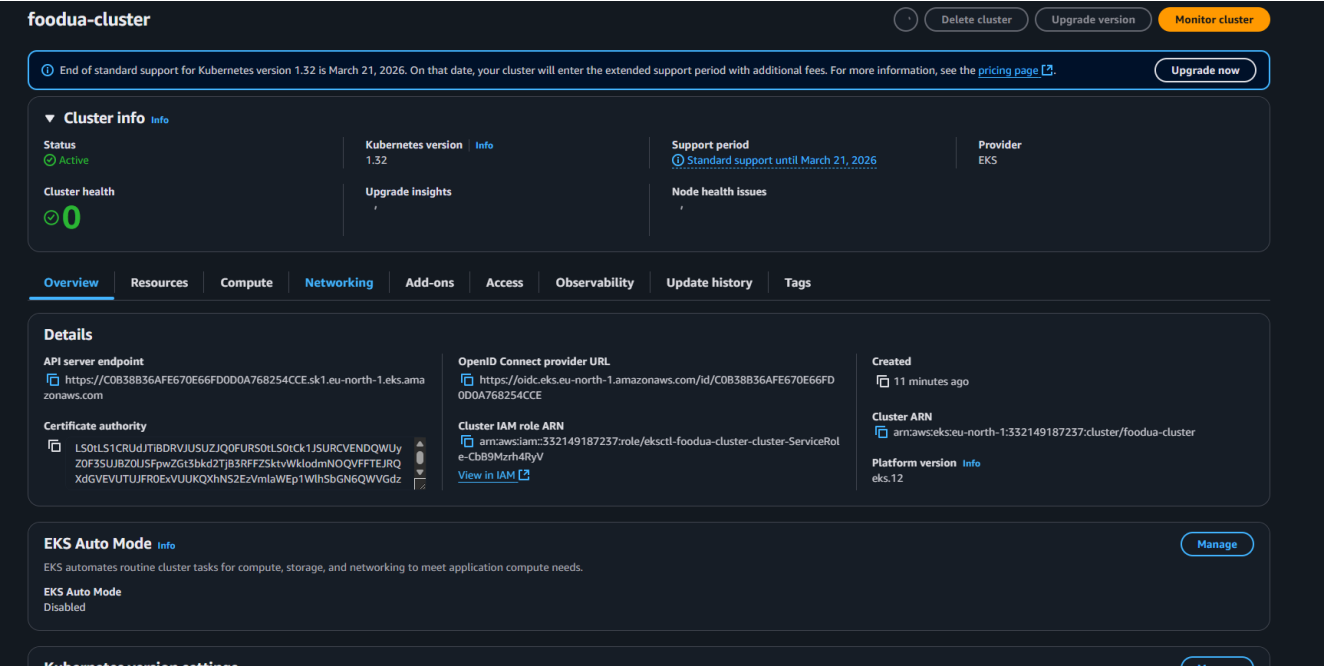


Рисунок 3.14 – Готовий кластер

Після створення та запуску Kubernetes-кластера основна взаємодія з ним відбувається за допомогою утиліти `kubectl`. Це офіційний клієнтський інструмент командного рядка, який дозволяє керувати всіма ресурсами кластера: розгортати додатки, моніторити їхній стан, налаштовувати мережеві правила, оновлювати чи видаляти компоненти, а також отримувати детальну інформацію про поди, ноди, сервіси, секрети й багато іншого.

`kubectl` працює з кластером через Kubernetes API Server, використовуючи налаштування автентифікації у файлі `kubeconfig`, який автоматично створюється або оновлюється під час налаштування кластера

```
PS E:\Work\Projects\any-teller\food-mr-app> kubectl get nodes
NAME                                STATUS    ROLES    AGE    VERSION
ip-10-0-2-119.eu-north-1.compute.internal Ready    <none>   3m32s  v1.32.3-eks-473151a
ip-10-0-2-193.eu-north-1.compute.internal Ready    <none>   3m26s  v1.32.3-eks-473151a
ip-10-0-2-80.eu-north-1.compute.internal Ready    <none>   3m35s  v1.32.3-eks-473151a
ip-10-0-3-188.eu-north-1.compute.internal Ready    <none>   3m26s  v1.32.3-eks-473151a
ip-10-0-3-25.eu-north-1.compute.internal Ready    <none>   3m28s  v1.32.3-eks-473151a
ip-10-0-3-68.eu-north-1.compute.internal Ready    <none>   3m33s  v1.32.3-eks-473151a
PS E:\Work\Projects\any-teller\food-mr-app>
```

Рисунок 3.15 – Створені робочі вузли

Для ізоляції середовищ (dev, staging, prod) та логічного розділення додатків одразу створюємо необхідні namespaces

```
PS E:\Work\Projects\any-teller\food-mr-app> kubectl create namespace dev
namespace/dev created
PS E:\Work\Projects\any-teller\food-mr-app> kubectl create namespace staging
namespace/staging created
PS E:\Work\Projects\any-teller\food-mr-app> kubectl create namespace prod
namespace/prod created
PS E:\Work\Projects\any-teller\food-mr-app> 
```

Рисунок 3.16 – Створення середовищ

Для забезпечення безпечного та контрольованого доступу до кластеру EKS використовується поєднання механізмів AWS IAM та вбудованої у Kubernetes системи ролей і прав RBAC

Для інтеграції IAM-користувача до Kubernetes-кластера EKS потрібно оновити об'єкт ConfigMap із назвою aws-auth у просторі імен kube-system.

У цьому ConfigMap прописується мапінг між IAM-обліковим записом і Kubernetes-групою, яка потім отримує відповідні права через RBAC.

```
aws > aws-auth.yaml
1  apiVersion: v1
2  kind: ConfigMap
3  metadata:
4    name: aws-auth
5    namespace: kube-system
6  data:
7    mapUsers: |
8    - userarn: arn:aws:iam::332149187237:user/foodua_deployer
9      username: foodua_deployer
10     groups:
11     - ci-deployer
12 
```

Рисунок 3.17 – Додавання IAM-користувача до kube-system

Щоб обмежити права цього користувача лише можливістю виконувати ко-

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		48

манду rollout restart для деплойментів у просторах dev, staging, prod, налаштовуються відповідні RoleBinding для кожного namespace. Тут використовується група ci-deployer, якій призначається роль deployment-restart.

```
aws > role-bindings.yaml
1  apiVersion: rbac.authorization.k8s.io/v1
2  kind: RoleBinding
3  metadata:
4    name: deployment-restart-binding
5    namespace: dev
6  subjects:
7    - kind: Group
8      name: ci-deployer
9      apiGroup: rbac.authorization.k8s.io
10 roleRef:
11   kind: ClusterRole
12   name: deployment-restart
13   apiGroup: rbac.authorization.k8s.io
14
15 ---
16 apiVersion: rbac.authorization.k8s.io/v1
17 kind: RoleBinding
18 metadata:
19   name: deployment-restart-binding
20   namespace: staging
21 subjects:
22   - kind: Group
23     name: ci-deployer
24     apiGroup: rbac.authorization.k8s.io
25 roleRef:
26   kind: ClusterRole
27   name: deployment-restart
28   apiGroup: rbac.authorization.k8s.io
29
30 ---
31 apiVersion: rbac.authorization.k8s.io/v1
32 kind: RoleBinding
33 metadata:
34   name: deployment-restart-binding
35   namespace: prod
36 subjects:
37   - kind: Group
38     name: ci-deployer
39     apiGroup: rbac.authorization.k8s.io
40 roleRef:
41   kind: ClusterRole
42   name: deployment-restart
43   apiGroup: rbac.authorization.k8s.io
44
```

Рисунок 3.18 – Призначення ролей до групи

		Міценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		49

Для застосування змін потрібно застосувати команду `kubectl apply`

```
PS E:\Work\Projects\any-teller\food-mr-app> kubectl apply -f ./aws/aws-auth.yaml
configmap/aws-auth created
PS E:\Work\Projects\any-teller\food-mr-app> kubectl apply -f ./aws/role-bindings
error: the path "./aws/role-bindings" does not exist
PS E:\Work\Projects\any-teller\food-mr-app> kubectl apply -f ./aws/role-bindings.yaml
rolebinding.rbac.authorization.k8s.io/deployment-restart-binding created
rolebinding.rbac.authorization.k8s.io/deployment-restart-binding created
rolebinding.rbac.authorization.k8s.io/deployment-restart-binding created
PS E:\Work\Projects\any-teller\food-mr-app>
```

Рисунок 3.19 – Застосування YAML-маніфестів

Наступним кроком було налаштування Ingress Controller. Ingress Controller - це спеціальний компонент Kubernetes, який дозволяє організувати доступ до додатків, розгорнутих у кластері, через єдину точку входу. У базовому Kubernetes кожен сервіс типу LoadBalancer або NodePort може отримувати зовнішній трафік напряму, проте це неефективно, якщо додатків багато: на кожен сервіс доведеться створювати окремий балансувальник або відкривати порти вручну. Ingress Controller вирішує цю проблему, виступаючи у ролі “розумного шлюзу” - він приймає зовнішній HTTP/HTTPS-трафік, аналізує його і відповідно до визначених правил спрямовує запити до потрібного сервісу чи додатку в кластері.

У хмарних середовищах зазвичай використовують спеціалізовані Ingress Controllers, які інтегруються з балансувальниками навантаження провайдера. Для Amazon EKS найпопулярніший варіант - AWS Load Balancer Controller, який автоматично створює балансувальники типу Application Load Balancer і працює у зв'язці з Ingress-ресурсами Kubernetes. [\[13\]](#)

Щоб забезпечити безпечну інтеграцію сервісних акаунтів Kubernetes із IAM-ролями AWS, необхідно створити OIDC-провайдер і зв'язати його з кластером EKS. OIDC - це протокол автентифікації, який дозволяє Kubernetes-кластеру взаємодіяти з AWS IAM для делегування прав доступу контейнерам.

```
PS E:\Work\Projects\any-teller\food-mr-app> eksctl utils associate-iam-oidc-provider --cluster=foodua-cluster --approve
2025-06-23 02:03:42 [i] will create IAM Open ID Connect provider for cluster "foodua-cluster" in "eu-north-1"
2025-06-23 02:03:43 [✓] created IAM Open ID Connect provider for cluster "foodua-cluster" in "eu-north-1"
PS E:\Work\Projects\any-teller\food-mr-app>
```

Рисунок 3.20 – Створення OIDC

		Мищенко М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				50
Змн.	Арк.	№ докум.	Підпис	Дата		

Після підключення OIDC-провайдера необхідно створити IAM-політику, яка визначає дозволи для майбутнього Ingress Controller. Політика створюється на основі JSON-файлу, який було завантажено з офіційної документації AWS.

```
C:\Users\kolya>curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.13.0/docs/install/iam_policy.json
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed

100 8955  100 8955    0     0  33914      0 --:--:-- --:--:-- --:--:-- 34179

C:\Users\kolya>aws iam create-policy --policy-name AWSLoadBalancerControllerIAMPolicy --policy-document file://iam_policy.json

An error occurred (EntityAlreadyExists) when calling the CreatePolicy operation: A policy called AWSLoadBalancerControllerIAMPolicy already exists. Duplicate names are not allowed.

C:\Users\kolya>
```

Рисунок 3.21 – Створення IAM-політики

Далі іде створення сервісного акаунта Kubernetes у просторі імен kube-system, до якого буде прикріплено цю політику. Це потрібно для того, щоб Ingress Controller міг безпечно отримувати доступ до необхідних ресурсів AWS.

```
PS E:\Work\Projects\any-teller\food-mr-app> eksctl create iamserviceaccount --cluster=foodua-cluster --namespace=kube-system --name=aws-load-balancer-controller --attach-policy-arn=arn:aws:iam::33214918723:policy/iam-policy
2025-06-23 02:21:17 [i] 1 iamserviceaccount (kube-system/aws-load-balancer-controller) was included (based on the include/exclude rules)
2025-06-23 02:21:17 [i] metadata of serviceaccounts that exist in Kubernetes will be updated, as --override-existing-serviceaccounts was set
2025-06-23 02:21:17 [i] 1 task: {
  2 sequential sub-tasks: {
    create IAM role for serviceaccount "kube-system/aws-load-balancer-controller",
    create serviceaccount "kube-system/aws-load-balancer-controller",
  }
} 2025-06-23 02:21:17 [i] building iamserviceaccount stack "eksctl-foodua-cluster-addon-iamserviceaccount-kube-system-aws-load-balancer-controller"
2025-06-23 02:21:17 [i] deploying stack "eksctl-foodua-cluster-addon-iamserviceaccount-kube-system-aws-load-balancer-controller"
2025-06-23 02:21:48 [i] waiting for CloudFormation stack "eksctl-foodua-cluster-addon-iamserviceaccount-kube-system-aws-load-balancer-controller"
2025-06-23 02:21:48 [i] waiting for CloudFormation stack "eksctl-foodua-cluster-addon-iamserviceaccount-kube-system-aws-load-balancer-controller"
2025-06-23 02:21:48 [i] created serviceaccount "kube-system/aws-load-balancer-controller"
PS E:\Work\Projects\any-teller\food-mr-app>
```

Рисунок 3.22 – Створення сервісного акаунту

Далі додавання репозиторію Ingress Controller, для цього зручно використувати менеджер пакетів Helm.

```
PS E:\Work\Projects\any-teller\food-mr-app> helm repo add eks https://aws.github.io/eks-charts
"eks" already exists with the same configuration, skipping
PS E:\Work\Projects\any-teller\food-mr-app>
PS E:\Work\Projects\any-teller\food-mr-app> helm repo update eks
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "eks" chart repository
Update Complete. ✨Happy Helming!✨
PS E:\Work\Projects\any-teller\food-mr-app>
```

Рисунок 3.23 – Додавання репозиторію Ingress Controller

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		51

Після додавання репозиторію Helm необхідно встановити AWS Load Balancer Controller у кластер Kubernetes. [\[14\]](#)

```
PS E:\Work\Projects\any-teller\food-mr-app> helm install aws-load-balancer-controller eks/aws-load-balancer-controller -n kube-system --set clusterName=foodua-cluster --set
NAME: aws-load-balancer-controller
LAST DEPLOYED: Mon Jun 23 02:31:18 2025
NAMESPACE: kube-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
AWS Load Balancer controller installed!
PS E:\Work\Projects\any-teller\food-mr-app> |
```

Рисунок 3.24 – Встановлення Ingress Controller

Тепер можна перевірити чи все зроблено правильно

```
PS E:\Work\Projects\any-teller\food-mr-app> kubectl get deployment -n kube-system aws-load-balancer-controller
NAME                                READY  UP-TO-DATE  AVAILABLE  AGE
aws-load-balancer-controller        2/2    2           2          66s
PS E:\Work\Projects\any-teller\food-mr-app> |
```

Рисунок 3.25 – Перевірка встановлення Ingress Controller

Для розгортання кожного додатку у кластері Kubernetes я використовую окрему папку з набором YAML-маніфестів. Для кожного сервісу описані наступні основні компоненти:

configmap.yaml - зберігає неконфіденційні налаштування додатку

```
apps > be-admin > k8s > dev > configmap.yaml
1  apiVersion: v1
2  kind: ConfigMap
3  metadata:
4    name: be-admin-config
5    namespace: dev
6  data:
7    DB_DIALECT: 'postgres'
8    DB_HOST: 'foodua-dev-db.cfcko0u6qcbk.eu-north-1.rds.amazonaws.com'
9    DB_PORT: '5432'
10
```

Рисунок 3.26 – configmap.yaml

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				52
Змн.	Арк.	№ докум.	Підпис	Дата		

deployment.yaml - описує основний об'єкт Deployment, який визначає, який контейнер і скільки його екземплярів потрібно розгорнути

```

1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: be-admin
5    namespace: dev
6    labels:
7      app: be-admin
8  spec:
9    selector:
10     matchLabels:
11       app: be-admin
12    template:
13     metadata:
14       labels:
15         app: be-admin
16     spec:
17       nodeSelector:
18         alpha.eksctl.io/nodegroup-name: be-group
19       containers:
20         - name: be-admin
21           image: 332149187237.dkr.ecr.eu-north-1.amazonaws.com/foodua/be-admin:dev
22           imagePullPolicy: Always
23           ports:
24             - containerPort: 3000
25           env:
26             - name: DB_DIALECT
27               valueFrom:
28                 configMapKeyRef:
29                   name: be-admin-config
30                   key: DB_DIALECT
31             - name: DB_USERNAME
32               valueFrom:
33                 secretKeyRef:
34                   name: db-credentials

```

Рисунок 3.27 – deployment.yaml

hpa.yaml - конфігурація Horizontal Pod Autoscaler, що дозволяє автоматично масштабувати кількість подів залежно від навантаження

```

1  apiVersion: autoscaling/v2
2  kind: HorizontalPodAutoscaler
3  metadata:
4    name: be-admin-hpa
5    namespace: dev
6  spec:
7    scaleTargetRef:
8      apiVersion: apps/v1
9      kind: Deployment
10     name: be-admin
11   minReplicas: 1
12   maxReplicas: 3
13   metrics:
14     - type: Resource
15       resource:
16         name: cpu
17         target:
18           type: Utilization
19           averageUtilization: 80
20     - type: Resource
21       resource:
22         name: memory
23         target:
24           type: Utilization
25           averageUtilization: 90

```

Рисунок 3.28 – hpa.yaml

		Міценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				53
Змн.	Арк.	№ докум.	Підпис	Дата		

service.yaml - описує сервіс Kubernetes, який відкриває доступ до додатку всередині кластера.

```
apps > be-admin > k8s > dev > service.yaml
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: be-admin
5    namespace: dev
6  spec:
7    selector:
8      app: be-admin
9    ports:
10     - port: 80
11       targetPort: 3000
12     type: ClusterIP
13
```

Рисунок 3.29 – service.yaml

Для застосування даних маніфестів потрібно використати команду kubectl apply

```
PS E:\Work\Projects\any-teller\food-mr-app> kubectl apply -f ./apps/be-admin/k8s/dev
configmap/be-admin-config created
deployment.apps/be-admin created
horizontalpodautoscaler.autoscaling/be-admin-hpa created
service/be-admin created
PS E:\Work\Projects\any-teller\food-mr-app>
```

Рисунок 3.30 – Застосування YAML-маніфестів

Окремо для чутливих налаштувань використовуються Kubernetes Secrets.

```
PS E:\Work\Projects\any-teller\food-mr-app> kubectl create secret generic db-credentials --from-literal=DB_USERNAME=dev_db_admin --from-literal=DB_PASSWORD=DBAdminpass1110DEV --from-literal=DB_DATABASE=foodua_dev_db -n dev
secret/db-credentials created
PS E:\Work\Projects\any-teller\food-mr-app>
```

Рисунок 3.31 – Створення Kubernetes Secrets

		Міценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				54
Змн.	Арк.	№ докум.	Підпис	Дата		

Тепер можна перевірити створені компоненти

```
PS E:\Work\Projects\any-teller\food-mr-app> kubectl get deploy -n dev
NAME          READY  UP-TO-DATE  AVAILABLE  AGE
be-admin      1/1    1           1          31s
PS E:\Work\Projects\any-teller\food-mr-app>
```

Рисунок 3.32 – Створений deployment

```
PS E:\Work\Projects\any-teller\food-mr-app> kubectl get svc -n dev
NAME          TYPE        CLUSTER-IP  EXTERNAL-IP  PORT(S)  AGE
be-admin      ClusterIP   172.20.146.63  <none>       80/TCP   85s
PS E:\Work\Projects\any-teller\food-mr-app>
```

Рисунок 3.33 – Створений service

```
PS E:\Work\Projects\any-teller\food-mr-app> kubectl get hpa -n dev
NAME          REFERENCE          TARGETS          MINPODS  MAXPODS  REPLICAS  AGE
be-admin-hpa  Deployment/be-admin  cpu: <unknown>/80%, memory: <unknown>/90%  1        3        1         114s
PS E:\Work\Projects\any-teller\food-mr-app>
```

Рисунок 3.34 – Створений horizontal pod autoscaling

```
be-admin-hpa  Deployment/be-admin  cpu: <unknown>/80%, memory: <unknown>/90%
PS E:\Work\Projects\any-teller\food-mr-app> kubectl get configmap -n dev
NAME          DATA  AGE
be-admin-config  3      2m39s
kube-root-ca.crt  1      99m
PS E:\Work\Projects\any-teller\food-mr-app>
```

Рисунок 3.35 – Створений configmap

3.4 Створення інших сервісів AWS

Перед створенням самої базм даних потрібно створити групу підмереж. До цієї групи будуть входити спеціальні, ізольовані підмережі які ми створювали для баз даних

Create DB subnet group

To create a new subnet group, give it a name and a description, and choose an existing VPC. You will then be able to add subnets related to that VPC.

Subnet group details

Name
You won't be able to modify the name after your subnet group has been created.
db-subnet-group
Must contain from 1 to 255 characters. Alphanumeric characters, spaces, hyphens, underscores, and periods are allowed.

Description
subnet group for database

VPC
Choose a VPC Identifier that corresponds to the subnets you want to use for your DB subnet group. You won't be able to choose a different VPC Identifier after your subnet group has been created.
foodua-vpc (vpc-0bf62eb6a63bdc8b7)
6 Subnets, 2 Availability Zones

Add subnets

Availability Zones
Choose the Availability Zones that include the subnets you want to add.
Choose an availability zone
eu-north-1a X eu-north-1b X eu-north-1c X

Subnets
Choose the subnets that you want to add. The list includes the subnets in the selected Availability Zones.
Select subnets
foodua-vpc-Database-B Subnet ID: subnet-04c99b423c4ea266b CIDR: 192.0.1.0/24 foodua-vpc-Database-A Subnet ID: subnet-0972ffb099cfb1a3b CIDR: 192.0.0.0/24

For Multi-AZ DB clusters, you must select 3 subnets in 3 different Availability Zones.

Subnets selected (2)

Availability zone	Subnet name	Subnet ID	CIDR block
eu-north-1b	foodua-vpc-Database-B	subnet-04c99b423c4ea266b	192.0.1.0/24
eu-north-1a	foodua-vpc-Database-A	subnet-0972ffb099cfb1a3b	192.0.0.0/24

Cancel Create

Рисунок 3.36 – Створення групи підмереж

Наступний крок створення самої бази даних. Для даного проєкту було обрано PostgreSQL базу даних.

Choose a database creation method

☒ **Standard create**
You set all of the configuration options, including ones for availability, security, backups, and maintenance.

☐ **Easy create**
Use recommended best-practice configurations. Some configuration options can be changed after the database is created.

Engine options

Engine type [Info](#)

☐ Aurora (MySQL Compatible) 

☐ Aurora (PostgreSQL Compatible) 

☐ MySQL 

☒ PostgreSQL 

☐ MariaDB 

☐ Oracle 

☐ Microsoft SQL Server 

☐ IBM Db2 

Engine version [Info](#)
View the engine versions that support the following database features.

Hide filters

☐ Show only versions that support the Multi-AZ DB cluster [Info](#)
Create a Multi-AZ DB cluster with one primary DB instance and two readable standby DB instances. Multi-AZ clusters provide up to 2x faster transaction commit latency and automatic failover in typically under 35 seconds.

Engine version
PostgreSQL 17.4-R1

☐ Enable RDS Extended Support [Info](#)
Amazon RDS Extended Support is a paid offering [2]. By selecting this option, you consent to being charged for this offering if you are running your database major version past the RDS end of standard support date for that version. Check the end of standard support date for your major version in the RDS for PostgreSQL documentation [2].

Рисунок 3.37 – Створення бази даних

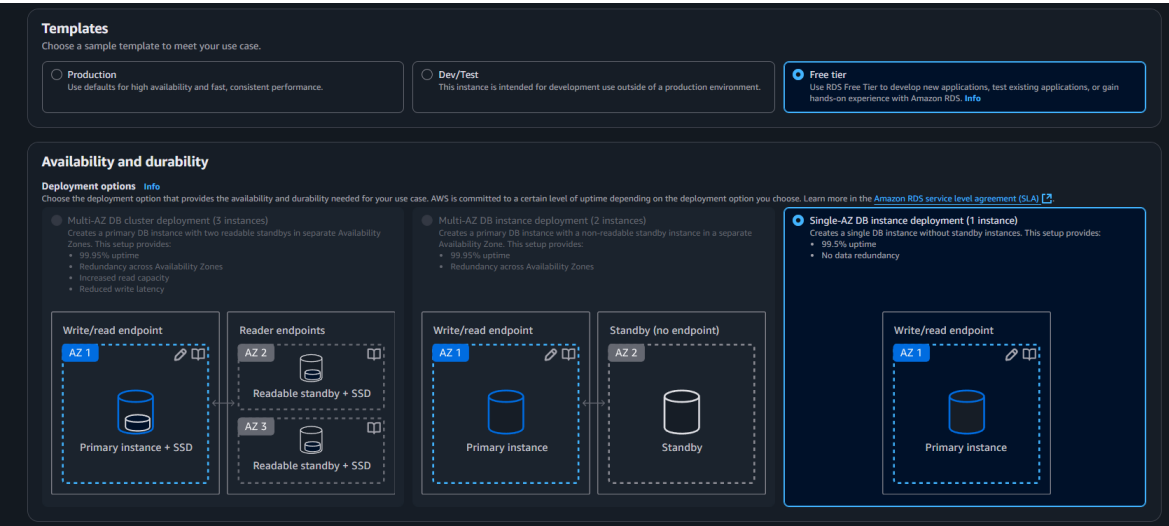


Рисунок 3.38 – Створення бази даних

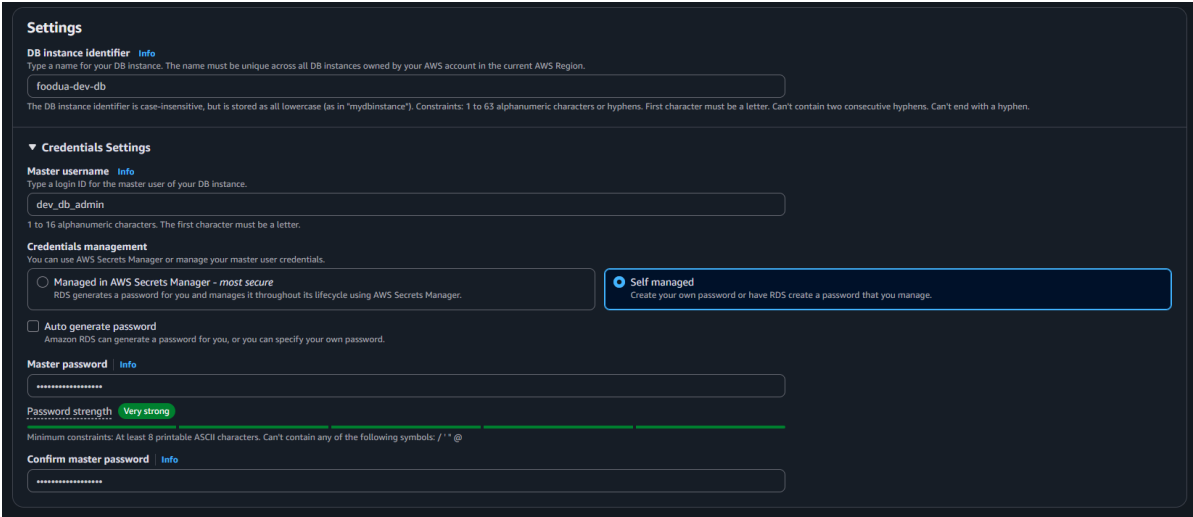


Рисунок 3.39 – Створення бази даних

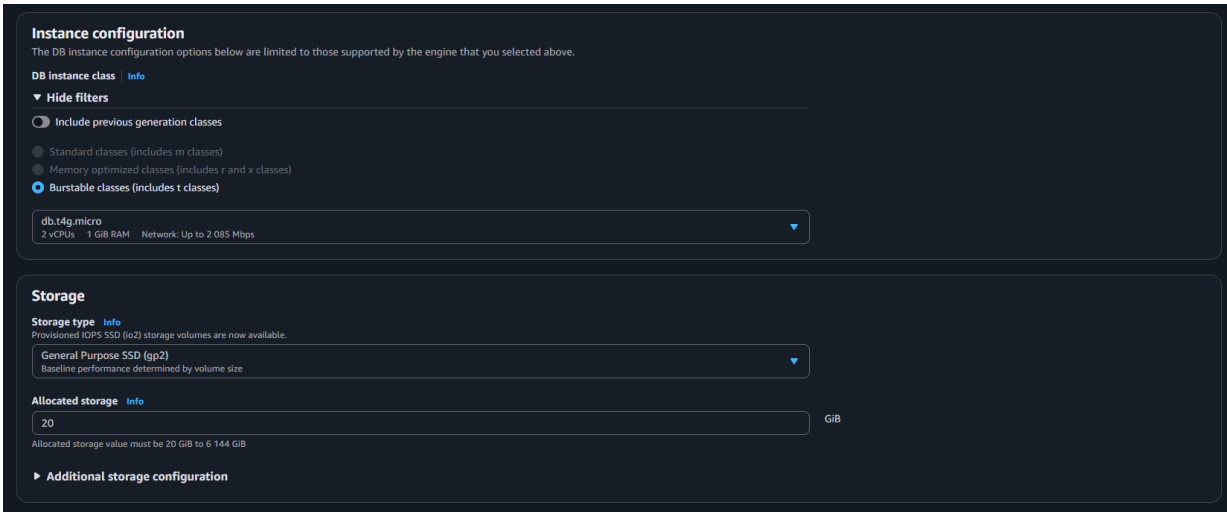


Рисунок 3.40 – Створення бази даних

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				57
Змн.	Арк.	№ докум.	Підпис	Дата		

Connectivity Info

Compute resource
Choose whether to set up a connection to a compute resource for this database. Setting up a connection will automatically change connectivity settings so that the compute resource can connect to this database.

☒ **Don't connect to an EC2 compute resource**
Don't set up a connection to a compute resource for this database. You can manually set up a connection to a compute resource later.

☐ **Connect to an EC2 compute resource**
Set up a connection to an EC2 compute resource for this database.

Network type Info
To use dual-stack mode, make sure that you associate an IPv6 CIDR block with a subnet in the VPC you specify.

☒ **IPv4**
Your resources can communicate only over the IPv4 addressing protocol.

☐ **Dual-stack mode**
Your resources can communicate over IPv4, IPv6, or both.

Virtual private cloud (VPC) Info
Choose the VPC. The VPC defines the virtual networking environment for this DB instance.

foodua-vpc (vpc-0bf62eb6a63bdc8b7)
6 Subnets, 2 Availability Zones

Only VPCs with a corresponding DB subnet group are listed.

ⓘ After a database is created, you can't change its VPC.

DB subnet group Info
Choose the DB subnet group. The DB subnet group defines which subnets and IP ranges the DB instance can use in the VPC that you selected.

db-subnet-group
2 Subnets, 2 Availability Zones

Public access Info

☐ **Yes**
RDS assigns a public IP address to the database. Amazon EC2 instances and other resources outside of the VPC can connect to your database. Resources inside the VPC can also connect to the database. Choose one or more VPC security groups that specify which resources can connect to the database.

☒ **No**
RDS doesn't assign a public IP address to the database. Only Amazon EC2 instances and other resources inside the VPC can connect to your database. Choose one or more VPC security groups that specify which resources can connect to the database.

VPC security group (firewall) Info
Choose one or more VPC security groups to allow access to your database. Make sure that the security group rules allow the appropriate incoming traffic.

☒ **Choose existing**
Choose existing VPC security groups

☐ **Create new**
Create new VPC security group

Existing VPC security groups
Choose one or more options

default ✕

Availability Zone Info
No preference

Рисунок 3.41 – Створення бази даних

▼ **Additional configuration**
Database options, encryption turned on, backup turned on, backtrack turned off, maintenance, CloudWatch Logs, delete protection turned off.

Database options

Initial database name Info

foodua_dev_db

If you do not specify a database name, Amazon RDS does not create a database.

Рисунок 3.42 – Створення бази даних

Наступний крок це створення та налаштування S3 бакету

General purpose buckets (3) Info All AWS Regions

Buckets are containers for data stored in S3.

Find buckets by name

Name	AWS Region	IAM Access Analyzer	Creation date
☐ foodua-dev	Europe (Stockholm) eu-north-1	View analyzer for eu-north-1	June 14, 2025, 17:27:56 (UTC+03:00)
☐ foodua-production	Europe (Stockholm) eu-north-1	View analyzer for eu-north-1	June 14, 2025, 17:28:43 (UTC+03:00)
☐ foodua-staging	Europe (Stockholm) eu-north-1	View analyzer for eu-north-1	June 14, 2025, 17:28:28 (UTC+03:00)

Рисунок 3.43 – Створення S3 бакетів

Для безпечної роботи з файлами у S3-бакеті було обрано приватний режим доступу, коли жоден користувач не має прямого доступу до об'єктів. Для передачі

доступу кінцевим користувачам застосовуються підписані URL з обмеженим терміном дії. Такі посилання генеруються динамічно на бекенді для кожного окремого файлу та дозволяють завантажити файл без розкриття секретних ключів чи надання надлишкових прав доступу.

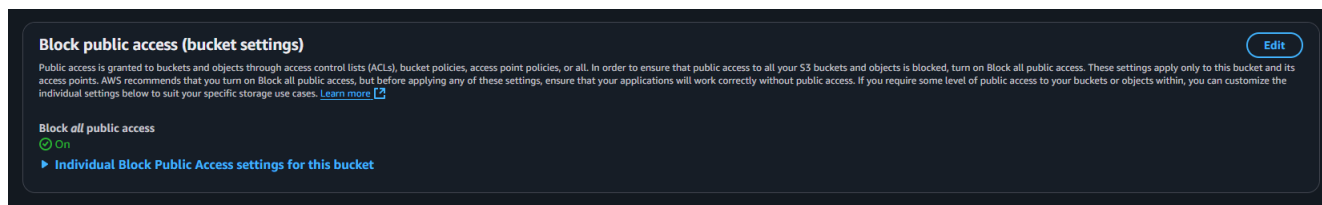


Рисунок 3.44 – Налаштування S3 бакету

Також необхідно створити репозиторії на ECR, для зберігання до отримання образів наших сервісів.

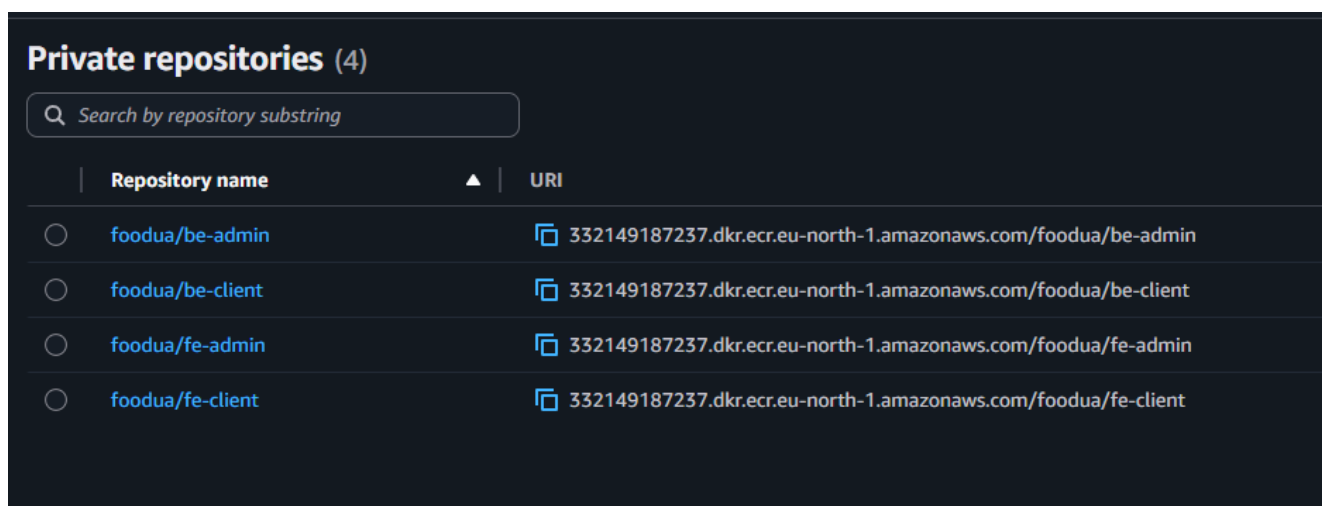


Рисунок 3.45 – Створення ECR репозиторіїв

3.5 Налаштування Ingress Controller та тестування роботи системи

Для організації доступу до різних компонентів системи у Kubernetes-кластері використовується Ingress rules, які описані у файлі `ingress.dev.yaml`. Цей маніфест визначає правила маршрутизації зовнішнього трафіку через балансувальник навантаження AWS Application Load Balancer, який було встановлено раніше, до різних сервісів, розгорнутих у просторі імен `dev`.

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				59
Змн.	Арк.	№ докум.	Підпис	Дата		

```

ws > ingress.dev.yaml
1  apiVersion: networking.k8s.io/v1
2  kind: Ingress
3  metadata:
4    name: foodua-dev-ingress
5    namespace: dev
6    annotations:
7      alb.ingress.kubernetes.io/scheme: internet-facing
8      alb.ingress.kubernetes.io/target-type: ip
9      alb.ingress.kubernetes.io/backend-protocol: HTTP
10 spec:
11   ingressClassName: alb
12   rules:
13     - host: food-ua.dev.local
14       http:
15         paths:
16           - path: /
17             pathType: Prefix
18             backend:
19               service:
20                 name: fe-client
21                 port:
22                   number: 80
23     - host: food-ua-api.dev.local
24       http:
25         paths:
26           - path: /
27             pathType: Prefix
28             backend:
29               service:
30                 name: be-client
31                 port:
32                   number: 80
33     - host: admin.food-ua.dev.local
34       http:
35         paths:
36           - path: /
37             pathType: Prefix
38             backend:
39               service:
40                 name: fe-admin
41                 port:
42                   number: 80
43     - host: admin.food-ua-api.dev.local
44       http:
45         paths:
46           - path: /
47             pathType: Prefix
48             backend:
49               service:
50                 name: be-admin
51                 port:
52                   number: 80

```

Рисунок 3.46 – ingress.dev.yaml

Далі потрібно застосувати даний маніфест

```

PS E:\Work\Projects\any-teller\food-mr-app> kubectl apply -f ./aws/ingress.dev.yaml
ingress.networking.k8s.io/foodua-dev-ingress created
PS E:\Work\Projects\any-teller\food-mr-app>

```

Рисунок 3.47 – Застосування YAML-маніфесту

		Міценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		60

Оскільки для розробки та тестування системи не використовувалися реальні публічні доменні імена, було прийнято рішення налаштувати локальні домени на власному комп'ютері. Такий підхід дозволяє імітувати роботу з реальними доменими ще до покупки чи налаштування DNS-зон у продакшн-оточенні.

Після минулих налаштувань я отримав публічну адресу.

```
PS E:\Work\Projects\any-teller\food-mr-app> kubectl get ingress -n dev
NAME                                CLASS    HOSTS
foodua-dev-ingress                 alb      food-ua.dev.local,food-ua-api.dev.local,admin.food-ua.dev.local + 1 more...
PS E:\Work\Projects\any-teller\food-mr-app>
```

NAME	CLASS	HOSTS	ADDRESS	PORTS	AGE
foodua-dev-ingress	alb	food-ua.dev.local,food-ua-api.dev.local,admin.food-ua.dev.local + 1 more...	k8s-dev-fooduade-441c311b8f-1586308479.eu-north-1.elb.amazonaws.com	80	40s

Рисунок 3.48 – Публічна адреса

Далі потрібно дізнатися IP-адресу

```
PS E:\Work\Projects\any-teller\food-mr-app> nslookup k8s-dev-fooduade-441c311b8f-1586308479.eu-north-1.elb.amazonaws.com 8.8.8.8
Server: dns.google
Address: 8.8.8.8

Non-authoritative answer:
Name:      k8s-dev-fooduade-441c311b8f-1586308479.eu-north-1.elb.amazonaws.com
Addresses: 13.49.52.248
           13.60.106.255
PS E:\Work\Projects\any-teller\food-mr-app>
```

Рисунок 3.49 – Отримання IP-адреси

Для того, щоб звернення до локальних доменів перенаправлялись на балансувальник, було додано відповідні записи у файл hosts на локальному ПК.

```
# Temp fake domains
13.49.52.248 food-ua.dev.local
13.49.52.248 food-ua-api.dev.local
13.49.52.248 admin.food-ua.dev.local
13.49.52.248 admin.food-ua-api.dev.local
# End of section
```

Рисунок 3.50 – Додавання локальних доменних імен

Тепер можемо перевірити роботу сервісу

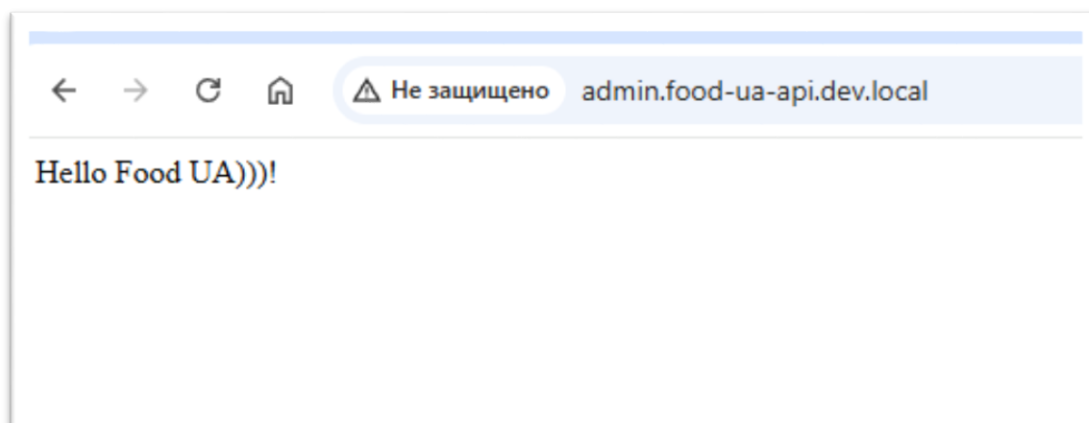


Рисунок 3.51 – Перевірка роботи сервісу

Також, для тестування підключення сервісу до бази даних в ньому було додано запити для створення та отримання юзера з бази даних.

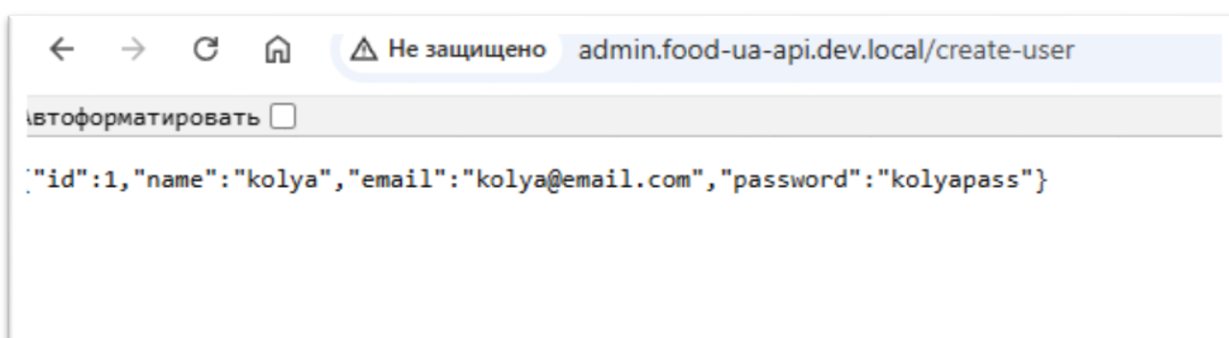


Рисунок 3.52 – Перевірка підключення сервісу до бази даних

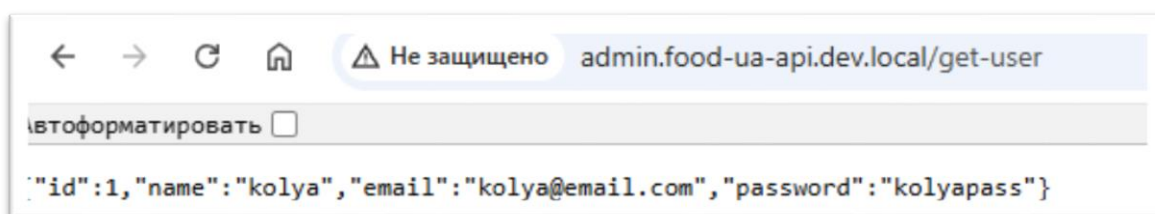


Рисунок 3.53 – Перевірка підключення сервісу до бази даних

3.6 Налаштування CI/CD

Для організації безперервної інтеграції та розгортання (CI/CD) у цьому проєкті використовується GitHub Actions. Це дозволяє автоматично збирати, публікувати та деплоїти контейнери у кластер Kubernetes після кожного пушу у потрібну гілку.

Пайплайн складається з трьох основних частин: виявлення змін у проєктах, побудова та публікація образів у ECR і деплой змін. Оскільки в проєкті використовується монорепозіторій, тому на першому етапі використовується спеціальний екшен для визначення, які з проєктів були змінені у коміті. Це дозволяє запускати наступні кроки лише для тих сервісів, які дійсно потребують оновлення, що економить час та ресурси.

```
16 jobs:
17   detect-changes:
18     runs-on: ubuntu-latest
19     outputs:
20       be_admin: ${ steps.filter.outputs.be_admin }
21       be_client: ${ steps.filter.outputs.be_client }
22       fe_admin: ${ steps.filter.outputs.fe_admin }
23       fe_client: ${ steps.filter.outputs.fe_client }
24     steps:
25       - uses: actions/checkout@v4
26
27       - name: Detect changed apps
28         id: filter
29         uses: dorny/paths-filter@v3
30         with:
31           filters: |
32             be_admin:
33               - 'apps/be-admin/**'
34             be_client:
35               - 'apps/be-client/**'
36             fe_admin:
37               - 'apps/fe-admin/**'
38             fe_client:
39               - 'apps/fe-client/**'
```

Рисунок 3.54 – Виявлення змін у проєктах

Далі, якщо виявлені зміни хоча б у одному з проєктів, виконується другий job. Спочатку налаштовується доступ до AWS через secrets і підключається до Docker Buildx. Далі відбувається аутентифікація у всіх відповідних ECR-

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		63

репозиторіях за допомогою AWS CLI. Далі для кожного сервісу, який було змінено, відбувається білд контейнерного образу та публікація його у ECR з відповідними тегами.

```

16 jobs:
41   build-and-push:
42     needs: detect-changes
43     if: >-
44       needs.detect-changes.outputs.be_admin == 'true' ||
45       needs.detect-changes.outputs.be_client == 'true' ||
46       needs.detect-changes.outputs.fe_admin == 'true' ||
47       needs.detect-changes.outputs.fe_client == 'true'
48     runs-on: ubuntu-latest
49     steps:
50       - uses: actions/checkout@v4
51
52       - name: Validate required secrets
53         run: |
54           if [ -z "${{ secrets.AWS_ACCESS_KEY_ID }}" ] || [ -z "${{ secrets.AWS_SECRET_ACCESS_KEY }}" ]; then
55             echo "Error: Required secrets are missing"
56             exit 1
57           fi
58
59       - name: Configure AWS Credentials
60         uses: aws-actions/configure-aws-credentials@v3
61         with:
62           aws-access-key-id: "${{ secrets.AWS_ACCESS_KEY_ID }}"
63           aws-secret-access-key: "${{ secrets.AWS_SECRET_ACCESS_KEY }}"
64           aws-region: "${{ env.AWS_REGION }}"
65
66       - name: Set up Docker
67         uses: docker/setup-buildx-action@v2
68
69       - name: Login to ECR
70         run: |
71           for REG in "${{ env.BE_ADMIN_ECR }}" "${{ env.BE_CLIENT_ECR }}" "${{ env.FE_ADMIN_ECR }}" "${{ env.FE_CLIENT_ECR }}"; do
72             [ -n "$REG" ] && aws ecr get-login-password \
73               --region "${{ env.AWS_REGION }}" | docker login --username AWS --password-stdin $REG
74           done
75
76       - name: Build & Push images
77         run: |
78           # be-admin
79           if [ "${{ needs.detect-changes.outputs.be_admin }}" = "true" ]; then
80             docker buildx build --platform linux/amd64 \
81               -t "${{ env.BE_ADMIN_ECR }}:latest" -t "${{ env.BE_ADMIN_ECR }}:dev" \
82               --push ./apps/be-admin
83           fi

```

Рисунок 3.55 – Білд і пуш образів до ECR

Після успішної публікації образів пайплайн оновлює kubeconfig для доступу до EKS-кластера і виконує rollout restart для відповідних деплойментів. Це дозволяє автоматично оновлювати піди у кластері лише для тих сервісів, у яких були зміни.

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				64
Змн.	Арк.	№ докум.	Підпис	Дата		

```

103   deploy:
104     needs: [detect-changes, build-and-push]
105     if: >-
106       needs.detect-changes.outputs.be_admin == 'true' ||
107       needs.detect-changes.outputs.be_client == 'true' ||
108       needs.detect-changes.outputs.fe_admin == 'true' ||
109       needs.detect-changes.outputs.fe_client == 'true'
110     runs-on: ubuntu-latest
111     steps:
112     - uses: actions/checkout@v4
113
114     - name: Configure AWS Credentials
115       uses: aws-actions/configure-aws-credentials@v3
116       with:
117         aws-access-key-id: ${ secrets.AWS_ACCESS_KEY_ID }
118         aws-secret-access-key: ${ secrets.AWS_SECRET_ACCESS_KEY }
119         aws-region: ${ env.AWS_REGION }
120
121     - name: Update kubeconfig
122       run: |
123         aws eks update-kubeconfig \
124           --region ${ env.AWS_REGION } \
125           --name ${ env.CLUSTER_NAME }
126
127     - name: Rollout restart changed deployments
128       run: |
129         if [ "${ needs.detect-changes.outputs.be_admin }" = "true" ]; then
130           kubectl rollout restart deployment/be-admin -n dev
131         fi
132         if [ "${ needs.detect-changes.outputs.be_client }" = "true" ]; then
133           kubectl rollout restart deployment/be-client -n dev
134         fi
135         if [ "${ needs.detect-changes.outputs.fe_admin }" = "true" ]; then
136           kubectl rollout restart deployment/fe-admin -n dev
137         fi
138         if [ "${ needs.detect-changes.outputs.fe_client }" = "true" ]; then
139           kubectl rollout restart deployment/fe-client -n dev
140         fi
141

```

Рисунок 3.56 – Деплой змін

Тепер можна перевірити даний пайплайн, для цього необхідно запустити зміни у гілку dev та переглянути процес у GitHub Actions.

@Injectable()	3	3	@Injectable()
export class AppService {	4	4	export class AppService {
getHello(): string {	5	5	getHello(): string {
- return 'Hello Food UA!'	✓ 6	✓ 6	+ return 'Hello Food UA!!!!'
}	7	7	}
}	8	8	}

Рисунок 3.57 – Зміни в коді проєкту

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		65

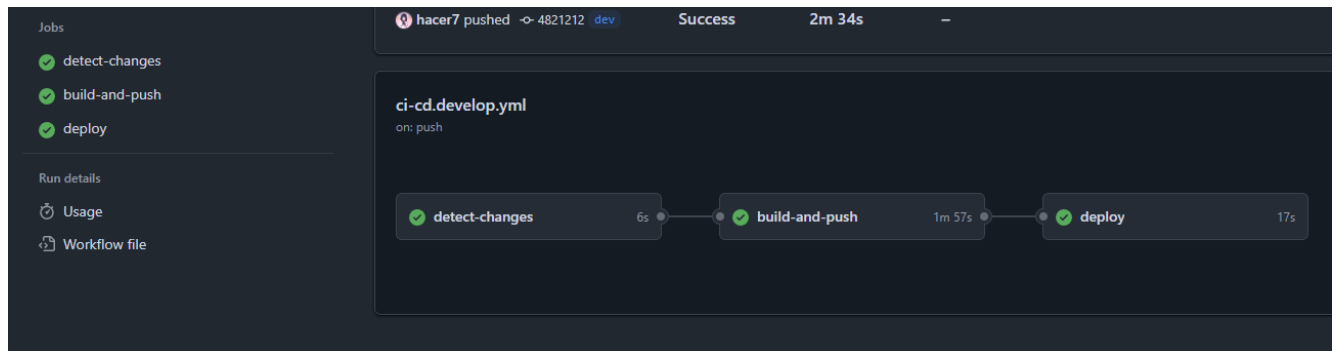


Рисунок 3.58 – Виконання пайплайну

Після того як пайплайн був успішно виконаний можемо перейти до нашого додатку та подивитися на зміни.

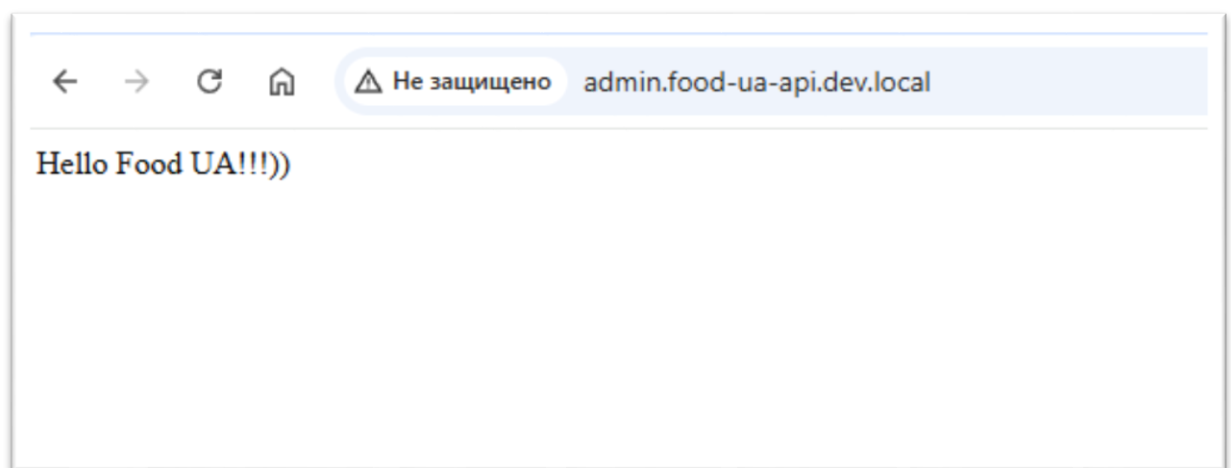


Рисунок 3.58 – Оновлений додаток

Висновки до розділу 3

У третьому розділі було детально продемонстровано процес впровадження хмарної інфраструктури для сучасного вебзастосунку з використанням AWS, Docker, Kubernetes та суміжних технологій. В рамках практичної реалізації послідовно налаштовано усі ключові компоненти інфраструктури: створено ізольовану віртуальну мережу з логічно розподіленими підмережами, інтегровано керувану реляційну базу даних за допомогою AWS RDS, налаштовано приватне файлове сховище в S3 для зберігання та обміну файлами між сервісами. Особливу увагу приділено організації безпечного доступу до ресурсів через генерацію тимчасових підписаних посилань для взаємодії із S3.

		Міценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		66

Показано, як за допомогою CloudFormation та eksctl автоматизується створення мережевої інфраструктури та кластеру Kubernetes, що значно знижує ймовірність людської помилки та спрощує масштабування проєкту у майбутньому. Усі необхідні сервіси розгорнуто у відповідних namespaces, застосовано політики доступу та ізоляції. Для забезпечення масштабованості та доступності системи впроваджено AWS Load Balancer Controller та організовано маршрутизацію трафіку через Ingress-ресурси, що дозволяє централізовано керувати зовнішніми підключеннями.

Практична частина містить також впровадження та налаштування пайплайну CI/CD на основі GitHub Actions, який автоматично виконує білд, публікацію та деплой додатків у кластер. Це дозволило мінімізувати ручні втручання та забезпечити стабільне й передбачуване оновлення системи. Результати тестування підтвердили працездатність усіх компонентів і узгодженість їх взаємодії, що є важливим показником якості впровадженої архітектури.

Загалом, виконана практична частина проєкту продемонструвала можливість побудови гнучкої, масштабованої та безпечної хмарної інфраструктури для сучасного вебзастосунку із застосуванням найкращих практик DevOps та хмарних технологій.

		Міценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		67

ВИСНОВКИ

У цій кваліфікаційній роботі було комплексно досліджено, спроектовано та реалізовано хмарну інфраструктуру для сучасного вебзастосунку із застосуванням технологій Docker, Kubernetes і сервісів AWS. У першому розділі детально проаналізовано теоретичні основи побудови хмарних рішень, розглянуто ключові переваги та особливості контейнеризації, принципи оркестрації з використанням Kubernetes, а також огляд базових сервісів AWS, необхідних для розгортання і підтримки хмарної інфраструктури. Цей розділ заклав міцну теоретичну базу для виконання практичних завдань у наступних частинах роботи.

Другий розділ присвячено етапам проектування інфраструктури, де на прикладі власного проекту було розглянуто формування ізольованої мережі VPC, структурування підмереж, налаштування маршрутних таблиць та вибір архітектури Kubernetes-кластера з урахуванням кращих практик. Окрему увагу приділено питанням безпеки, масштабованості та відмовостійкості, а також організації життєвого циклу застосунків за допомогою namespaces. Ці рішення забезпечили фундамент для надійної, масштабованої та зручно адміністрованої інфраструктури.

Третій розділ містить детальний опис практичної реалізації: розгортання всіх компонентів інфраструктури через CloudFormation і eksctl, інтеграцію з керованою базою даних AWS RDS, налаштування приватного файлового сховища S3, а також впровадження пайплайну CI/CD на базі GitHub Actions. Описано процеси автоматизації деплою, організації безпечного доступу до сервісів і тестування працездатності додатків у кластері. Під час тестування було підтверджено стабільність і взаємодію всіх компонентів системи.

У результаті виконання всіх етапів дослідження та розробки було повністю досягнуто мети роботи - створено гнучку, масштабовану і безпечну хмарну інфраструктуру для вебзастосунку з використанням сучасних технологій і найкращих практик DevOps. Результати цієї роботи можуть бути використані для подальших досліджень у сфері хмарних технологій.

		Міценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		68

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Docker – Docker guides [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.docker.com/guides/>
2. AWS – Containers vs Virtual Machines [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/compare/the-difference-between-containers-and-virtual-machines/?nc1=h_ls
3. Docker – How Compose works [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.docker.com/compose/intro/compose-application-model/>
4. ITEDU – AWS [Електронний ресурс] – Режим доступу до ресурсу: https://itedu.center/ua/blog/guides/aws-vse-shho-potribno-znati-pro-xmarnu-platformu/?srsltid=AfmBOopnj67_HMEhQGkBqqmvGI22SKMuAqt0R5mkM0UTo6wiDZjevwrw
5. AWS – What is Amazon VPC? [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html>
6. DOU – AWS CloudFormation [Електронний ресурс] – Режим доступу до ресурсу: <https://dou.ua/forums/topic/30771/>
7. Kubernetes – Що таке Kubernetes? [Електронний ресурс] – Режим доступу до ресурсу: <https://kubernetes.io/uk/docs/concepts/overview/what-is-kubernetes/>
8. Kubernetes – Kubernetes Components [Електронний ресурс] – Режим доступу до ресурсу: <https://kubernetes.io/docs/concepts/overview/components/>
9. Medium – Setup Amazon EKS cluster [Електронний ресурс] – Режим доступу до ресурсу: <https://medium.com/@mudasirhaji/setup-kubernetes-cluster-on-amazon-eks-56cbbadace04>
10. eksctl – Creating and managing clusters [Електронний ресурс] – Режим доступу до ресурсу: <https://eksctl.io/usage/creating-and-managing-clusters/>
11. AWS – What is Amazon RDS? [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Welcome.html>

		Міценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		69

12. Medium – AWS S3 Tutorial [Електронний ресурс] – Режим доступу до ресурсу: <https://medium.com/edureka/aws-s3-tutorial-d883e5cc2cf5>
13. AWS – ALB Ingress Controller [Електронний ресурс] – Режим доступу до ресурсу: <https://aws.amazon.com/ru/blogs/opensource/kubernetes-ingress-aws-alb-ingress-controller/>
14. AWS – Install AWS Load Balancer [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.aws.amazon.com/eks/latest/userguide/lbc-helm.html>
15. Medium – Amazon ECR [Електронний ресурс] – Режим доступу до ресурсу: [https://medium.com/@mudasirhaji/amazon-elastic-container-registry-ecr-simplifying-container-image-management-64da472ffaff /](https://medium.com/@mudasirhaji/amazon-elastic-container-registry-ecr-simplifying-container-image-management-64da472ffaff/)

		Міценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		70

ДОДАТКИ

		Міценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				71
Змн.	Арк.	№ докум.	Підпис	Дата		

Dockerfile be-admin та be-client

FROM node:18-alpine

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 3000

CMD ["npm", "run", "start"]

Dockerfile fe-admin та fe-client

FROM node:18 AS builder

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

RUN npm run build

EXPOSE 3000

CMD ["npm", "run", "start"]

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				72
Змн.	Арк.	№ докум.	Підпис	Дата		

docker-compose.yml

```
version: "3.9"
```

```
services:
```

```
  postgres:
```

```
    image: postgres:15
```

```
    environment:
```

```
      POSTGRES_USER: dev
```

```
      POSTGRES_PASSWORD: dev
```

```
      POSTGRES_DB: food_mr_db
```

```
    ports:
```

```
      - "5432:5432"
```

```
    volumes:
```

```
      - pgdata:/var/lib/postgresql/data
```

```
    networks:
```

```
      - food-mr-network
```

```
  be-admin:
```

```
    build: ./apps/be-admin
```

```
    ports:
```

```
      - "3001:3000"
```

```
    depends_on:
```

```
      - postgres
```

```
    environment:
```

```
      - DB_DIALECT=postgres
```

```
      - DB_USERNAME=dev
```

```
      - DB_PASSWORD=dev
```

```
      - DB_DATABASE=food_mr_db
```

```
      - DB_HOST=postgres
```

```
      - DB_PORT=5432
```

```
    command: npm run start:dev
```

```
    networks:
```

```
      - food-mr-network
```

```
  develop:
```

```
    watch:
```

```
      - action: sync
```

```
        path: ./apps/be-admin/src
```

```
        target: /app
```

```
      - action: rebuild
```

```
        path: ./apps/be-admin/package.json
```

```
  be-client:
```

```
    build: ./apps/be-client
```

```
    ports:
```

```
      - "3002:3000"
```

```
    depends_on:
```

```
      - postgres
```

```
    environment:
```

```
      - DATABASE_URL=postgres://dev:dev@postgres:5432/food_mr_db
```

```
    command: npm run start:dev
```

```
    networks:
```

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		73

- food-mr-network
develop:
watch:
- action: sync
path: ./apps/be-client/src
target: /app/src
- action: rebuild
path: ./apps/be-client/package.json

fe-admin:
build: ./apps/fe-admin
ports:
- "3003:3000"
depends_on:
- be-admin
command: npm run dev
networks:
- food-mr-network
develop:
watch:
- action: sync
path: ./apps/fe-admin
target: /app
- action: rebuild
path: ./apps/fe-admin/package.json

fe-client:
build: ./apps/fe-client
ports:
- "3004:3000"
depends_on:
- be-client
command: npm run dev
networks:
- food-mr-network
develop:
watch:
- action: sync
path: ./apps/fe-client
target: /app
- action: rebuild
path: ./apps/fe-client/package.json

volumes:
pgdata:

networks:
food-mr-network:
driver: bridge

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		74

VPC setup

AWS::TemplateFormatVersion: 2010-09-09
Description: "Network: VPC, Subnets + RouteTables, Internet + NAT Gateways"

Metadata:
AWS::CloudFormation::Interface:
ParameterGroups:
- Label:
 default: "Network Configuration"
Parameters:
- Environment
- VPCBlock
- VPCBlock2
- Label:
 default: "Subnets CIDR Blocks"
Parameters:
- PublicSubnetACIDR
- PublicSubnetBCIDR
- PrivateSubnetACIDR
- PrivateSubnetBCIDR
- DatabaseSubnetACIDR
- DatabaseSubnetBCIDR

Parameters:
Environment:
 Type: String
 Default: "foodua-vpc"

VPCBlock:
 Type: String
 Default: "10.0.0.0/22"

VPCBlock2:
 Type: String
 Default: "192.0.0.0/23"

PublicSubnetACIDR:
 Type: String
 Default: "10.0.0.0/24"
 Description: "Public Subnet-A CIDR"

PublicSubnetBCIDR:
 Type: String
 Default: "10.0.1.0/24"
 Description: "Public Subnet-B CIDR"

PrivateSubnetACIDR:
 Type: String
 Default: "10.0.2.0/24"
 Description: "Private Subnet-A CIDR"

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				75
Змн.	Арк.	№ докум.	Підпис	Дата		

PrivateSubnetBCIDR:
Type: String
Default: "10.0.3.0/24"
Description: "Public Subnet-B CIDR"

DatabaseSubnetACIDR:
Type: String
Default: "192.0.0.0/24"
Description: "Database Subnet-A CIDR"

DatabaseSubnetBCIDR:
Type: String
Default: "192.0.1.0/24"
Description: "Database Subnet-B CIDR"

Resources:

VPC:
Type: AWS::EC2::VPC
Properties:
CidrBlock: !Ref VPCBlock
EnableDnsSupport: true
EnableDnsHostnames: true
Tags:
- Key: Name
Value: !Ref Environment

VPCCidrBlock2:
Type: "AWS::EC2::VPCCidrBlock"
DependsOn: VPC
DeletionPolicy: Delete
Properties:
CidrBlock: !Ref VPCBlock2
VpcId: !Ref VPC

GatewayInternet:
Type: "AWS::EC2::InternetGateway"
Properties:
Tags:
- Key: Name
Value: !Ref Environment

GatewayAttachmentInternet:
Type: "AWS::EC2::VPCGatewayAttachment"
Properties:
VpcId: !Ref VPC
InternetGatewayId: !Ref GatewayInternet

RouteTableForPublicSubnet:
Type: "AWS::EC2::RouteTable"
Properties:
VpcId: !Ref VPC
Tags:

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				76
Змн.	Арк.	№ докум.	Підпис	Дата		

- Key: Name
Value: !Join [" ", [!Ref Environment, "PublicRouteTable"]]

RoutesForPublicRouteTable:
Type: "AWS::EC2::Route"
DependsOn: GatewayAttachmentInternet
Properties:
DestinationCidrBlock: 0.0.0.0/0
GatewayId: !Ref GatewayInternet
RouteTableId: !Ref RouteTableForPublicSubnet

RouteTableForPrivateSubnetA:
Type: "AWS::EC2::RouteTable"
Properties:
VpcId: !Ref VPC
Tags:
- Key: Name
Value: !Join [" ", [!Ref Environment, "PrivateRouteTableA"]]

RouteTableForPrivateSubnetB:
Type: "AWS::EC2::RouteTable"
Properties:
VpcId: !Ref VPC
Tags:
- Key: Name
Value: !Join [" ", [!Ref Environment, "PrivateRouteTableB"]]

RoutesForPrivateRouteTableA:
Type: AWS::EC2::Route
DependsOn: NATGatewayA
Properties:
DestinationCidrBlock: 0.0.0.0/0
RouteTableId: !Ref RouteTableForPrivateSubnetA
NatGatewayId: !Ref NATGatewayA

RoutesForPrivateRouteTableB:
Type: AWS::EC2::Route
DependsOn: NATGatewayB
Properties:
DestinationCidrBlock: 0.0.0.0/0
RouteTableId: !Ref RouteTableForPrivateSubnetB
NatGatewayId: !Ref NATGatewayB

RouteAssociationPublicA:
Type: "AWS::EC2::SubnetRouteTableAssociation"
Properties:
RouteTableId: !Ref RouteTableForPublicSubnet
SubnetId: !Ref PublicSubnetA

RouteAssociationPublicB:
Type: "AWS::EC2::SubnetRouteTableAssociation"
Properties:

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		77

RouteTableId: !Ref RouteTableForPublicSubnet
SubnetId: !Ref PublicSubnetB

RouteAssociationPrivateA:
Type: "AWS::EC2::SubnetRouteTableAssociation"
Properties:
RouteTableId: !Ref RouteTableForPrivateSubnetA
SubnetId: !Ref PrivateSubnetA

RouteAssociationPrivateB:
Type: "AWS::EC2::SubnetRouteTableAssociation"
Properties:
RouteTableId: !Ref RouteTableForPrivateSubnetB
SubnetId: !Ref PrivateSubnetB

RouteTableForDatabase:
Type: "AWS::EC2::RouteTable"
Properties:
VpcId: !Ref VPC
Tags:
- Key: Name
Value: !Join [" ", [!Ref Environment, "DatabaseRouteTable"]]

RouteAssociationDatabaseA:
Type: "AWS::EC2::SubnetRouteTableAssociation"
Properties:
RouteTableId: !Ref RouteTableForDatabase
SubnetId: !Ref DatabaseSubnetA

RouteAssociationDatabaseB:
Type: "AWS::EC2::SubnetRouteTableAssociation"
Properties:
RouteTableId: !Ref RouteTableForDatabase
SubnetId: !Ref DatabaseSubnetB

EIP1:
Type: "AWS::EC2::EIP"
DependsOn: GatewayAttachmentInternet
Properties:
Domain: !Ref VPC
Tags:
- Key: Name
Value: EIP-NATGateway-A

EIP2:
Type: "AWS::EC2::EIP"
DependsOn: GatewayAttachmentInternet
Properties:
Domain: !Ref VPC
Tags:
- Key: Name
Value: EIP-NATGateway-B

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				78
Змн.	Арк.	№ докум.	Підпис	Дата		

NATGatewayA:

DependsOn: GatewayAttachmentInternet

Type: AWS::EC2::NatGateway

Properties:

SubnetId: !Ref PublicSubnetA

AllocationId: !GetAtt EIP1.AllocationId

Tags:

- Key: Name

Value: NATGateway-A

NATGatewayB:

DependsOn: GatewayAttachmentInternet

Type: AWS::EC2::NatGateway

Properties:

SubnetId: !Ref PublicSubnetB

AllocationId: !GetAtt EIP2.AllocationId

Tags:

- Key: Name

Value: NATGateway-B

PublicSubnetA:

Type: "AWS::EC2::Subnet"

Properties:

VpcId: !Ref VPC

AvailabilityZone: !Select [0, "Fn::GetAZs": { Ref: "AWS::Region" }]

CidrBlock: !Ref "PublicSubnetACIDR"

MapPublicIpOnLaunch: true

Tags:

- Key: Name

Value: !Join [""], [!Ref Environment, "-Public-A"]]

- Key: kubernetes.io/role/elb

Value: 1

PublicSubnetB:

Type: "AWS::EC2::Subnet"

Properties:

VpcId: !Ref VPC

AvailabilityZone: !Select [1, "Fn::GetAZs": { Ref: "AWS::Region" }]

CidrBlock: !Ref "PublicSubnetBCIDR"

MapPublicIpOnLaunch: true

Tags:

- Key: Name

Value: !Join [""], [!Ref Environment, "-Public-B"]]

- Key: kubernetes.io/role/elb

Value: 1

PrivateSubnetA:

Type: "AWS::EC2::Subnet"

Properties:

VpcId: !Ref VPC

AvailabilityZone: !Select [0, "Fn::GetAZs": { Ref: "AWS::Region" }]

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		79

CidrBlock: !Ref "PrivateSubnetACIDR"
MapPublicIpOnLaunch: false
Tags:
- Key: Name
Value: !Join [""], [!Ref Environment, "-Private-A"]]
- Key: kubernetes.io/role/internal-elb
Value: 1

PrivateSubnetB:
Type: "AWS::EC2::Subnet"
Properties:
VpcId: !Ref VPC
AvailabilityZone: !Select [1, "Fn::GetAZs": { Ref: "AWS::Region" }]
CidrBlock: !Ref "PrivateSubnetBCIDR"
MapPublicIpOnLaunch: false
Tags:
- Key: Name
Value: !Join [""], [!Ref Environment, "-Private-B"]]
- Key: kubernetes.io/role/internal-elb
Value: 1

DatabaseSubnetA:
Type: "AWS::EC2::Subnet"
DependsOn: VPCCidrBlock2
Properties:
VpcId: !Ref VPC
AvailabilityZone: !Select [0, "Fn::GetAZs": { Ref: "AWS::Region" }]
CidrBlock: !Ref "DatabaseSubnetACIDR"
MapPublicIpOnLaunch: false
Tags:
- Key: Name
Value: !Join [""], [!Ref Environment, "-Database-A"]]

DatabaseSubnetB:
Type: "AWS::EC2::Subnet"
DependsOn: VPCCidrBlock2
Properties:
VpcId: !Ref VPC
AvailabilityZone: !Select [1, "Fn::GetAZs": { Ref: "AWS::Region" }]
CidrBlock: !Ref "DatabaseSubnetBCIDR"
MapPublicIpOnLaunch: false
Tags:
- Key: Name
Value: !Join [""], [!Ref Environment, "-Database-B"]]

Outputs:
VPC:
Description: ID for the VPC
Value: !Ref VPC
Export:
Name: !Join ["-"], [!Ref "Environment", "VPC"]]

VPCBlock1:

Description: Primary CIDR block for the VPC

Value: !GetAtt VPC.CidrBlock

Export:

Name: !Join ["-", [!Ref "Environment", "CIDR1"]]

VPCBlock2:

Description: Secondary CIDR block for the VPC

Value: !Ref VPCBlock2

Export:

Name: !Join ["-", [!Ref "Environment", "CIDR2"]]

PublicA:

Description: ID for Public Subnet A

Value: !Ref PublicSubnetA

Export:

Name: !Join ["-", [!Ref "Environment", "PublicSubnetA"]]

PublicB:

Description: ID for Public Subnet B

Value: !Ref PublicSubnetB

Export:

Name: !Join ["-", [!Ref "Environment", "PublicSubnetB"]]

PrivateA:

Description: ID for Private Subnet A

Value: !Ref PrivateSubnetA

Export:

Name: !Join ["-", [!Ref "Environment", "PrivateSubnetA"]]

PrivateB:

Description: ID for Private Subnet B

Value: !Ref PrivateSubnetB

Export:

Name: !Join ["-", [!Ref "Environment", "PrivateSubnetB"]]

DatabaseA:

Description: ID for Database Subnet A

Value: !Ref DatabaseSubnetA

Export:

Name: !Join ["-", [!Ref "Environment", "DatabaseSubnetA"]]

DatabaseB:

Description: ID for Database Subnet B

Value: !Ref DatabaseSubnetB

Export:

Name: !Join ["-", [!Ref "Environment", "DatabaseSubnetB"]]

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		81

EKS config

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
```

```
metadata:
  name: foodua-cluster
  region: eu-north-1
```

```
vpc:
  id: "vpc-0bf62eb6a63bdc8b7"
subnets:
  public:
    eu-north-1a: { id: subnet-0e83143b8e372611b }
    eu-north-1b: { id: subnet-04a09b3da19a46e86 }
  private:
    eu-north-1a: { id: subnet-00122983533e9632e }
    eu-north-1b: { id: subnet-0ce3940912b1c1e26 }
```

```
nodeGroups:
- name: fe-group
  instanceType: t3.micro
  desiredCapacity: 3
  minSize: 1
  maxSize: 3
  privateNetworking: true

- name: be-group
  instanceType: t3.micro
  desiredCapacity: 3
  minSize: 1
  maxSize: 4
  privateNetworking: true
```

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		82

be-admin deployment.yaml

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: be-admin
  namespace: dev
  labels:
    app: be-admin
spec:
  selector:
    matchLabels:
      app: be-admin
  template:
    metadata:
      labels:
        app: be-admin
    spec:
      nodeSelector:
        alpha.eksctl.io/nodegroup-name: be-group
      containers:
        - name: be-admin
          image: 332149187237.dkr.ecr.eu-north-1.amazonaws.com/foodua/be-admin:dev
          imagePullPolicy: Always
          ports:
            - containerPort: 3000
          env:
            - name: DB_DIALECT
              valueFrom:
                configMapKeyRef:
                  name: be-admin-config
                  key: DB_DIALECT
            - name: DB_USERNAME
              valueFrom:
                secretKeyRef:
                  name: db-credentials
                  key: DB_USERNAME
            - name: DB_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: db-credentials
                  key: DB_PASSWORD
            - name: DB_DATABASE
              valueFrom:
                secretKeyRef:
                  name: db-credentials
                  key: DB_DATABASE
            - name: DB_HOST
              valueFrom:
                configMapKeyRef:
                  name: be-admin-config
                  key: DB_HOST

```

		Миценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				83
Змн.	Арк.	№ докум.	Підпис	Дата		

- name: DB_PORT
valueFrom:
configMapKeyRef:
 name: be-admin-config
 key: DB_PORT

		Міценчук М.М.			ККІКБ.КР.Б – 123 – 11 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		84

be-admin service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: be-admin
  namespace: dev
spec:
  selector:
    app: be-admin
  ports:
    - port: 80
      targetPort: 3000
  type: ClusterIP
```

Додаток H – be-admin hpa.yaml

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: be-admin-hpa
  namespace: dev
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: be-admin
  minReplicas: 1
  maxReplicas: 3
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 80
    - type: Resource
      resource:
        name: memory
        target:
          type: Utilization
          averageUtilization: 90
```