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 магістра

на тему: «Методи та інструменти безпечного розгортання та експлуатації мікросервісних Cloud-Native інформаційних систем у середовищі Kubernetes та Microsoft Azure»

Виконав: студент 2-го курсу групи КБм-24-1
спеціальності 125 «Кіберберзпека та захист інформації»
_____Микита ТКАЧУК
Керівник: Старший викладач кафедри КІ та КБ,
_____ Вадим МИКОЛАЙЧУК
Рецензент: Професор кафедри КІ та КБ, д.т.н., доцент
_____Володимир ВОРОТНІКОВ

Житомир – 2025

Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки
Спеціальність 125 «Кіберберзпека та захист інформації»
Освітня програма «Кіберберзпека»

ЗАТВЕРДЖУЮ
Завідувач
кафедри комп'ютерної
інженерії та кібербезпеки
Андрій ЄФІМЕНКО
6 жовтня 2025 р.

ЗАВДАННЯ

на випускну кваліфікаційну роботу магістра

Студент: Ткачук Микита Андрійович

Тема роботи: «**Методи та інструменти безпечного розгортання та експлуатації мікросервісних Cloud-Native інформаційних систем у середовищі Kubernetes та Microsoft Azure**»

затверджена Наказом університету від 6 жовтня 2025 р. № 566/с

Термін здачі студентом закінченої роботи 22 грудня 2025 р.

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

1. **Предмет дослідження** – є методи та технології розгортання мікросервісних Cloud-Native інформаційних систем у середовищі Kubernetes у хмарі Microsoft Azure з використанням кращих практик безпеки та дослідження.
2. **Об'єкт дослідження** – є процес побудови мікросервісної Cloud-Native інформаційних системи у середовищі Kubernetes у хмарі Microsoft Azure з використанням кращих практик безпеки.

Обладнання та програмне забезпечення:

1. Azure Kubernetes Service
2. Azure Container Registry

3. Microsoft Defender for Cloud
4. Azure DevOps
5. Docker v4.55.0
6. Kubernetes v1.35.0
7. Helm v4.0.0
8. .NET 8
9. ОС Windows 11

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

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Андрій ЄФІМЕНКО	6.10.25	6.10.25
2	Андрій ЄФІМЕНКО	6.10.25	6.10.25
3	Андрій ЄФІМЕНКО	6.10.25	6.10.25

Студент _____ Микита ТКАЧУК

Керівник _____ Вадим МИКОЛАЙЧУК

Календарний план

№ з/п	Назви етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Аналіз тематичної області, формулювання мети та завдань кваліфікаційної роботи	14.09.2025-20.09.2025	
2	Огляд сучасних підходів до побудови та захисту cloud-native мікросервісних систем, вибір технологічного стеку	21.09.2025-6.10.2025	
3	Проєктування архітектури системи «CoffeeShop» та інфраструктури в Microsoft Azure (AKS, мережа, сховище даних)	10.10.2025-20.10.2025	
4	Розроблення моделі безпеки, політик доступу та управління секретами (Entra ID, Key Vault, RBAC, Zero Trust)	21.10.2025-29.10.2025	
5	Створення мікросервісів «CoffeeShop» на ASP.NET Core та контейнеризація застосунку	30.10.2025-2.11.2025	
6	Проєктування та реалізація DevSecOps-процесів, CI/CD-конвеєрів у Azure DevOps, розробка Helm-чартів	3.11.2025-10.11.2025	
7	Налаштування моніторингу, логування та спостережуваності (Azure Monitor, Application Insights, Container Insights)	11.11.2025-15.11.2025	
8	Налаштування засобів виявлення загроз і підвищення безпекової позиції (Azure Policy, Microsoft Defender for Cloud/Containers/DevOps)	20.11.2025-27.11.2025	
9	Проведення експериментів, тестування працездатності та безпеки системи, аналіз результатів	28.11.2025-29.11.2025	
10	Оформлення пояснювальної записки, підготовка додатків та презентаційних матеріалів	30.11.2025-15.12.2025	

Студент _____ Микита ТКАЧУК

Керівник _____ Вадим МИКОЛАЙЧУК

АНОТАЦІЯ

Кваліфікаційна робота магістра присвячена розробленню еталонної архітектури захищеної мікросервісної cloud-native інформаційної системи класу інтернет-магазину «CoffeeShop» на базі платформи Microsoft Azure з використанням Kubernetes. Обсяг пояснювальної записки становить 90 сторінок, містить 42 рисунки і 4 таблиці. Список використаних джерел включає 47 найменувань і розміщений на 15 сторінках.

Метою роботи є обґрунтування та реалізація еталонної cloud-native архітектури мікросервісної системи з інтеграцією кращих практик безпеки для розгортання в середовищі Azure Kubernetes Service (AKS).

У роботі визначено основні завдання переходу від монолітних архітектур до мікросервісних cloud-native систем, проаналізовано технологічний стек Microsoft Azure, який буде застосуватись для вирішення даної задачі, розглянуто особливості побудови захищеної інфраструктури з мережевою сегментацією та моделлю Zero Trust, показано розробку DevSecOps-методології з уніфікованими Helm-чартами та CI/CD-конвеєрами в Azure DevOps, наведено результати впровадження систем моніторингу та виявлення загроз.

AKS, CLOUD-NATIVE, МІКРОСЕРВІСИ, KUBERNETES, DEVSECOPS, AZURE, ZERO TRUST, HELM, CI/CD

ANNOTATION

The master's qualification work is devoted to the development of a reference architecture for a secure microservices cloud-native information system of the online store class "CoffeeShop" based on the Microsoft Azure platform using Kubernetes. The volume of the explanatory note is 90 pages, contains 42 figures and 4 tables. The list of used sources includes 47 items and is placed on 15 pages.

The purpose of the work is to justify and implement a reference cloud-native architecture of a microservices system with the integration of best security practices for deployment in the Azure Kubernetes Service environment.

The work defines the main tasks of transitioning from monolithic architectures to microservices cloud-native systems, analyzes the Microsoft Azure technology stack considers the features of building secure infrastructure with network segmentation and Zero Trust model, demonstrates the development of DevSecOps methodology with unified Helm charts and CI/CD pipelines in Azure DevOps, presents the results of implementing monitoring systems and threat detection.

AKS, CLOUD-NATIVE, MICROSERVICES, KUBERNETES, DEVSECOPS, AZURE, ZERO TRUST, HELM, CI/CD

ЗМІСТ

ВСТУП 9

РОЗДІЛ 1. АНАЛІЗ ТЕХНОЛОГІЙ ТА ІНСТРУМЕНТАРІЮ ДЛЯ ПОБУДОВИ ЗАХИЩЕНИХ CLOUD-NATIVE СИСТЕМ 11

1.1 Еволюція архітектури вебдодатків від монолітної 11

1.2 Аналіз платформи оркестрації Azure Kubernetes Service: переваги керованого сервісу над іншими сервісами для роботи з контейнеризованими рішеннями 14

1.3 Роль пакетного менеджера Helm у стандартизації розгортання додатків 17

1.4 Огляд стеку безпеки Microsoft Azure: Zero Trust, Microsoft Defender for Cloud, Azure Policy. 20

1.5 Постановка завдання дослідження: вимоги до архітектури системи «CoffeeShop» 22

РОЗДІЛ 2. ПРОЄКТУВАННЯ ЗАХИЩЕНОЇ ІНФРАСТРУКТУРИ СИСТЕМИ В AZURE (INFRASTRUCTURE DESIGN) 26

2.1 Проєктування мережевої топології та захисту периметра 26

2.2 Організація безпечного зберігання даних та асинхронної комунікації 31

2.3 Управління ідентифікацією та секретами (Identity Management) 35

РОЗДІЛ 3. РЕАЛІЗАЦІЯ БЕЗПЕЧНОГО РОЗГОРТАННЯ З ВИКОРИСТАНЯНМ МЕТОДОЛОГІЇ DEVSECOPS НА ПЛАТФОРМІ AZURE 43

3.1 Організація репозиторію та управління версіями в Azure Repos 43

3.2 Розробка уніфікованих Helm-чартів для мікросервісів ASP.NET Core 8 48

3.3 Побудова конвеєра CI/CD в Azure DevOps Pipelines 53

РОЗДІЛ 4. РЕАЛІЗАЦІЯ КОМПЛЕКСНОЇ СИСТЕМИ МОНІТОРИНГУ ТА ВИЯВЛЕННЯ ЗАГРОЗ 59

4.1 Забезпечення відповідності політикам безпеки (Governance) 59

4.2 Моніторинг стану безпеки та виявлення вразливостей (Runtime Security) 62

4.3 Побудова системи моніторінгу (Observability) 66

ВИСНОВКИ 70

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 72

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AKS – Azure Kubernetes Service
CI/CD – Continuous Integration / Continuous Deployment
CNI – Container Network Interface
DevOps – Development and Operations
DevSecOps – Development, Security, Operations
IaC – Infrastructure as Code
RBAC – Role-Based Access Control
WAF – Web Application Firewall
ACR – Azure Container Registry
SQL – Structured Query Language
LTS – Long Term Support
SRE – Site Reliability Engineering
SAST – Static Application Security Testing
SOA – Service-Oriented Architecture
CNAPP – Cloud-Native Application Protection Platform
CSPM – Cloud Security Posture Management
CWPP – Cloud Workload Protection Platform
SIEM – Security Information and Event Management
TLS – Transport Layer Security
XSS – Cross-Site Scripting
OWASP – Open Worldwide Application Security Project
CRS – Core Rule Set
HTTP – HyperText Transfer Protocol
REST – Representational State Transfer
API – Application Programming Interface
TDE – Transparent Data Encryption
TTL – Time To Live
YAML – YAML Ain’t Markup Language
OIDC – OpenID Connect

ВСТУП

Актуальність теми. Сучасні інформаційні системи переходять від монолітної архітектури до cloud-native підходів, що базуються на контейнеризації, мікросервісах та оркестраторах на кшталт Kubernetes, оскільки це дозволяє підвищити масштабованість, відмовостійкість та швидкість оновлення програмного забезпечення. Водночас саме у таких розподілених середовищах суттєво зростає складність забезпечення безпеки: збільшується кількість точок доступу, з'являються ризики, пов'язані з неправильним керуванням секретами, вразливими контейнерними образами та помилками конфігурації хмарної інфраструктури.

Метою кваліфікаційної роботи є дослідження методів розгортання та захисту мікросервісних cloud-native інформаційних систем у середовищі Kubernetes у хмарі Microsoft Azure та розроблення еталонної архітектури на базі екосистеми Microsoft із урахуванням кращих практик безпеки.

Об'єктом дослідження є процес проєктування, розгортання та експлуатації мікросервісних cloud-native інформаційних систем у хмарному середовищі.

Предметом дослідження є методи, засоби та технології забезпечення безпечного розгортання мікросервісних вебзастосунків на платформі ASP.NET Core 8.0/.NET 8 (актуальний LTS-реліз на момент початку розроблення системи) у кластері Kubernetes в хмарі Microsoft Azure, включно з використанням сервісів екосистеми Microsoft для керування секретами, мережевою безпекою та моніторингом.

Визначена мета обумовлює наступні завдання:

- проаналізувати сучасні підходи до побудови та захисту мікросервісних cloud-native інформаційних систем у хмарному середовищі;
- дослідити можливості платформи Microsoft Azure та Kubernetes щодо забезпечення безпеки контейнеризованих застосунків (аутентифікація, авторизація, шифрування, керування секретами, мережеві політики тощо);

- спроєктувати еталонну мікросервісну архітектуру вебзастосунку на базі ASP.NET Core 8.0/.NET 8 із розгортанням у середовищі Azure Kubernetes Service та супровідною інфраструктурою;
- реалізувати прототип цієї архітектури на основі вдосконаленої бакалаврської роботи, інтегрувавши обрані механізми безпеки;
- провести експериментальне дослідження впливу впроваджених засобів безпеки на показники роботи системи (надійність, відмовостійкість, продуктивність) та сформулювати практичні рекомендації.

У процесі виконання кваліфікаційної роботи було розроблено проєкт мікросервісної cloud-native інформаційної системи у вигляді вебзастосунку на базі ASP.NET Core 8.0/.NET 8, розгорнутого у кластері Azure Kubernetes Service. Архітектура проєкту включає використання керованих сервісів Microsoft Azure (реєстр контейнерів, засоби зберігання секретів, моніторинг та журналювання), що дозволяє продемонструвати практичне застосування кращих практик безпеки для мікросервісних інформаційних систем.

РОЗДІЛ 1. АНАЛІЗ ТЕХНОЛОГІЙ ТА ІНСТРУМЕНТАРІЮ ДЛЯ ПОБУДОВИ ЗАХИЩЕНИХ CLOUD-NATIVE СИСТЕМ

1.1 Еволюція архітектури вебдодатків від монолітної

Перші вебдодатки будувалися переважно як монолітні системи, у яких уся бізнес-логіка, презентаційний рівень та доступ до даних реалізовувалися в межах єдиного застосунку, що розгортався на одному сервері або кластері з декількох однотипних вузлів [1]. Така архітектура спрощувала початкову розробку й розгортання, але зі зростанням функціональності призводила до ускладнення коду, збільшення часу випуску оновлень та проблем з масштабуванням окремих компонентів системи [2].

З появою хмарних обчислень та зростанням вимог до доступності й продуктивності почали широко застосовувати багатошарову (n-tier) архітектуру, де вебінтерфейс, прикладна логіка та база даних розділялися на окремі рівні, що могли розгортатися на різних серверах або сервісах [3]. Це дозволило гнучкіше масштабувати фронтенд і частково ізолювати бізнес-логіку, але застосунок залишався цілісним з погляду коду й циклу розробки: зміни в одній підсистемі часто вимагали перезбирання та перезапуску всього застосунку [4]. Ілюстрація, в якій представлено еволюцію архітектури вебдодатків від моноліту до мікросервісних cloud-native систем на рис. 1.1.

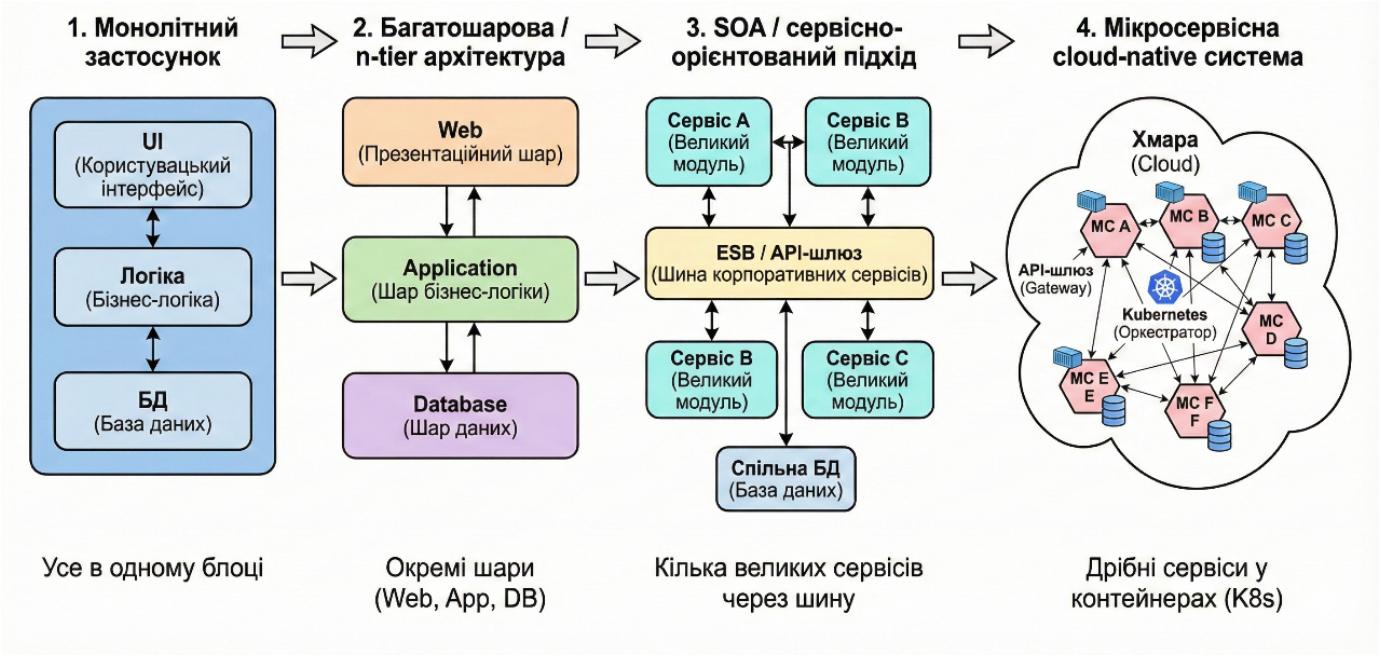


Рисунок 1.1 – Еволюція архітектури вебдодатків від моноліту до мікросервісних cloud-native систем

Подальший розвиток вебтехнологій, поява контейнеризації та оркестраторів, а також поширення DevOps-практик сприяли переходу до сервісно-орієнтованих підходів, а згодом — до мікросервісної архітектури, де функціональність системи розділяється на незалежні сервіси зі своїми інтерфейсами, циклами розробки й масштабування [5]. У поєднанні з хмарними платформами це стало передумовою формування концепції cloud-native систем, у яких застосунки спочатку проєктуються з урахуванням контейнеризації, автоматичного масштабування, відмовостійкості та спостережуваності, що кардинально змінює як підхід до розробки, так і до забезпечення безпеки [6].

У межах сервісно-орієнтованої архітектури (SOA) з’явилась ідея розбивати великі системи на окремі сервіси, які взаємодіють через стандартизовані протоколи, але на практиці такі системи часто залишалися важкими в розгортанні та еволюції через громіздкі інтеграційні шари й складне централізоване керування [7]. Ці обмеження стимулювали перехід до мікросервісної архітектури, що робить акцент на дрібні, автономні сервіси з чіткими API, власними циклами релізів та незалежним масштабуванням [8].

Важливим етапом еволюції стало впровадження контейнеризації, зокрема технологій на кшталт Docker, які дозволяють пакувати застосунок разом із його залежностями в ізольоване середовище, гарантуючи передбачувану поведінку під час розгортання в різних інфраструктурах [9]. Для керування великою кількістю контейнерів у виробничих середовищах почали застосовувати оркестратори, серед яких фактичним стандартом де-факто став Kubernetes, що забезпечує автоматичне масштабування, балансування навантаження, самоодужання сервісів та декларативне керування конфігурацією [10].

Концепція cloud-native поєднала мікросервіси, контейнеризацію, оркестрацію та практики DevOps/CI/CD, орієнтуючись на початкове проєктування систем з урахуванням можливостей хмарних платформ [11]. У такій парадигмі вебдодаток розглядається як набір незалежних сервісів, що розгортаються у хмарі, масштабується відповідно до навантаження, спостерігається за допомогою вбудованих засобів логування й моніторингу та інтегрується з керованими сервісами зберігання даних, черг повідомлень, балансувальниками навантаження тощо [12].

Разом з еволюцією архітектури змінювалися і підходи до безпеки: якщо в монолітних застосунках основний акцент робився на захисті периметра та бази даних, то в мікросервісних cloud-native системах критично важливими стають питання автентифікації й авторизації між сервісами, захисту мережевих комунікацій всередині кластера, безпечного керування секретами та оновлення вразливих контейнерних образів [13]. Таким чином, історичний перехід від монолітної до мікросервісної архітектури не лише розширив можливості масштабування й гнучкості вебдодатків, а й сформував нові виклики, що безпосередньо впливають на вибір технологій і інструментарію для побудови захищених cloud-native систем, які розглядаються у цій роботі [14].

1.2 Аналіз платформи оркестрації Azure Kubernetes Service: переваги керованого сервісу над іншими сервісами для роботи з контейнеризованими рішеннями

Azure Kubernetes Service (AKS) є керованим сервісом оркестрації контейнерів, у якому хмарна платформа Microsoft Azure бере на себе розгортання, оновлення та підтримку контрольної площини Kubernetes (API server, etcd, controller manager, scheduler), забезпечуючи її високу доступність і автоматичне застосування оновлень [15]. Користувач керує лише вузлами кластера (worker nodes) і робочими навантаженнями, що значно зменшує операційний тягар порівняно з розгортанням Kubernetes «з нуля» на віртуальних машинах або фізичних серверах [16].

У self-hosted сценаріях саме команда розробки або експлуатації відповідає за інсталяцію всіх компонентів Kubernetes, налаштування високої доступності контрольної площини, регулярні оновлення версій, резервне копіювання тощо, що вимагає глибокої експертизи й суттєвих витрат часу [17]. Будь-які помилки у конфігурації можуть призвести до простою кластера або зниження рівня безпеки, оскільки адміністратори мають самостійно інтегрувати засоби моніторингу, журналювання, аутентифікації та мережевої безпеки [18]. Ілюстрація, в якій представлено порівняння self-hosted Kubernetes з керованим сервісом Azure Kubr-netes Service архітектури представлено на рис. 1.2.

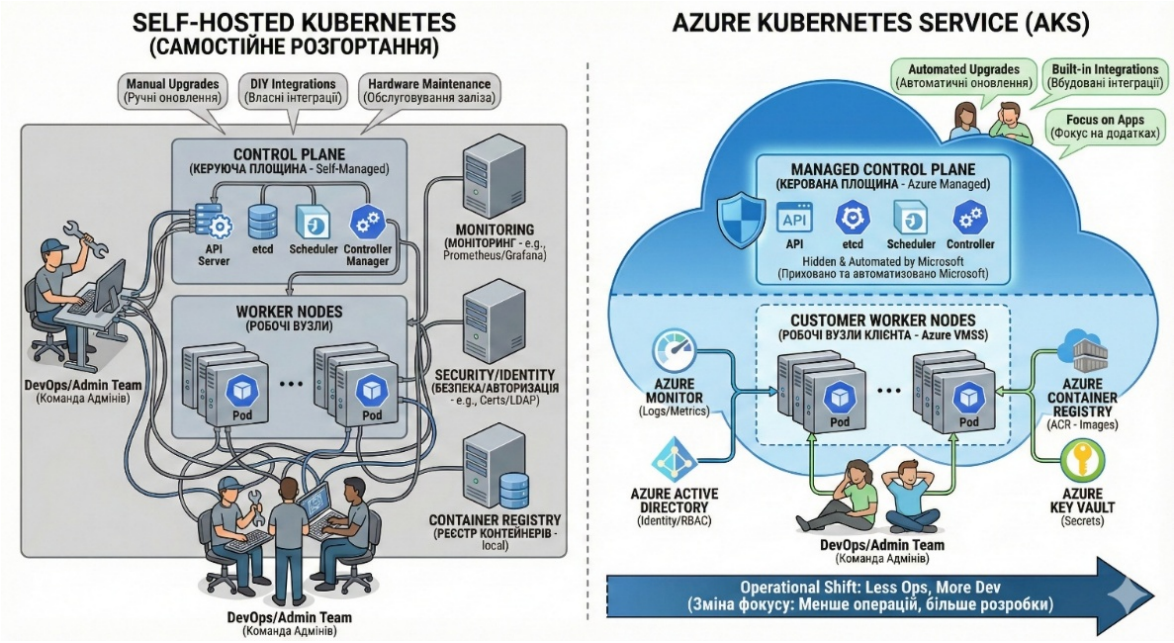


Рисунок 1.2 – Порівняння архітектури self-hosted кластера Kubernetes та керованого сервісу Azure Kubernetes Service

Окрім порівняння з self-hosted Kubernetes, важливо враховувати й інші варіанти розгортання контейнеризованих рішень в Azure, серед яких Azure App Service (Web App for Containers), Azure Container Apps та Azure Container Instances. Згідно з рекомендаціями Azure Architecture Center, AKS доцільно обирати тоді, коли потрібен повноцінний оркестратор із прямим доступом до API Kubernetes, складна мікросервісна архітектура, розширені мережеві політики та гнучке керування кластерами; для простіших вебзастосунків доцільніше використовувати Azure App Service, для подійно-орієнтованих та serverless-контейнерів — Azure Container Apps, а для короткочасних batch-навантежень без постійного кластера — Azure Container Instances.

Таким чином, AKS є виправданим вибором переважно для команд, які очікують значного масштабу навантажень, потребують тонкого контролю над конфігурацією кластера (мережеві плагіни, ingress-контролери, service mesh, політики безпеки) та планують стандартизувати розгортання на базі «чистого» Kubernetes. У випадках, коли архітектура системи є монолітною або наближеною до традиційних вебдодатків, а вимоги до кастомізації інфраструктури мінімальні, використання повноцінного кластера AKS може створювати зайвий операційний

оверхед порівняно з більш простими сервісами на кшталт Azure App Service чи Azure Container Apps.

Керований характер AKS надає низку переваг: сервіс автоматично здійснює health-моніторинг контрольної площини, підтримує кілька пулів вузлів, вбудовану підтримку горизонтального масштабування, а також інтеграцію з екосистемою Azure — Azure Container Registry, Azure Monitor, Log Analytics, Azure Active Directory та Azure Key Vault [19]. Це спрощує впровадження кращих практик безпеки, таких як централізоване керування ідентичностями через Entra ID, розмежування прав доступу за допомогою RBAC, застосування мережевих політик та безпечне зберігання секретів у спеціалізованих сервісах, без необхідності ручного налаштування кожного з цих компонентів у self-hosted кластері [20].

На відміну від self-hosted варіантів, де організація повністю відповідає за масштабування, оновлення та безпеку інфраструктури, AKS дозволяє зосередитися на розробці й експлуатації мікросервісних застосунків, використовуючи модель оплати лише за обчислювальні ресурси агентних вузлів, тоді як сама контрольна площина не тарифікується окремо [21]. Такий підхід робить AKS привабливим вибором для побудови захищених cloud-native систем на платформі Microsoft, особливо коли важливі швидкий старт, інтеграція з іншими сервісами Azure та зниження операційних ризиків, пов’язаних з помилками в конфігурації Kubernetes у self-hosted середовищі [22].

Таблиця 1.1

Порівняння AKS та self-hosted Kubernetes

Характеристика	AKS (керований сервіс)	Self-hosted Kubernetes
Розгортання control plane	Надано й обслуговується Azure	Повна відповідальність команди
Оновлення та патчі	Автоматизовані/керовані через AKS	Ручне планування та виконання
Інтеграція з Entra ID, ACR, Monitor, Key Vault	Вбудована підтримка, мінімальна конфігурація	Потрібна окрема інтеграція й налаштування
Витрати на експлуатацію	Зменшені операційні витрати	Вищі витрати на адміністрування
Фокус команди	На розробці та робочих навантаженнях	На підтримці інфраструктури кластера

1.3 Роль пакетного менеджера Helm у стандартизації розгортання додатків

Helm є пакетним менеджером для Kubernetes, який дозволяє описувати ресурси кластеру у вигляді повторно використованих шаблонів (charts), спрощуючи розгортання, оновлення та видалення складних застосунків [23]. Завдяки використанню декларативних маніфестів і параметризованих значень Helm забезпечує єдиний підхід до конфігурації середовищ (dev, test, production), що зменшує ризик помилок ручного налаштування та підвищує відтворюваність розгортань [24].

У контексті cloud-native мікросервісних систем Helm виступає ключовим інструментом стандартизації, оскільки дає змогу формалізувати структуру застосунку (Deployment, Service, Ingress, ConfigMap, Secret, Policy тощо) в межах одного chart’а або набору залежних chart’ів [25]. Це спрощує інтеграцію з конвеєрами CI/CD: збірка контейнерного образу, оновлення значень у values.yaml та автоматизований helm upgrade стають типовими кроками

доставки, що дозволяє підтримувати єдині практики розгортання для всієї організації [26].

Окрему роль у екосистемі Microsoft Azure відіграє сервіс Azure Container Registry, який використовується як приватне сховище не лише контейнерних образів, а й Helm-чартів, що публікуються як OCI-артефакти [27]. Розміщення chart'ів системи «CoffeeShop» в ACR дозволяє тримати в одному захищеному реєстрі як образи мікросервісів, так і пакети їх розгортання, застосовувати до них єдину політику контролю доступу на основі Azure RBAC та керованих ідентичностей, а також забезпечити аудит усіх операцій публікації та використання [28]. Фактично ACR виступає приватним аналогом публічних реєстрів на кшталт Docker Hub для образів та Artifact Hub для Helm-чартів, але з повним адміністративним контролем у межах підписки Azure [29]. Ілюстрація, в якій представлено схему процесу розгортання мікросервісного застосунку в Kubernetes за допомогою пакетного менеджера Helm представлено на рис. 1.3.

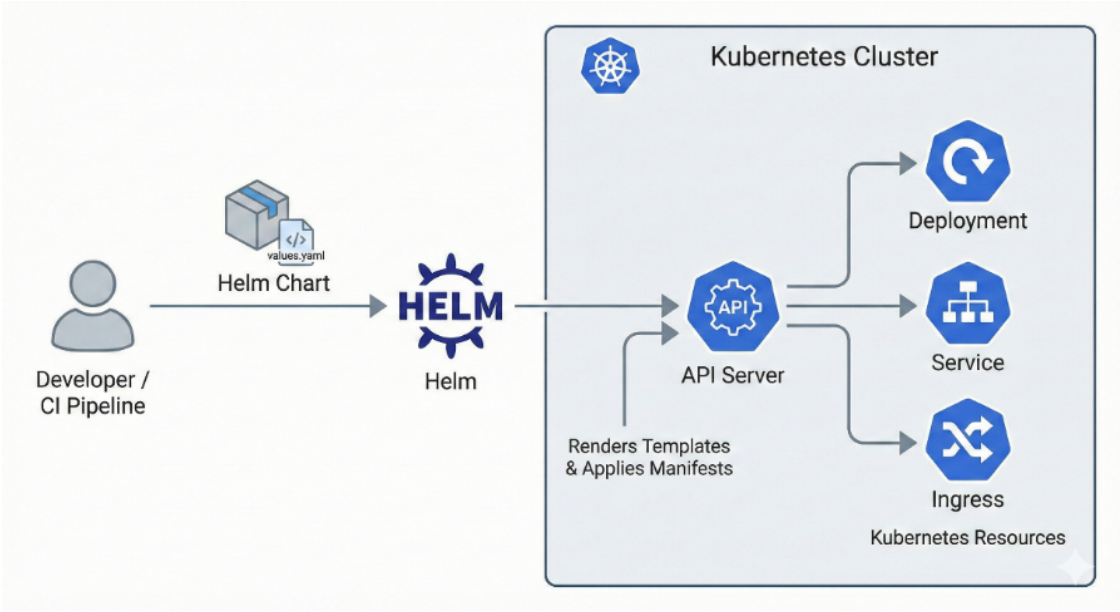


Рисунок 1.3 – Схема процесу розгортання мікросервісного застосунку в Kubernetes за допомогою пакетного менеджера Helm

З точки зору безпеки Helm полегшує впровадження кращих практик, оскільки у шаблонах можна одразу закласти обов'язкові параметри: обмеження ресурсів, securityContext, використання конфігурацій для секретів, налаштування

мережевих політик, а також інтеграцію з сервісами на кшталт Azure Key Vault через відповідні контролери [30]. Наявність централізованого репозиторію chart'ів (публічного чи приватного, зокрема на базі Azure Container Registry) дозволяє підтримувати «золоті» стандартизовані конфігурації сервісів, які проходять попередній аудит безпеки, що особливо важливо для корпоративних cloud-native платформ [31].

Крім базових операцій інсталяції та оновлення, Helm підтримує ведення історії релізів і можливість швидкого повернення до попередніх версій конфігурації за допомогою механізму rollback, що є важливим для мінімізації простоїв у виробничому середовищі [32]. Завдяки стандартній структурі chart'ів та підтримці шаблонів спільнота й вендори публікують перевірені пакунки для популярних компонентів (бази даних, брокери повідомлень, системи моніторингу), а в середовищі Azure ці пакунки можуть додатково кешуватися або зберігатися у власному реєстрі ACR, що суттєво прискорює побудову інфраструктури й знижує ризики помилок при ручному створенні маніфестів [33].

Таблиця 1.2

Переваги використання Helm для розгортання додатків у Kubernetes

Аспект	**Без Helm**	**З використанням Helm**
Опис ресурсів	Окремі YAML-маніфести	Структурований chart з шаблонами
Повторне використання	Обмежене, копіювання файлів	Параметризовані значення, спільні шаблони
Управління версіями	Ручне відстеження	Вбудована історія релізів і rollback
Узгодженість середовищ	Високий ризик відмінностей	Єдині chart'и для dev/test/prod
Інтеграція з CI/CD та ACR	Скрипти під кожен застосунок, окремі реєстри	Типові кроки helm install/upgrade із приватного реєстру ACR

1.4 Огляд стеку безпеки Microsoft Azure: Zero Trust, Microsoft Defender for Cloud, Azure Policy.

Zero Trust у Microsoft Azure.

Zero Trust у середовищі Microsoft Azure розглядається як стратегічний підхід до безпеки, що ґрунтується на принципах явної перевірки, найменших привілеїв і припущення компрометації, незалежно від розташування користувача чи ресурсу [34]. Це означає, що кожен запит до ресурсу в хмарі має бути автентифікований, авторизований і зафіксований з урахуванням контексту (ідентичність, стан пристрою, розташування, чутливість даних), а мережа за замовчуванням вважається ненадійною [35].
У контексті cloud-native мікросервісних систем Zero Trust реалізується через поєднання керування ідентичностями (Microsoft Entra ID), мікросегментації мережі (NSG, Azure Firewall, private endpoints) та безперервного моніторингу подій безпеки [36]. Такий підхід особливо важливий для Kubernetes-кластерів і сервісів, що обробляють чутливі дані, оскільки дозволяє обмежити бічний рух атак та зменшити наслідки потенційного компрометування окремого компонента системи [37].

Microsoft Defender for Cloud

Microsoft Defender for Cloud є хмарною платформою захисту застосунків (CNAPP), що поєднує можливості управління безпековою позицією (CSPM) та захисту робочих навантажень (CWPP) для ресурсів у Azure, мультихмарних середовищах і гібридній інфраструктурі [38]. Сервіс надає єдину панель для оцінки рівня безпеки (secure score), виявлення вразливостей, аналізу конфігурацій і генерації рекомендацій щодо підвищення захищеності ресурсів, включно з контейнерними кластерами AKS та пов’язаними службами зберігання даних [39]. Для мікросервісних cloud-native систем Defender for Cloud забезпечує моніторинг загроз, аналіз аномалій у поведінці сервісів і мережевого трафіку, а також інтеграцію з SIEM-рішеннями, такими як Microsoft Sentinel, для централізованого реагування на інциденти [40]. Використання цього сервісу дозволяє автоматизувати виявлення неправильно сконфігурованих ресурсів, небезпечних

відкритих портів, застарілих TLS-налаштувань і вразливих контейнерних образів, що є критичним для підтримки безпеки в динамічних середовищах Kubernetes [41].

Azure Policy як інструмент відповідності

Azure Policy є сервісом керування політиками, який дає змогу визначати, призначати та контролювати правила конфігурації ресурсів у хмарному середовищі з метою забезпечення відповідності корпоративним стандартам і вимогам безпеки [42]. Політики застосовуються на рівні груп керування, підписок, ресурсних груп або окремих ресурсів і можуть автоматично блокувати створення некоректно сконфігурованих ресурсів, виконувати їхнє виправлення або лише фіксувати порушення для подальшого аналізу [43].
У контексті захищених cloud-native архітектур Azure Policy дозволяє централізовано нав’язувати вимоги до розгортання кластерів AKS, мережевих ресурсів і сховищ даних, наприклад: обмежувати дозволені регіони, типи віртуальних машин, вимагати використання керованих ідентичностей, шифрування даних та ввімкнення журналювання [44]. Поєднання Azure Policy з принципами Zero Trust і можливостями Microsoft Defender for Cloud створює багаторівневий стек безпеки, який забезпечує як проактивне попередження помилкових конфігурацій, так і оперативне виявлення та реагування на загрози в межах хмарної мікросервісної системи [45]. Ілюстрація, в якій представлено узагальнену схему стеку безпеки Microsoft Azure для мікросервісної cloud-native системи, що поєднує принципи Zero Trust, Microsoft Defender for Cloud та Azure Policy на рис. 1.4.

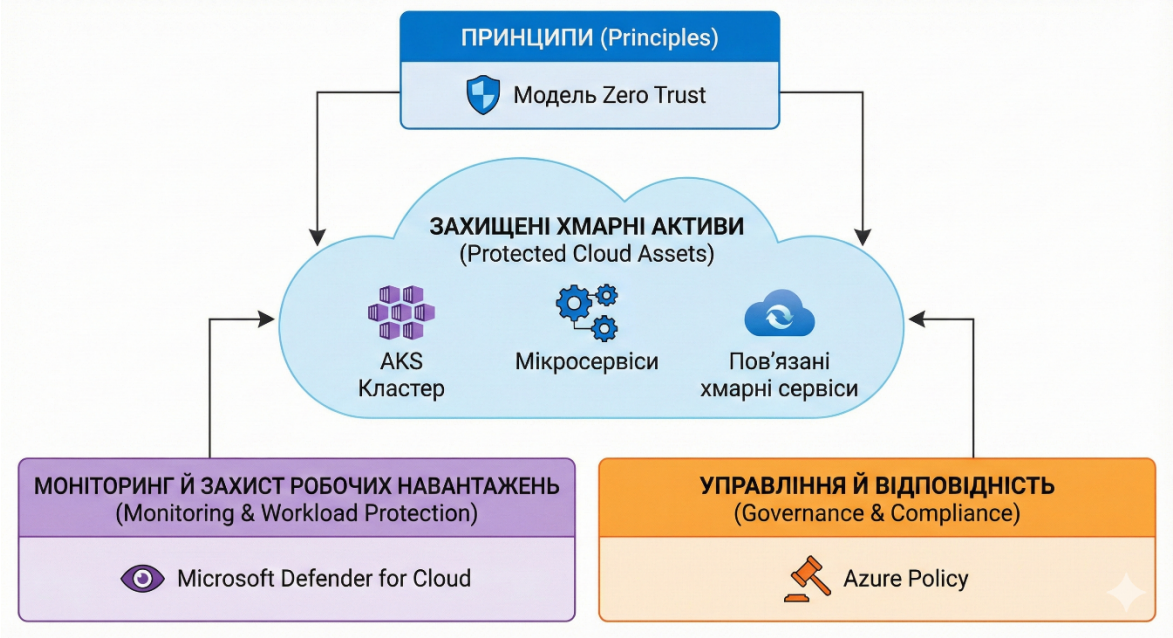


Рисунок 1.4 – Узагальнена схема стеку безпеки Microsoft Azure для мікросервісної cloud-native системи, що поєднує принципи Zero Trust, Microsoft Defender for Cloud та Azure Policy

1.5 Постановка завдання дослідження: вимоги до архітектури системи «CoffeeShop»

У межах даної роботи як об'єкт прикладного впровадження обрано інформаційну систему «CoffeeShop», призначену для реалізації інтернет-магазину з продажу різних видів кави, кавових напоїв та пов'язаних з кавовою тематикою товарів (аксесуарів, сувенірної продукції тощо) через вебінтерфейс з можливістю подальшого масштабування. Система розглядається як типове представлення класу вебзастосунків електронної комерції малого та середнього бізнесу, для яких особливо актуальними є вимоги до доступності, безпеки обробки персональних даних та гнучкості розвитку функціоналу.

Функціональні вимоги. Демонстраційна система «CoffeeShop» реалізує базовий сценарій інтернет-магазину (каталог товарів, кошик, оформлення замовлень, облікові записи користувачів, просте адміністрування), але основний фокус роботи зроблено на тому, як цей функціонал розгортається та захищається в хмарній інфраструктурі Microsoft Azure. Для реалізації прикладних сценаріїв використовуються керовані сервіси Azure: зберігання структурованих даних у

кількох базах Azure SQL Database, мультимедійних файлів у Azure Blob Storage, обмін подіями між мікросервісами через Azure Service Bus, публікація вебінтерфейсу та API через зв'язку Application Gateway for Containers + Web Application Firewall, а також інтеграція з Microsoft Entra ID для автентифікації користувачів і сервісів. Вибір саме цих служб обумовлений рекомендаціями Azure Architecture Center щодо підбору сховищ даних та сервісів обміну повідомленнями для хмарних застосунків. Для зберігання транзакційних даних інтернет-магазину (замовлення, облікові записи, програма лояльності) використовується Azure SQL Database як керований реляційний сервіс з високою доступністю та вбудованими механізмами резервного копіювання, тоді як Cosmos DB або Azure SQL Managed Instance доцільніші для сценаріїв глобального розподілу даних або міграції існуючих SQL-серверів. Мультимедійні файли товарів розміщуються в Azure Blob Storage, оскільки цей сервіс надає масштабоване й економічне об'єктне сховище, а self-hosted варіанти на кшталт MinIO чи Ceph не відповідають вимогам до повністю керованого cloud-native рішення. Для асинхронної взаємодії між мікросервісами обрано Azure Service Bus, який підтримує надійні черги, теми й підписки з гарантією доставки та розширеними можливостями маршрутизації; порівняно з Storage Queues, Event Hubs чи Event Grid це робить його більш придатним для критичних бізнес-процесів електронної комерції. Саме така конфігурація дозволяє в подальших розділах дослідити методи розгортання мікросервісної cloud-native архітектури в AKS із використанням Helm, Azure DevOps, Azure Policy, Microsoft Defender for Cloud та інших засобів безпеки екосистеми Azure.

Нефункціональні вимоги. З урахуванням переходу до cloud-native архітектури ключовими нефункціональними вимогами є: масштабованість (можливість горизонтального масштабування окремих мікросервісів у кластері AKS), відмовостійкість (стійкість до виходу з ладу окремих вузлів або сервісів), спостережуваність (централізований збір логів, метрик та трас), а також інформаційна безпека. Окремо формулюються вимоги до застосування принципів Zero Trust, використання керованих сервісів захисту (Microsoft Defender for Cloud)

та політик відповідності (Azure Policy) для контролю конфігурації ресурсів і мінімізації ризиків, пов’язаних з людським фактором.

Задача дослідження. Виходячи з сформульованих вимог, основною задачею дослідження є проєктування та експериментальна перевірка еталонної мікросервісної архітектури системи «CoffeeShop» на базі ASP.NET Core 8.0/.NET 8, розгорнутої в кластері Azure Kubernetes Service із застосуванням Helm, Zero Trust-підходу, Microsoft Defender for Cloud та Azure Policy. Вибір платформи .NET 8 зумовлений тим, що на момент початку розроблення прикладної частини вона була актуальним LTS-релізом, який забезпечує тривалий супровід, стабільність API та сумісність з необхідними бібліотеками, а також тим, що вихідні компоненти системи були реалізовані саме на цій версії, що робить міграцію на .NET 10 недоцільною в межах поставлених завдань. У межах дослідження планується оцінити вплив обраних рішень і налаштувань безпеки на показники продуктивності, надійності та зручності експлуатації системи, а також сформувати практичні рекомендації щодо побудови захищених cloud-native вебзастосунків класу «інтернет-магазин кав’ярні». Ілюстрація, в якій представлено високорівневу архітектуру системи «CoffeeShop» у середовищі Azure Kubernetes Service на рис. 1.5.

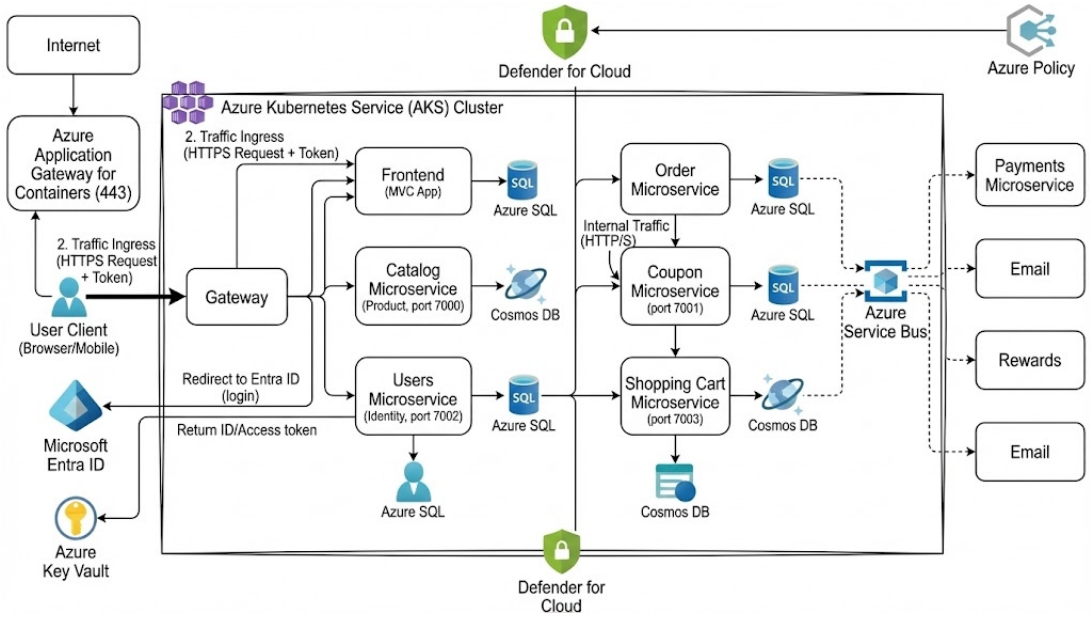


Рисунок 1.5 – Високорівнева архітектура системи «CoffeeShop» у середовищі Azure Kubernetes Service

Висновки до розділу 1

У першому розділі проаналізовано еволюцію архітектури вебдодатків від монолітної до мікросервісної cloud-native парадигми, що продемонструвало переваги контейнеризації й оркестрації для масштабованості, відмовостійкості та гнучких оновлень вебсистем класу інтернет-магазинів «CoffeeShop». Порівняння Azure Kubernetes Service із self-hosted кластерами обґрунтувало вибір AKS як рішення, що знижує операційний тягар, спрощує інтеграцію з екосистемою Azure та підтримує впровадження кращих практик безпеки.

Показано, що стандартизовані Helm-чарти уніфікують процес розгортання мікросервісів, підвищують відтворюваність конфігурацій і дозволяють закласти необхідні налаштування безпеки в шаблони інфраструктури. Огляд стеку безпеки Microsoft Azure засвідчив, що поєднання принципів Zero Trust, Microsoft Defender for Cloud та Azure Policy формує багаторівневу модель захисту для cloud-native систем і задає вимоги до архітектури системи «CoffeeShop» та подальших етапів її дослідження.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ЗАХИЩЕНОЇ ІНФРАСТРУКТУРИ СИСТЕМИ В AZURE (INFRASTRUCTURE DESIGN)

2.1 Проєктування мережевої топології та захисту периметра

Використання Azure Application Gateway for Containers та Web Application Firewall (WAF) для захисту периметра системи «CoffeeShop»

Для захисту зовнішнього периметра системи «CoffeeShop» доцільно використовувати Azure Application Gateway for Containers з увімкненим модулем Web Application Firewall (WAF), який виконує роль L7-балансувальника навантаження та центральної точки фільтрації HTTP(S)-трафіку. Application Gateway for Containers, інтегрований з Kubernetes через Gateway API, забезпечує термінацію TLS-з'єднань, маршрутизацію запитів до відповідних бекенд-служб кластера AKS та допоміжних сервісів, підтримку cookie-based session affinity й контент-орієнтоване маршрутизування, що дозволяє гнучко розподіляти навантаження між мікросервісами вебзастосунку. У контексті системи «CoffeeShop» Azure Application Gateway for Containers виконує роль зовнішнього ingress-контролера для кластера AKS, термінуючи TLS-з'єднання, розподіляючи трафік між frontend-застосунком та API-шлюзом і одночасно застосовуючи WAF-політики до всіх вхідних HTTP(S)-запитів. У типових сценаріях публікації система «CoffeeShop» використовує окремі HTTP(S)-маршрути для вебінтерфейсу та REST-API, що описуються через Gateway API й обробляються Application Gateway for Containers, який далі спрямовує запити до відповідних сервісів у кластері AKS. Це дозволяє застосовувати різні WAF-політики та обмеження доступу для публічних сторінок, адміністративних інтерфейсів і сервісних API, зменшуючи площу атаки вебрівня.

Модуль Web Application Firewall на базі Azure Application Gateway for Containers реалізує захист від типових вебатак, зокрема SQL-ін'єкцій, міжсайтового скриптингу (XSS) та інших загроз, описаних в OWASP Top 10, за рахунок застосування оновлюваних правил Core Rule Set (CRS). Централізовані WAF-політики, що прив'язуються до HTTP-маршрутів і хостів, а також можливість

створення кастомних правил дозволяють адаптувати рівень фільтрації до специфіки REST-API та фронтенд-запитів системи «CoffeeShop», тоді як інтеграція з Azure Monitor і Microsoft Defender for Cloud забезпечує збір журналів атак і побудову аналітики інцидентів безпеки. Окрім класичних сигнатур OWASP Top 10, модуль Web Application Firewall підтримує можливості bot management, що дозволяють відрізняти легітимний трафік від автоматизованих ботів, блокувати відомі шкідливі user-agent'и та застосовувати додаткові обмеження до підозрілої активності. Це знижує ризик автоматизованих атак на публічні інтерфейси системи, такі як brute-force, сканування вразливостей чи агресивний web-scraping. Ілюстрація, в якій представлено використання Azure Application Gateway for Containers для публікації сервісів кластера AKS на рис. 2.1.

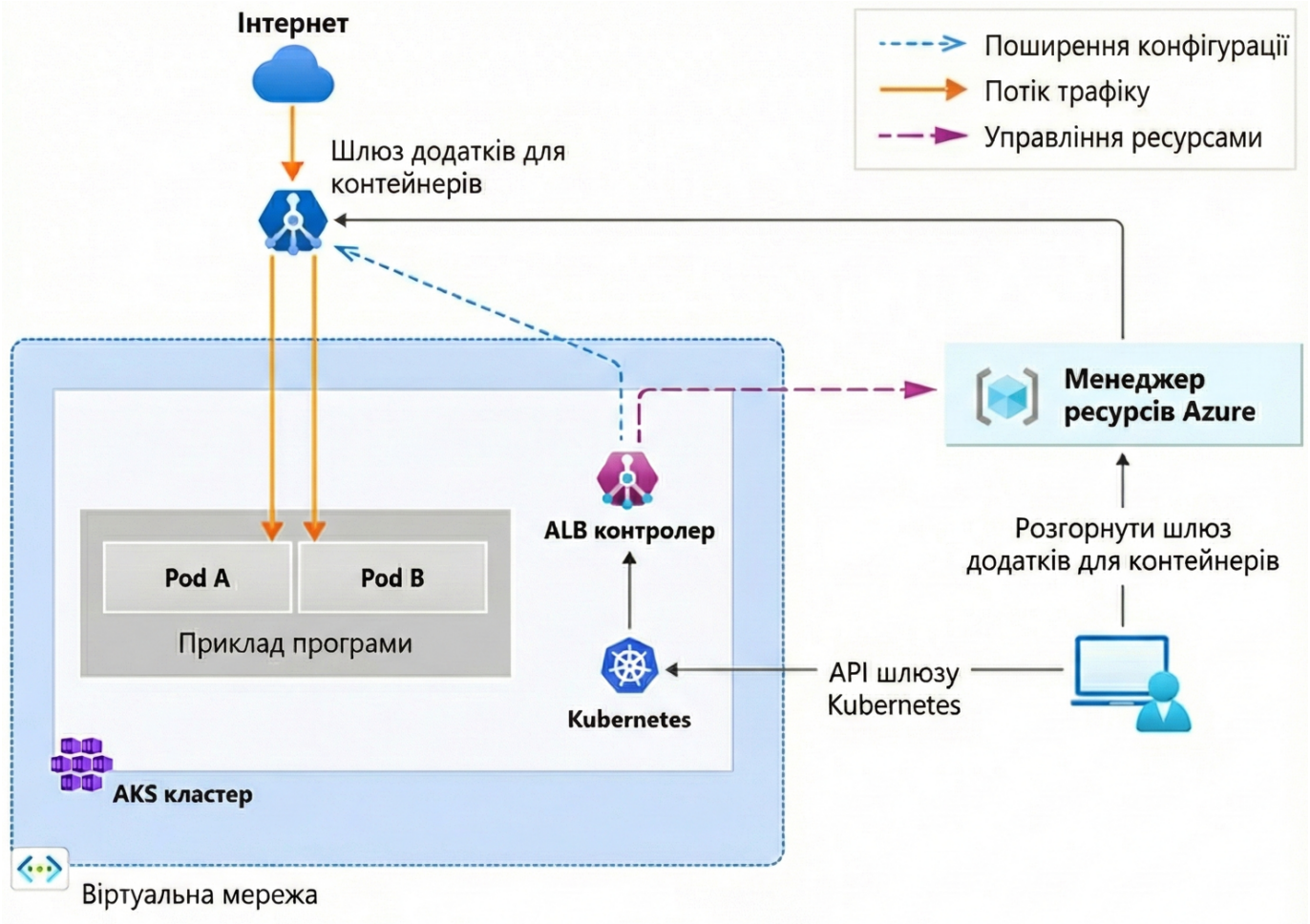


Рисунок 2.1 – Використання Azure Application Gateway for Containers для публікації сервісів кластера AKS

Доповнюючи захист периметра, важливо врахувати сценарії публікації як класичного вебінтерфейсу, так і API-шлюзу для мобільних або зовнішніх інтеграцій, оскільки саме ці компоненти стають основною ціллю атак на рівні додатків [46]. Використання окремих HTTP-маршрутів і хостів у конфігурації Azure Application Gateway for Containers дозволяє розділити потоки трафіку між публічним вебсайтом «CoffeeShop» та службовими API, застосовуючи до них різні політики WAF і різні рівні обмежень доступу [47]. Це дає змогу, наприклад, посилити перевірку запитів до адміністративних інтерфейсів або API-методів, що оперують платіжними й персональними даними користувачів [48].

Окрему увагу слід приділити налаштуванню TLS-термінації та шифруванню трафіку «від клієнта до бекенда», оскільки слабкі криптографічні налаштування або використання самопідписаних сертифікатів можуть нівелювати інші заходи захисту [49]. Azure Application Gateway for Containers підтримує використання сертифікатів, виданих довіреними центрами сертифікації, а також інтеграцію з Azure Key Vault для безпечного зберігання й автоматичного оновлення TLS-сертифікатів, що дає змогу мінімізувати ризики, пов’язані з їхнім ручним адмініструванням [50]. Для внутрішньої взаємодії між шлюзом додатків для контейнерів та сервісами в кластері AKS може застосовуватися наскрізне TLS-шифрування з окремими сертифікатами для бекенд-служб, що відповідає вимогам Zero Trust-підходу, згідно з яким навіть внутрішній трафік розглядається як потенційно ненадійний [51].

Для підвищення надійності периметра доцільно розглянути сценарій розгортання Application Gateway for Containers у кількох зонах доступності з увімкненим автоматичним масштабуванням залежно від навантаження [52]. Така конфігурація дозволяє зменшити ризик простою фронтового рівня системи «CoffeeShop» у разі відмови окремої зони або пікових навантажень, пов’язаних, наприклад, з маркетинговими кампаніями чи сезонними акціями. Централізований збір журналів доступу та журналів WAF у сховище журналів, інтегроване з системою моніторингу та аналітики Azure Monitor і Microsoft Defender for Cloud,

створює основу для подальшого коригування правил фільтрації та вдосконалення моделі загроз для вебрівня застосунку [53].

Реалізація мережевої ізоляції за допомогою Azure CNI та Network Policies (Cilium)

Для внутрішнього захисту кластера AKS у системі «CoffeeShop» доцільно використовувати мережевий плагін Azure CNI, який надає кожному поду власну IP-адресу з адресного простору віртуальної мережі Azure VNet, роблячи поди [54] «повноцінними» учасниками корпоративної мережі. Це спрощує інтеграцію з іншими хмарними й гібридними ресурсами, оскільки до подів можна застосовувати ті самі мережеві засоби захисту (Network Security Groups, Azure Firewall, журнали трафіку), що й до звичайних віртуальних машин, що важливо для побудови zero-trust-архітектури. У поєднанні з механізмом мережевих політик на базі Cilium (Azure CNI powered by Cilium), який використовує eBPF-програми в ядрі Linux, забезпечується ефективне застосування політик L3/L4 та, за потреби, політик рівня L7 з фільтрацією HTTP/gRPC-запитів, FQDN-фільтрацією й розширеною спостережуваністю трафіку в кластері. Такий підхід дозволяє реалізувати мікросегментацію трафіку між подами та сервісами, зменшити накладні витрати порівняно з класичними iptables-базованими рішеннями та масштабувати кластер до більшої кількості вузлів і сервісів без втрати продуктивності. Виділення окремих namespace для різних доменів функціональності (наприклад, frontend, orders, catalog, identity, admin) формує логічні зони всередині кластера, для яких політики Cilium описуються через мітки pod-ів і сервісів, а не конкретні IP-адреси. Це дозволяє гнучко обмежити, які сервіси можуть ініціювати з’єднання один з одним і до яких портів або URL-шляхів, тим самим зменшуючи площу атаки та спрощуючи аудит політик безпеки.

Практична реалізація мережевої ізоляції ґрунтується на принципі «deny by default», коли за замовчуванням внутрішній трафік між namespace та подами блокується, а дозволи надаються лише для необхідних комунікацій згідно з моделлю взаємодії мікросервісів «CoffeeShop». Для цього створюються глобальні

політики, що забороняють будь-які вхідні з'єднання до сервісів, крім тих, що явно дозволені, а також спеціалізовані правила, які, наприклад, надають доступ фронтенд-подам до API-шлюзу, сервісу замовлень і сервісу каталогу, але блокують прямий доступ до служб управління користувачами чи адміністративних інтерфейсів.

Cilium як реалізація мережевих політик рівня L3–L7 дозволяє не лише обмежувати трафік за IP-адресами й портами, а й будувати правила з урахуванням протоколів верхнього рівня, HTTP-шляхів та ідентичностей служб, що підвищує точність контролю доступу всередині кластера. Такий підхід зменшує можливості для бічного руху зловмисника у випадку компрометації окремого пода, оскільки навіть за наявності виконуваного коду в скомпрометованому контейнері мережеві обмеження не дозволять йому безпосередньо звернутися до критичних сервісів або баз даних. Ілюстрація, в якій представлено внутрішню мережну сегментацію кластера Azure Kubernetes Service для системи «CoffeeShop» із використанням Azure CNI та мережевих політик Cilium на рис. 2.2.

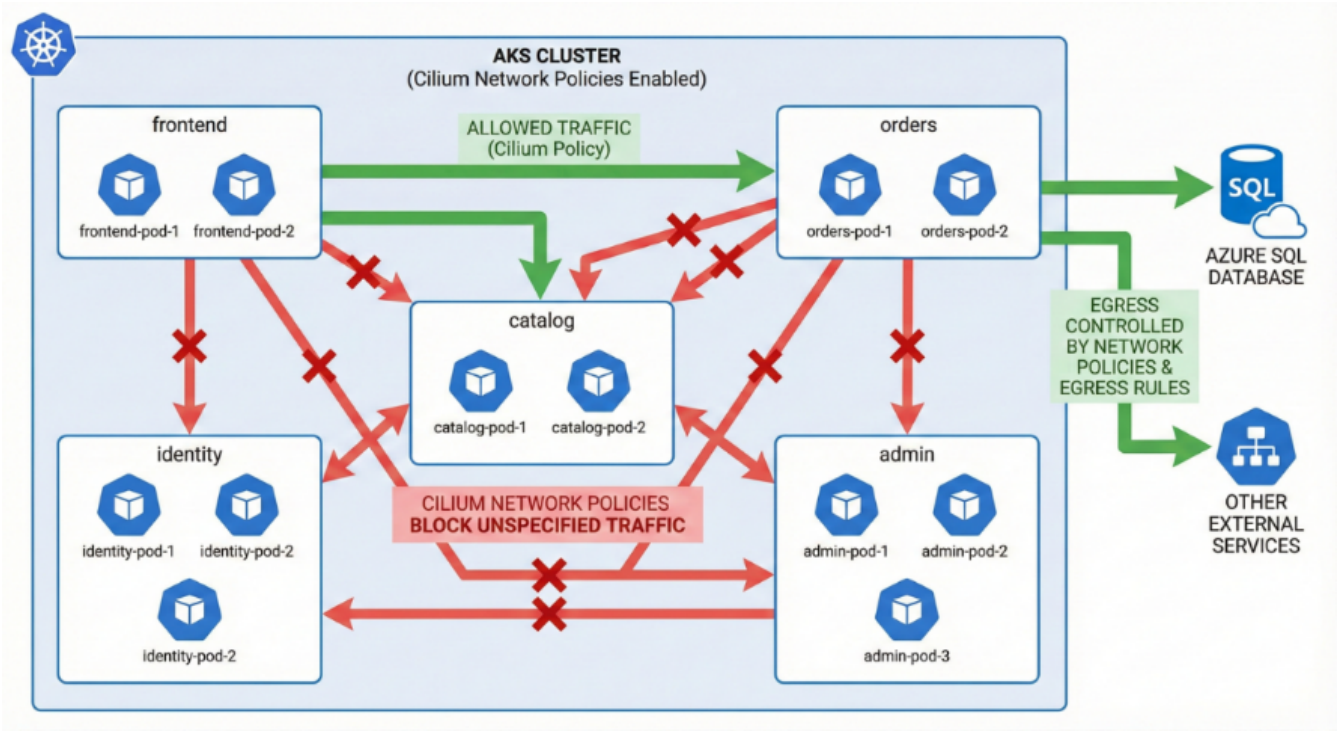


Рисунок 2.2 – Внутрішня мережна сегментація кластера Azure Kubernetes Service для системи «CoffeeShop» із використанням Azure CNI та мережевих політик Cilium

2.2 Організація безпечного зберігання даних та асинхронної комунікації Використання Azure SQL Database та Azure Blob Storage.

Основні структуровані дані системи «CoffeeShop» зберігаються у декількох логічно розділених базах даних, що відповідають доменним областям застосунку: автентифікація й керування користувачами (CoffeeShop_Auth), каталог товарів (CoffeeShop_Product), замовлення (CoffeeShop_Order), кошик (CoffeeShop_ShoppingCart), купони та знижки (CoffeeShop_Coupon), система лояльності (CoffeeShop_Reward) та сервіс розсилок (CoffeeShop_Email). Для цих даних доцільно використовувати керовану реляційну платформу Azure SQL Database (PaaS), яка забезпечує вбудовані механізми високої доступності, автоматичного резервного копіювання, шифрування даних та інтеграції із засобами безпеки Azure. Ілюстрація, в якій представлено перелік баз даних мікросервісів системи «CoffeeShop» на сервері Azure SQL на рис. 2.3.

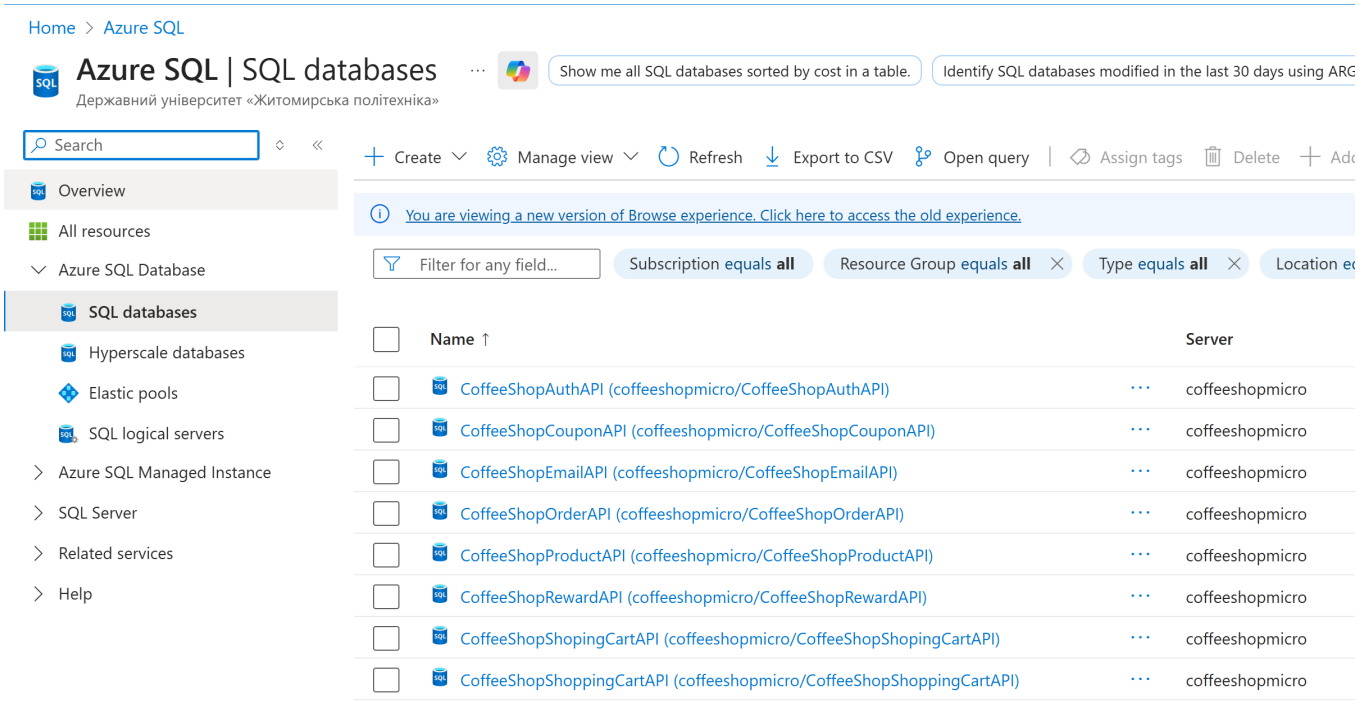


Рисунок 2.3 – Перелік баз даних мікросервісів системи «CoffeeShop» на сервері Azure SQL

Azure SQL Database підтримує Transparent Data Encryption (TDE), шифрування даних у русі за допомогою TLS, а також вбудовані засоби виявлення загроз і моніторингу аномальної активності запитів, що особливо важливо для

таблиць, які містять персональні дані користувачів, історію замовлень і інформацію про програму лояльності. Додаткове використання рольової моделі доступу й принципу найменших привілеїв для облікових записів застосунків дозволяє обмежити доступ окремих мікросервісів лише тими операціями, які необхідні їхній бізнес-логіці (наприклад, сервіс розсилок має лише операції читання адрес із CoffeeShop_Email, а не повний доступ до замовлень чи системи купонів). Ілюстрація, в якій представлено логічна схема зберігання даних системи «CoffeeShop» на базі Azure SQL Database та Azure Blob Storage на рис. 2.4.

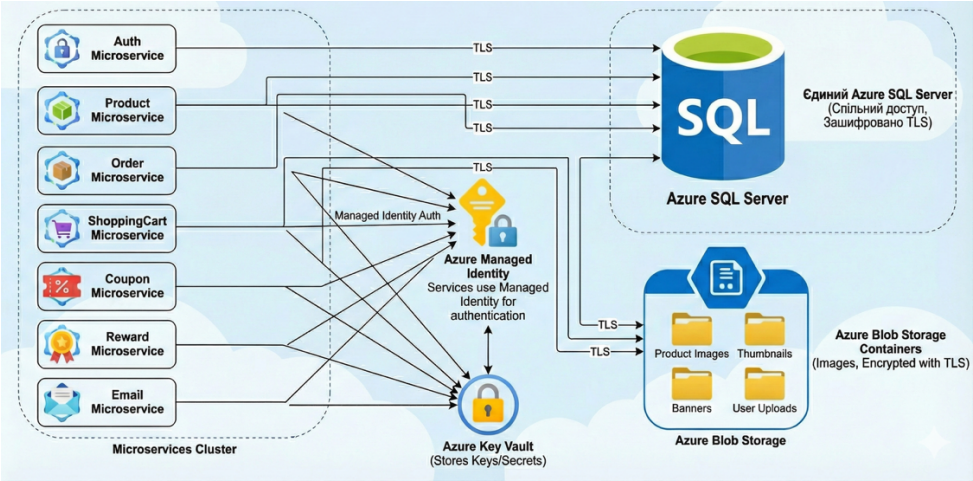


Рисунок 2.4 – Логічна схема зберігання даних системи «CoffeeShop» на базі Azure SQL Database та Azure Blob Storage

Неструктуровані та великі об’єкти, зокрема зображення кави, напоїв, аксесуарів та іншої пов’язаної з брендом продукції, доцільно зберігати в Azure Blob Storage, що оптимізований для масштабованого й дешевшого зберігання файлів. Доступ до контейнерів Blob Storage має здійснюватися через керовані ідентичності та тимчасові токени (SAS-посилання) замість статичних ключів доступу, що знижує ризик компрометації секретів та відповідає вимогам Zero Trust-підходу. Неструктуровані та великі об’єкти, зокрема зображення кави, напоїв, аксесуарів та іншої пов’язаної з брендом продукції, доцільно зберігати в Azure Blob Storage, що оптимізований для масштабованого й дешевшого зберігання файлів. Доступ до контейнерів Blob Storage має здійснюватися через керовані ідентичності та тимчасові токени (SAS-посилання) замість статичних ключів доступу, коли бекенд-сервіси отримують права читання/запису завдяки Azure AD-аутентифікації,

а для клієнтських застосунків генеруються обмежені за часом і дозволами SAS-токени. Такий підхід усуває потребу розповсюджувати довгоживучі ключі облікового запису сховища, знижує ризик компрометації секретів і відповідає вимогам Zero Trust-підходу, за яким доступ надається за принципом найменших привілеїв і лише на необхідний час.

Захист черг повідомлень Azure Service Bus

Для реалізації асинхронних процесів у системі «CoffeeShop» (обробка замовлень, нарахування бонусів, надсилання email-повідомлень тощо) використовується Azure Service Bus, який надає черги та топіки з гарантованою доставкою повідомлень і підтримкою шаблону «publish/subscribe». Такий підхід дозволяє розв’язати вебінтерфейс і бекенд-процеси в часі: фронтенд або API-шлюз швидко реєструє замовлення і публікує подію в Service Bus, а спеціалізовані робітники (order-processor, email-sender, reward-processor) обробляють її у фоновому режимі, що підвищує відмовостійкість і чутливість системи до пікових навантажень.

З точки зору безпеки, доступ до просторів імен Service Bus має здійснюватися переважно через керовані ідентичності (Managed Identity / Workload Identity) з використанням ролей на кшталт Azure Service Bus Data Sender та Azure Service Bus Data Receiver, замість передачі статичних connection string із загальними ключами. Це дозволяє застосувати принцип найменших привілеїв: мікросервіс, який лише надсилає події про створення замовлень, отримує права на відправлення в конкретну чергу або топік, але не може читати або видаляти повідомлення; натомість worker-служба обробки замовлень має права лише на читання й підтвердження обробки відповідних повідомлень. Усі з’єднання з Service Bus повинні встановлюватися по TLS, а журнали доступу мають передаватися в централізовану систему моніторингу для виявлення підозрілої активності.

Для підвищення стійкості до збоїв та атак на цілісність даних у чергах доцільно використовувати механізми dead-letter черг, повторних спроб (retry policies) та часових обмежень життя повідомлень (TTL), що запобігає

нескінченному «застряганню» некоректних повідомлень у основних чергах. Це особливо важливо для критичних бізнес-подій, пов’язаних із платіжними транзакціями, нарахуванням бонусів і застосуванням купонів, де повторна або помилкова обробка може призвести до фінансових втрат чи порушення цілісності обліку. Ілюстрація, в якій представлено організація асинхронної обробки подій у системі «CoffeeShop» за допомогою Azure Service Bus queues/topics та робітників-споживачів на рис. 2.5.

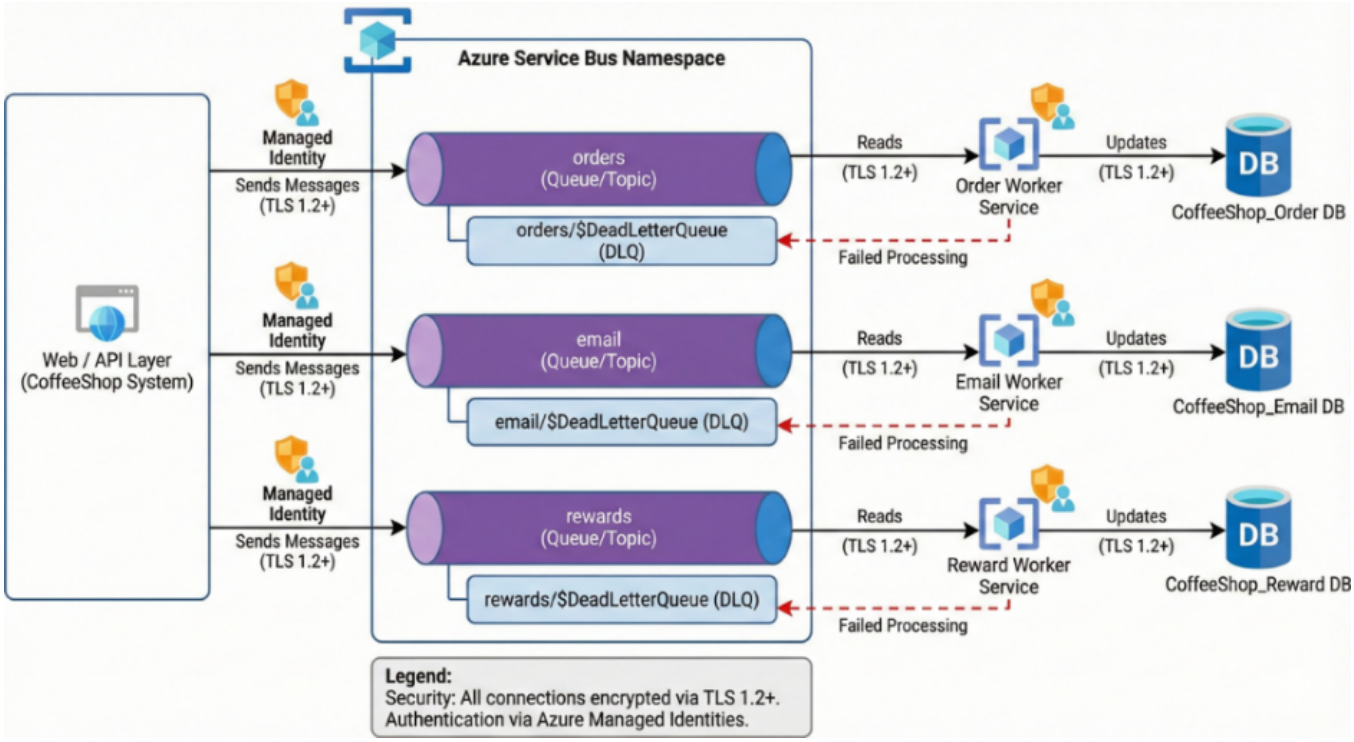


Рисунок 2.5 – Організація асинхронної обробки подій у системі «CoffeeShop» за допомогою Azure Service Bus queues/topics та робітників-споживачів

Таблиця 2.1

Порівняння синхронної та асинхронної взаємодії в системі «CoffeeShop»

Аспект	Синхронна взаємодія (прямий HTTP-запит)	Асинхронна взаємодія через Azure Service Bus
Зв’язність компонентів	Сильна: клієнт очікує негайної відповіді від бекенду	Слабка: клієнт лише публікує подію, обробка виконується окремо
Стійкість до пікових навантажень	Низька: перевантаження бекенду → зростання затримок	Вища: черги згладжують сплески навантаження
Обробка довготривалих операцій	Ускладнена, ризик тайм-аутів	Зручна: операції виконуються у фоні worker-сервісами
Ризики повторної обробки	Контролюються на рівні транзакцій БД	Керуються через idempotency + dead-letter черги
Вимоги до доступності окремих сервісів	Критичні: відмова бекенду → помилка запиту клієнта	Гнучкіші: тимчасова недоступність worker’а не блокує прийом замовлень

2.3 Управління ідентифікацією та секретами (Identity Management)

Відмова від статичних паролів: впровадження Workload Identity (Managed Identity).

У традиційних вебсистемах доступ до зовнішніх сервісів та баз даних часто реалізується за допомогою статичних облікових даних — паролів, connection string’ів або ключів доступу, які зберігаються в конфігураційних файлах чи змінних середовища. Такий підхід створює значні ризики безпеки: витік цих секретів, наприклад через репозиторій коду або журнали, дає зловмиснику змогу безпосередньо підключатися до критичних ресурсів, оминаючи прикладну логіку системи. У case-стаді системи «CoffeeShop» ці ризики є особливо чутливими для доступу до Azure SQL Database, Azure Blob Storage, Azure Service Bus та Azure Key Vault, через які проходять персональні дані користувачів, історія замовлень і службові повідомлення.

У середовищі Azure Kubernetes Service для усунення залежності від статичних секретів застосовується підхід Workload Identity, що базується на використанні керованих ідентичностей (Managed Identities) Microsoft Entra для окремих робочих навантажень у кластері. Сутність цього підходу полягає в тому, що кожен критично важливий мікросервіс системи «CoffeeShop» (наприклад, Order, Product, Email, Reward) працює від імені власної керованої ідентичності, зв'язаної з відповідним Kubernetes service account, а автентифікація до сервісів Azure здійснюється за допомогою короткоживучих токенів, виданих Microsoft Entra ID, без зберігання паролів або ключів у конфігурації контейнера.

Робота Workload Identity базується на інтеграції Kubernetes service account із об'єктом керованої ідентичності в Microsoft Entra ID: у кластері створюється service account з анотаціями, які вказують, з якою Entra-ідентичністю він пов'язаний, а на вузлах AKS розгортається спеціальний брокер токенів (workload identity webhook/agent), що видає pod'ам короткоживучі OIDC-токени. Під час звернення до Azure SDK або REST-API контейнер отримує токен доступу, підписаний Entra ID, на основі свого service account, після чого цей токен перевіряється цільовим сервісом (Azure SQL, Storage, Service Bus чи Key Vault), який приймає або відхиляє запит згідно з призначеними ролями Azure RBAC, не вимагаючи жодних паролів чи connection string'ів у конфігурації.

Завдяки Workload Identity реалізується більш тонке розмежування прав доступу: кожній керованій ідентичності призначаються лише ті ролі Azure RBAC, які відповідають бізнес-функціям конкретного мікросервісу (наприклад, Sql DB Contributor для сервісу замовлень, Storage Blob Data Reader для сервісу каталогу, Azure Service Bus Data Sender/Receiver для інтеграційних worker'ів). Це не тільки знижує ризик компрометації в разі захоплення окремого пода, а й спрощує операційне управління доступом: відкликання прав або зміна ролей виконується централізовано на рівні Microsoft Entra та Azure RBAC без модифікації конфігурацій застосунку й перегенерації секретів. Ілюстрація, в якій представлено використання Microsoft Entra Workload Identity для доступу мікросервісів системи «CoffeeShop» до ресурсів Azure без статичних секретів на рис. 2.6.

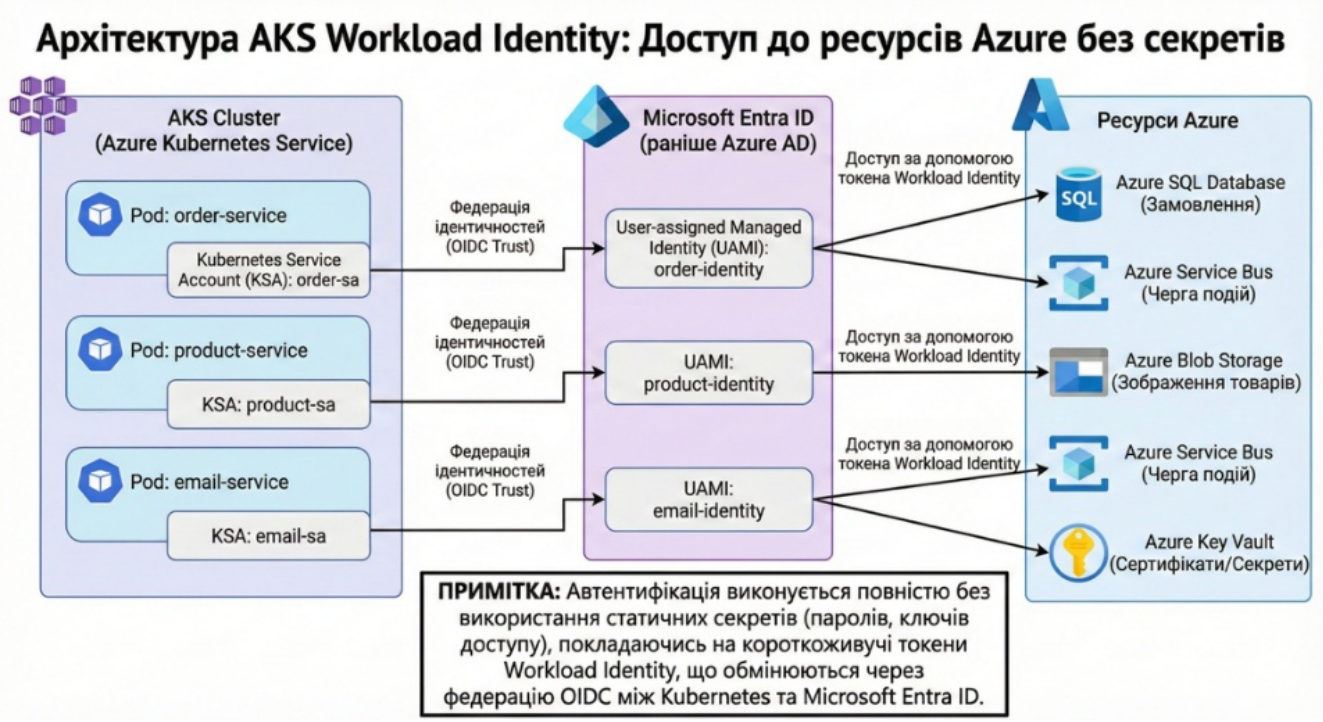


Рисунок 2.6 – Використання Microsoft Entra Workload Identity для доступу мікросервісів системи «CoffeeShop» до ресурсів Azure без статичних секретів

Інтеграція Azure Key Vault для зберігання критичних секретів

Попри використання Workload Identity, у системі «CoffeeShop» залишається низка чутливих даних, які необхідно зберігати у вигляді секретів: криптографічні ключі для підпису токенів, сертифікати TLS, конфіденційні параметри інтеграцій із зовнішніми платіжними системами або сервісами розсилок. Для їхнього захищеного зберігання застосовується Azure Key Vault, який забезпечує централізоване керування секретами, ключами та сертифікатами з апаратним захистом і розмежуванням доступу на основі ролей. Завдяки цьому чутливі дані не потрапляють у конфігураційні файли, змінні середовища чи контейнерні образи, а доступ до них надається лише через захищений API із журналюванням усіх звернень. Ілюстрація, в якій представлено секрети з рядками підключення мікросервісів системи «CoffeeShop» в сховищі Azure Key Vault на рис. 2.7.

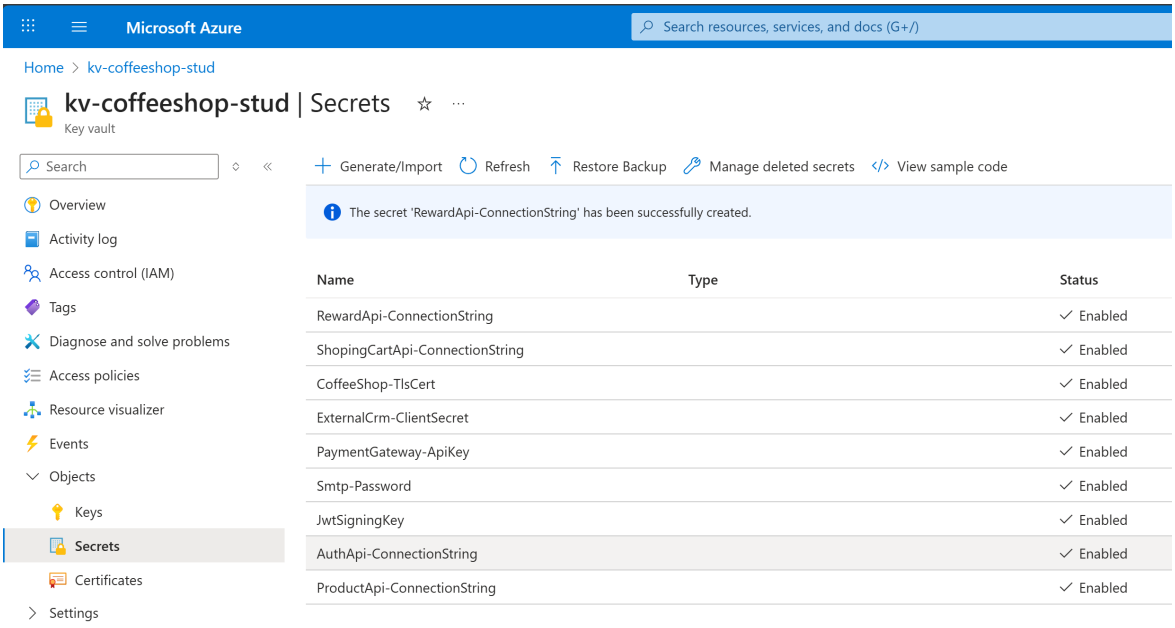


Рисунок 2.7 – Секрети з рядками підключення мікросервісів системи «CoffeeShop» в сховищі Azure Key Vault kv-coffeeshop-stud

Окремі мікросервіси системи «CoffeeShop», що потребують доступу до секретів (наприклад, сервіс автентифікації для ключа підпису JWT або сервіс email-розсилок для параметрів SMTP), автентифікуються до Key Vault через власні керовані ідентичності. Кожна критична мікросервіс отримує індивідуально призначену роль Key Vault Secrets User на рівні сховища Key Vault: mi-coffeeshop-auth, mi-coffeeshop-email, mi-coffeeshop-gateway та mi-coffeeshop-messagebus, як показано на рисунку 2.9. Kubernetes service account кожного мікросервісу пов'язується зі своєю керованою ідентичністю через механізм Workload Identity, що забезпечує федеративну автентифікацію через Microsoft Entra ID. При звертанні контейнера до Key Vault система генерує короткоживучий токен, закріплений за конкретною ідентичністю, що дозволяє отримати доступ лише до тих секретів, для яких призначена ця роль. Таким чином реалізується принцип найменших привілеїв на рівні окремих сервісів: один мікросервіс не може отримати доступ до секретів іншого, а операції створення або оновлення секретів обмежуються адміністративними обліковими записами або конвеєрами CI/CD. Ілюстрація, в якій представлено взаємодія керованих ідентичностей мікросервісів системи «CoffeeShop» з Azure Key Vault при доступі до критичних секретів на рис. 2.8.

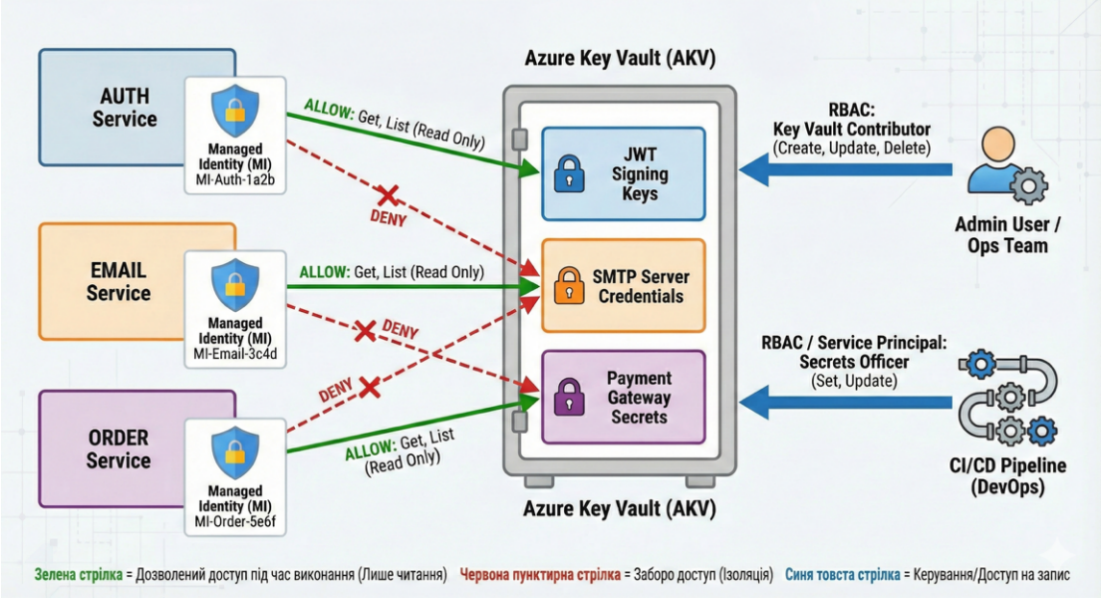


Рисунок 2.8 – Взаємодія керованих ідентичностей мікросервісів системи «CoffeeShop» з Azure Key Vault при доступі до критичних секретів

Використання Azure Container Registry для зберігання образів та Helm-чартів

Окремим елементом інфраструктури керування ідентичністю та секретами в системі «CoffeeShop» є Azure Container Registry, який слугує центральним сховищем контейнерних образів мікросервісів і Helm-чартів, що описують їх розгортання. Через інтеграцію з Microsoft Entra ID та Azure RBAC доступ до ACR здійснюється за допомогою керованих ідентичностей як для конвеєрів CI/CD, що публікують нові версії образів і chart'ів, так і для кластера AKS, який завантажує ці артефакти під час розгортання. Це дозволяє відмовитися від використання статичних облікових даних для доступу до реєстру та забезпечити централізований аудит усіх операцій push/pull, що безпосередньо впливає на загальний рівень безпеки ланцюга поставки ПЗ.

У типовому сценарії оновлення мікросервісу «CoffeeShop» конвеєр CI/CD збирає новий контейнерний образ, завантажує його в ACR, публікує оновлену версію Helm-чарту в тому ж реєстрі та ініціює helm upgrade у кластері AKS, автентифікуючись до реєстру з використанням керованої ідентичності. Таким чином, зв'язка Helm + Azure Container Registry органічно вписується в загальну

модель Workload Identity: як робочі навантаження в кластері, так і інструменти доставки отримують доступ до артефактів без зберігання паролів або ключів, що зменшує площу атаки й спрощує управління секретами в усьому життєвому циклі застосунку.

Рольова модель доступу (RBAC) до ресурсів кластера

Ефективне управління ідентичністю доповнюється коректно налаштованою рольовою моделлю доступу як на рівні Azure, так і всередині кластера Kubernetes. На рівні підписки та ресурсних груп Azure використовується рольова модель на базі Azure RBAC, яка дозволяє розділити доступ між адміністраторами платформи, розробниками, операційною командою та службами автоматизації, надаючи їм лише ті дії, які потрібні для виконання службових обов'язків. Наприклад, адміністратор кластера AKS може мати повний доступ до керування кластером, тоді як розробники обмежуються розгортанням застосунків у певних namespace та переглядом логів, без права змінювати мережеву інфраструктуру або політики безпеки. Ілюстрація, в якій представлено диференційований доступ мікросервісів до Key Vault через окремі керовані ідентичності з роллю Key Vault Secrets User рис. 2.9.

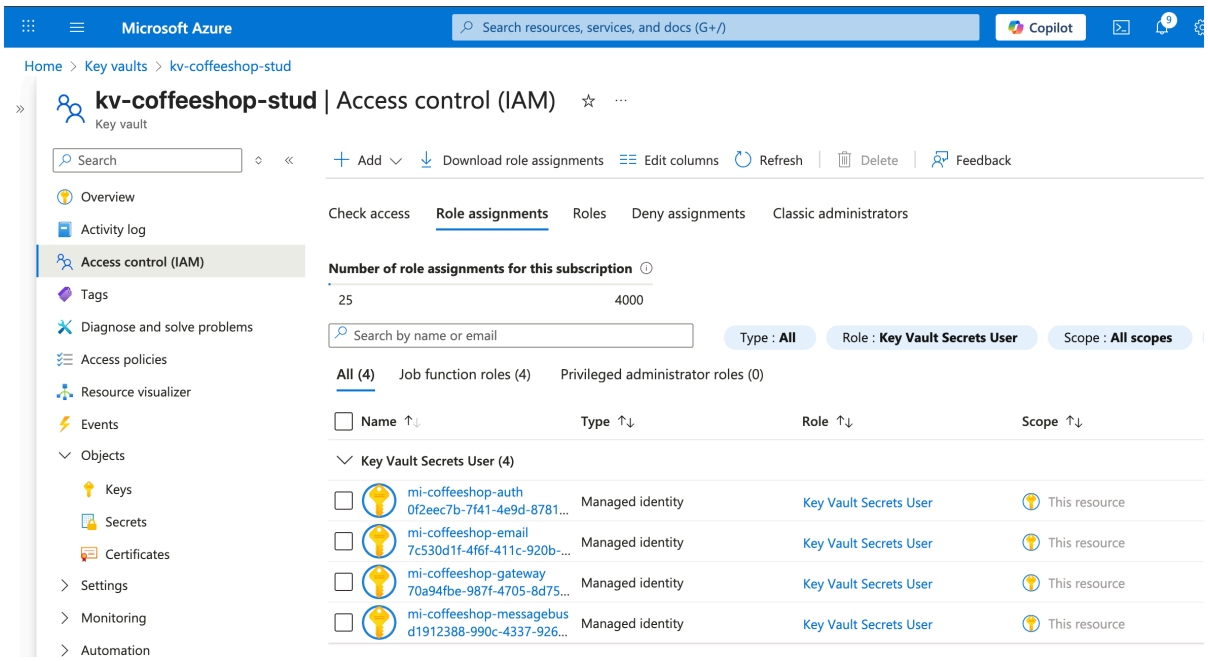


Рисунок 2.9 – Диференційований доступ мікросервісів до Key Vault через окремі керовані ідентичності з роллю Key Vault Secrets User

Всередині самого AKS використовується механізм ролей Kubernetes RBAC у поєднанні з інтеграцією з Microsoft Entra ID, що дозволяє призначати правами доступу до namespaces, ресурсів типу Deployment, Service, ConfigMap, Secret тощо тільки конкретним групам користувачів або сервісних облікових записів. Для системи «CoffeeShop» це означає можливість чіткого розмежування: наприклад, одна роль для розробників, які працюють лише з простором імен dev, інша — для операторів, що мають доступ до логів і подій у production-просторі prod, і окрема роль для автоматизованих конвеєрів CI/CD, які виконують розгортання через Helm, але не можуть змінювати конфігурацію кластера поза межами визначених маніфестів. Такий підхід знижує ризики випадкових або навмисних змін критичної інфраструктури й відповідає принципам Zero Trust та «least privilege», окресленим у попередніх розділах.

Таблиця 2.2

Приклади ролей та прав доступу в системі «CoffeeShop»

Роль / суб'єкт	**Рівень доступу в Azure**	**Рівень доступу в AKS/RBAC**
Адміністратор платформи	Повний доступ до ресурсної групи з AKS, ACR, Key Vault	Кластерна роль з можливістю керування кластерами, namespace та політиками
DevOps / SRE	Керування AKS, ACR, Key Vault, Service Bus у межах RG	Розгортання релізів, перегляд логів, оновлення конфігурацій в prod/dev
Розробник	Обмежений доступ до dev-ресурсів (AKS dev, ACR dev)	Доступ до namespace dev, створення/оновлення Deployment, перегляд логів
CI/CD конвеєр	Доступ до ACR, AKS, іноді Key Vault для секретів деплою	Сервісний account з правами на helm install/upgrade у визначених namespace

Продовження таблиці 2.2

Сервісні ідентичності	Доступ лише до конкретних БД, сховищ, Service Bus, KV	Лише використання секретів/конфігів через Workload Identity, без адмін-прав

Висновки з розділу 2

У другому розділі було виконано проєктування захищеної інфраструктури системи «CoffeeShop» у хмарі Microsoft Azure з урахуванням вимог до масштабованості, відмовостійкості та безпеки cloud-native архітектур. Розглянуто мережеву топологію з використанням Azure Application Gateway та Web Application Firewall для захисту периметра, а також підхід до мікросегментації внутрішнього трафіку в кластері Azure Kubernetes Service на основі Azure CNI та мережевих політик Cilium, що дозволяє обмежити бічний рух потенційного зловмисника між мікросервісами.

Окрему увагу приділено організації зберігання даних та асинхронної комунікації: для структурованих даних обґрунтовано використання Azure SQL Database, для зображень та великих об’єктів – Azure Blob Storage, а для обробки подій і фонових процесів – Azure Service Bus з чергами та топіками, що забезпечує розв’язування компонентів у часі та підвищує стійкість до піків навантаження. У підрозділі 2.3 сформовано цілісну модель управління ідентичністю та секретами, яка поєднує Microsoft Entra Workload Identity, Azure Key Vault, приватний реєстр артефактів Azure Container Registry та рольову модель доступу на основі Azure RBAC і Kubernetes RBAC; це дає змогу відмовитися від статичних облікових даних, централізувати зберігання секретів і чітко розмежувати права користувачів, сервісів і конвеєрів CI/CD. Сукупність цих рішень визначає еталонну інфраструктурну основу, на якій у подальших розділах реалізується й досліджується мікросервісна система «CoffeeShop».

РОЗДІЛ 3. РЕАЛІЗАЦІЯ БЕЗПЕЧНОГО РОЗГОРТАННЯ З ВИКОРИСТАНЯНМ МЕТОДОЛОГІЇ DEVSECOPS НА ПЛАТФОРМІ AZURE

3.1 Організація репозиторію та управління версіями в Azure Repos

Для розробки та супроводу мікросервісної системи «CoffeeShop» використовується система контролю версій Git у складі сервісу Azure Repos, що входить до платформи Azure DevOps. Репозиторій містить вихідний код мікросервісів ASP.NET Core 8, конфігураційні файли інфраструктури (Helm-чарти, YAML-маніфести, пайплайни CI/CD), а також супровідну документацію, що дозволяє централізовано керувати всіма артефактами життєвого циклу застосунку.

Для забезпечення передбачуваності релізів і мінімізації конфліктів застосовується гілкова стратегія, заснована на використанні стабільної основної гілки main та короткоживучих feature-гілок для розробки окремих функцій чи виправлень. Новий функціонал або зміни до інфраструктури виконуються в гілках формату feature/<назва-фічі> чи bugfix/<ідентифікатор-задачі>, після чого інтегруються в main через pull-request із обов’язковим код-рев’ю та автоматичною перевіркою збірки в Azure Pipelines, що відповідає практикам DevSecOps і вимогам до якості коду.

На рівні Azure Repos для гілки main налаштовано політики гілок (branch policies), які забороняють прямі push-операції, вимагають щонайменше один схвалений код-рев’ю та успішне виконання пов’язаного build-пайплайну перед злиттям змін. Додатково можуть застосовуватися політики обов’язкової перевірки за допомогою сканерів безпеки (наприклад, Microsoft Defender for DevOps), що дозволяє виявляти вразливості й помилки конфігурації ще на етапі pull-request і не допускати їх у основну гілку. Ілюстрація, в якій представлено структуру гілок репозиторію CoffeeShop в Azure Repos рис. 3.1.

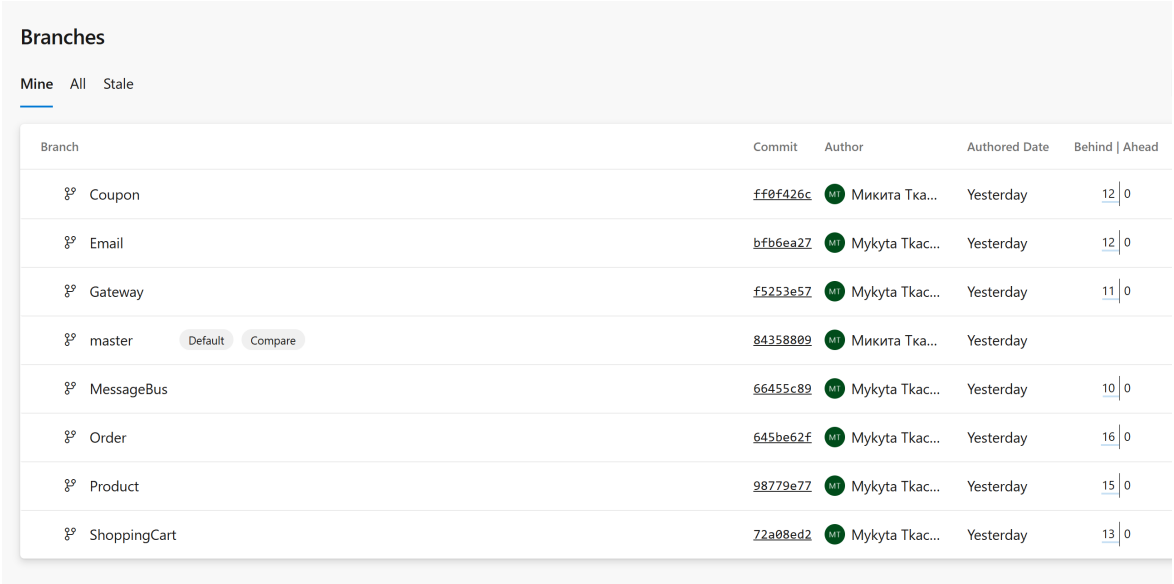


Рисунок 3.1 – Структура гілок репозиторію CoffeeShop в Azure Repos (основна гілка main та короткоживучі feature-гілки)

Політика **Require a minimum number of reviewers** для гілки master встановлює, що кожен pull request має бути схвалений щонайменше двома рецензентами, перш ніж його можна злити в основну гілку. Це означає, що автор змін не може самостійно «протиснути» код в master: система блокує завершення PR, доки два інші учасники проєкту не переглянуть diff, тести та результати CI й явно не натиснуть Approve. Такий підхід зменшує ризик помилок і вразливостей, адже кожна зміна проходить незалежну перевірку щонайменше двома парами очей і відповідає практикам колективного володіння кодом у DevSecOps-процесах.

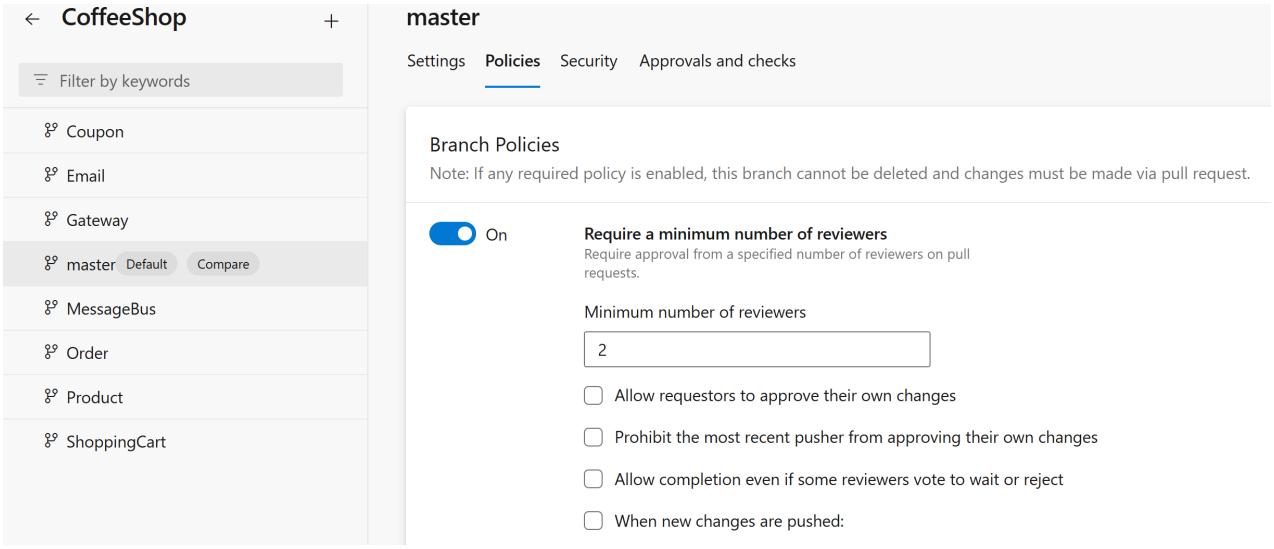


Рисунок 3.2 – Вимога мінімальної кількості рецензентів у політиках гілки master

На цьому екрані налаштовується політика **Check for linked work items** для гілки master, і тут доступні декілька режимів роботи. Режим **Required** блокує завершення pull request, поки до нього не буде прив’язано хоча б один робочий елемент, тоді як режим **Optional** лише показує попередження, але дозволяє завершити PR навіть без прив’язки.

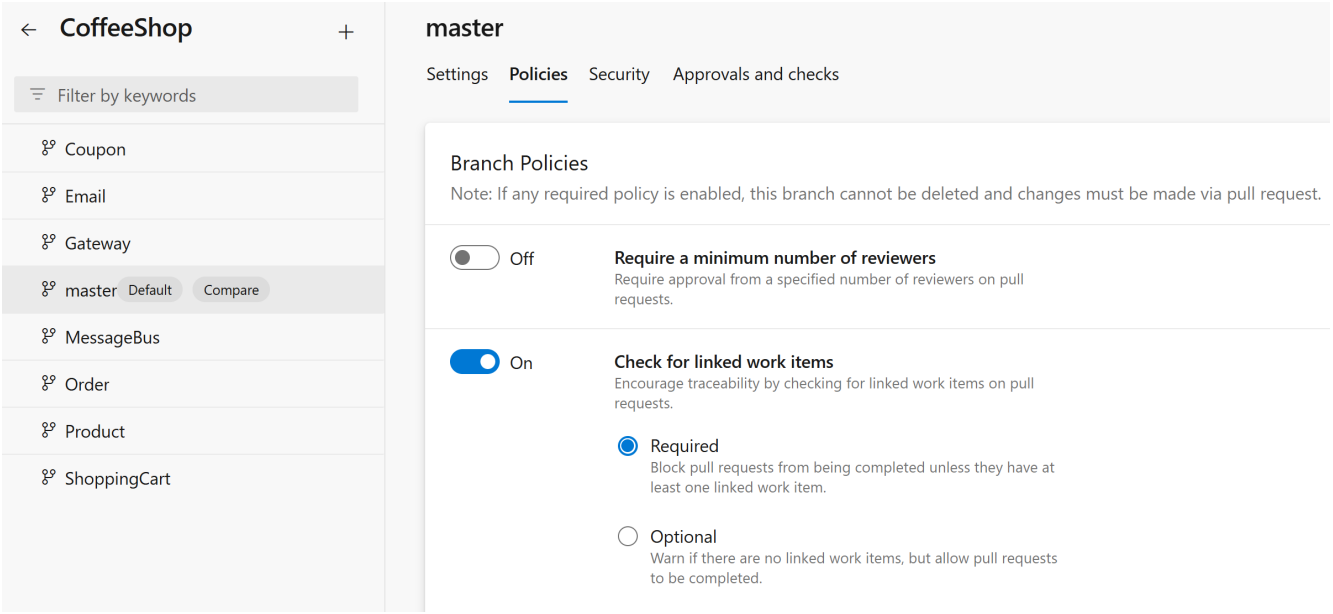


Рисунок 3.3 – Налаштування вимоги прив’язки робочих елементів до pull request

Політика **Check for comment resolution** контролює, чи всі коментарі в pull request були опрацьовані перед злиттям у гілку master. Azure Repos аналізує стан кожного треду коментарів у PR. Нижче доступні два режими: **Required** блокує завершення pull request, поки хоча б один коментар має статус Active і не позначений як resolved, змушуючи авторів змін або виправити код, або явно закрити зауваження; режим **Optional** лише показує попередження про наявність нерозв’язаних коментарів, але не перешкоджає злиттю.

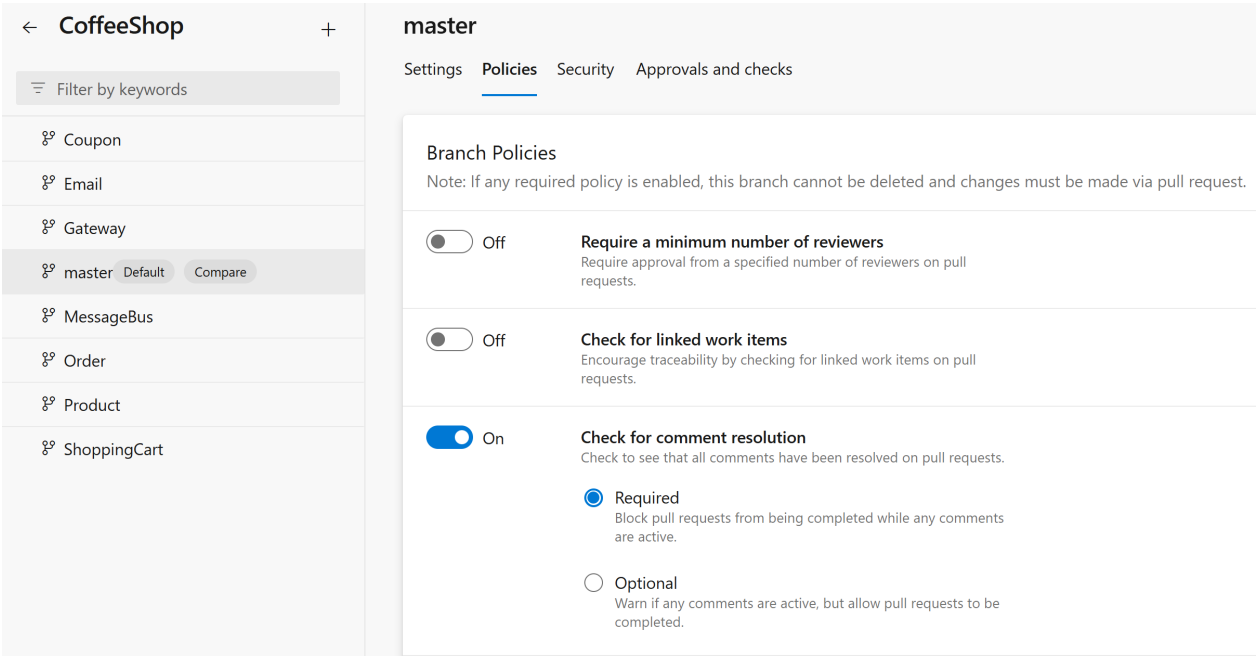


Рисунок 3.4 – Перевірка розв’язання коментарів у pull request у політиках гілки master

Політика **Limit merge types** використовується для контролю історії комітів у гілці master, обмежуючи, які саме варіанти злиття дозволені при завершенні pull request. Розробники можуть завершити PR лише за допомогою тих типів merge, які відмічені нижче в розділі Allowed merge types. У списку доступні кілька опцій: **Basic merge (no fast-forward)** зберігає окремий merge-коміт і точну історію гілкування; **Rebase and fast-forward** перезаписує історію так, ніби всі коміти були зроблені послідовно в master; **Squash merge** об’єднує всі коміти з гілки функціоналу в один коміт у основній гілці, що спрощує історію; **Rebase with merge commit** комбінує переваги rebase і окремого merge-коміту. Обмежуючи набір дозволених режимів, команда може досягти узгодженого стилю історії змін і спростити аналіз аудиту та пошук регресій у master.

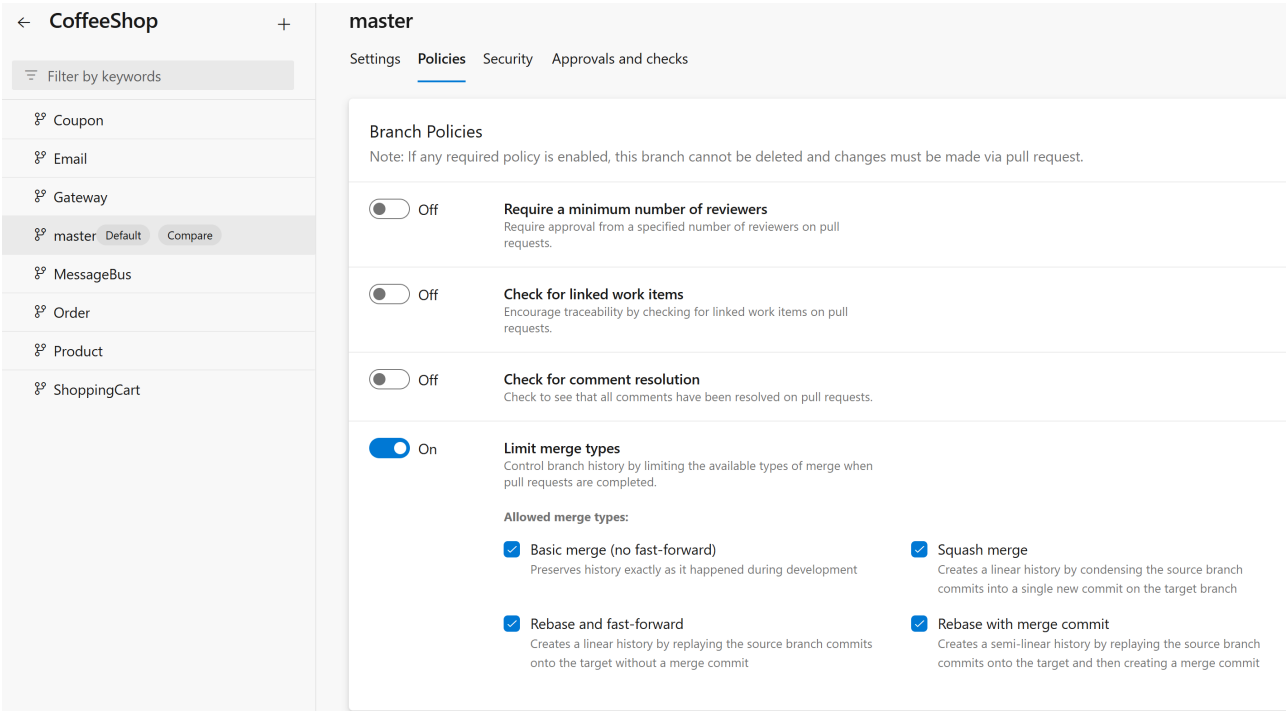


Рисунок 3.5 – Обмеження типів злиття (Limit merge types) для гілки master репозиторію CoffeeShop

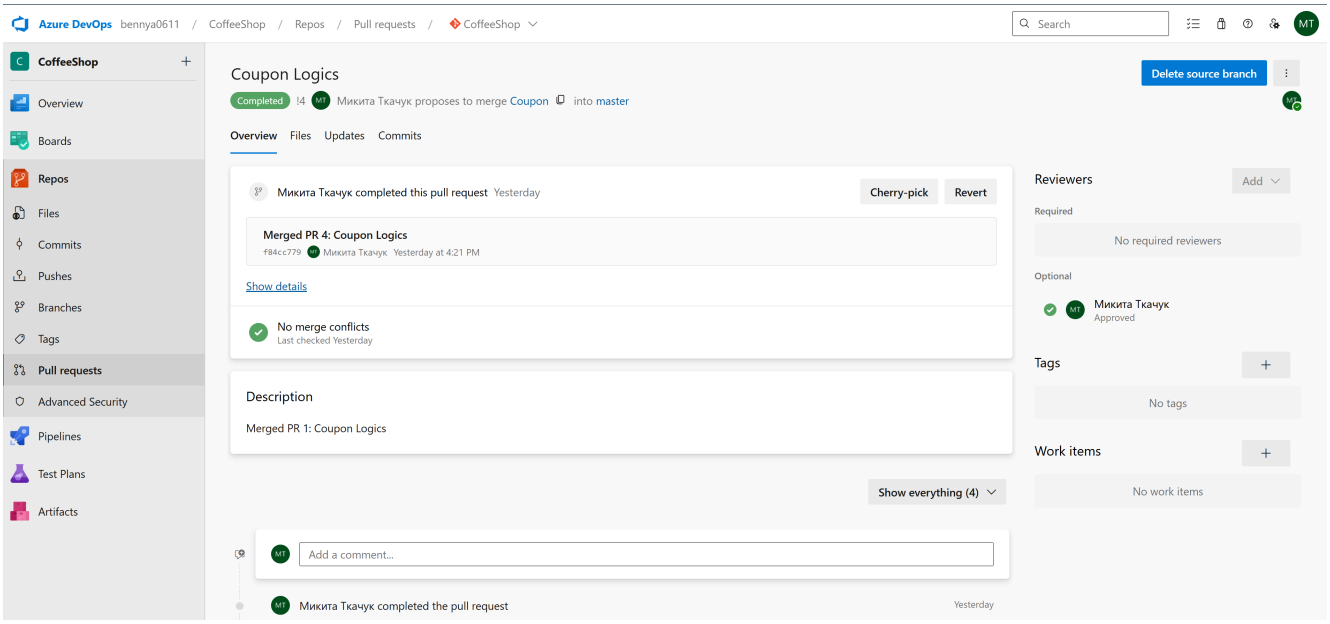


Рисунок 3.6 – Результати перевірок pull request у Azure Repos: обов’язкові вимоги до прив’язки робочих елементів та проходження CI-перевірок

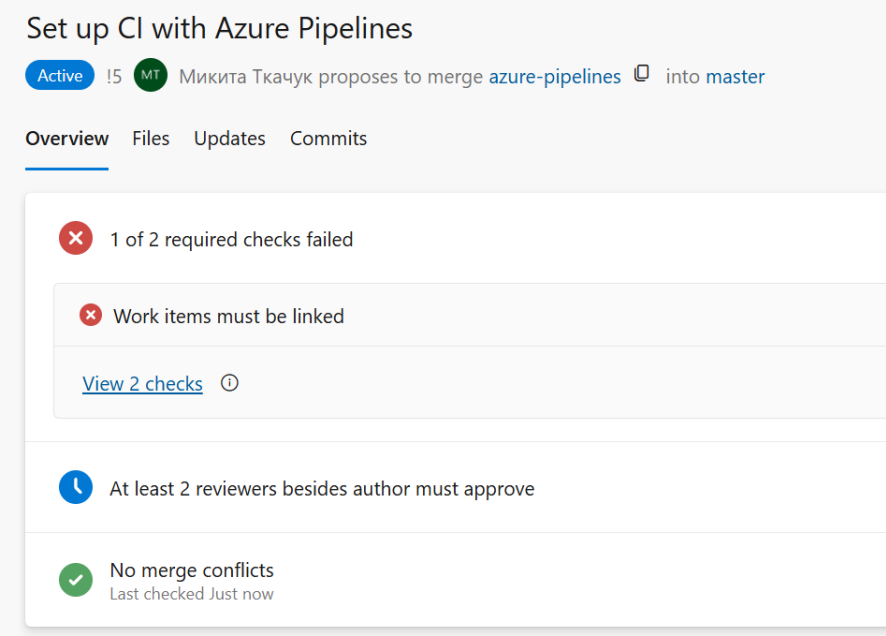


Рисунок 3.7 – Результати перевірок pull request у Azure Repos: обов’язкові вимоги до прив’язки робочих елементів та проходження CI-перевірок

3.2 Розробка уніфікованих Helm-чартів для мікросервісів ASP.NET Core 8

Для стандартизації розгортання мікросервісів системи «CoffeeShop» на платформі ASP.NET Core 8 використовується набір уніфікованих Helm-чартів, що зберігаються в репозиторії Azure Repos поряд із вихідним кодом сервісів. Кожен мікросервіс (наприклад, CoffeeShop.Product, CoffeeShop.Order, CoffeeShop.Email) розгортається за допомогою власного чарту, побудованого на спільному шаблоні, який визначає однакові підходи до керування ресурсами, змінними середовища, секретами, пробами готовності та живучості. Це дозволяє зменшити кількість унікальних конфігурацій та спростити підтримку інфраструктури під час еволюції системи.

Базова структура чарту включає директорії templates/ та файл values.yaml, у якому задаються параметри середовища (імена образів з ACR, теги версій, кількість реплік, ресурси CPU/RAM, налаштування ingress тощо). У каталозі templates/ знаходяться узагальнені шаблони Kubernetes-ресурсів: deployment.yaml для опису розгортання контейнерів ASP.NET Core 8, service.yaml для публікації HTTP-ендпойнтів усередині

кластера, ingress.yaml для інтеграції з ingress-контролером і Azure Application Gateway, а також допоміжні маніфести configmap.yaml, secret.yaml та, за потреби, шаблони мережевих політик. Ілюстрація, в якій представлено структуру уніфікованого Helm-чарту та схема розгортання мікросервісу ASP.NET Core 8 системи «CoffeeShop» у кластері Azure Kubernetes Service із використанням Azure Container Registry та параметризованих файлів значень рис. 3.8.

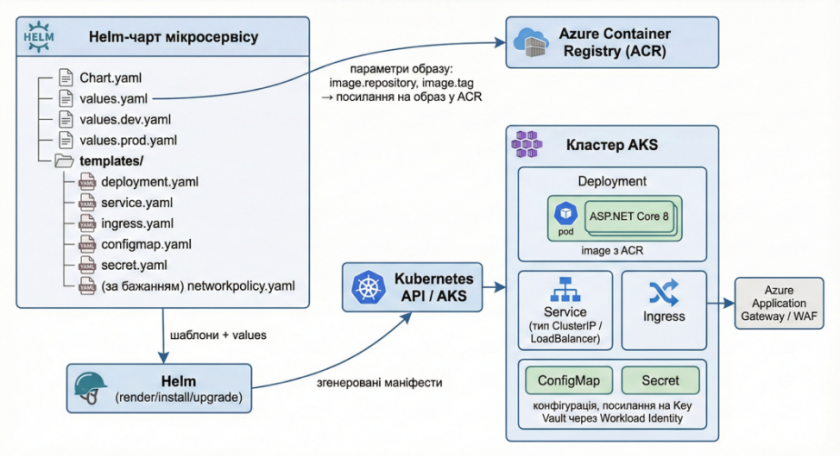


Рисунок 3.8 – Структура уніфікованого Helm-чарту та схема розгортання мікросервісу ASP.NET Core 8 системи «CoffeeShop» у кластері Azure Kubernetes Service із використанням Azure Container Registry та параметризованих файлів значень

Шаблон deployment.yaml містить параметризовані секції для образу контейнера (посилання на репозиторій Azure Container Registry та тег версії), налаштування змінних середовища, прив'язаних до секретів у Azure Key Vault через Workload Identity, та проб liveness/readiness, що відповідають специфіці ASP.NET Core 8-застосунків. У шаблоні service.yaml стандартизовано тип сервісу (ClusterIP для внутрішніх API, NodePort/ingress-маршрути для зовнішнього доступу через Application Gateway), а в ingress.yaml – єдиний підхід до побудови шляхів, TLS-термінації та анотацій для інтеграції з WAF. Така уніфікація Helm-чартів спрощує побудову конвеєрів CI/CD та знижує ризик помилок під час розгортання нових версій мікросервісів у кластері AKS. Ілюстрація, в якій представлено побудову Docker-образу на основі відповідного Dockerfile рис. 3.9.

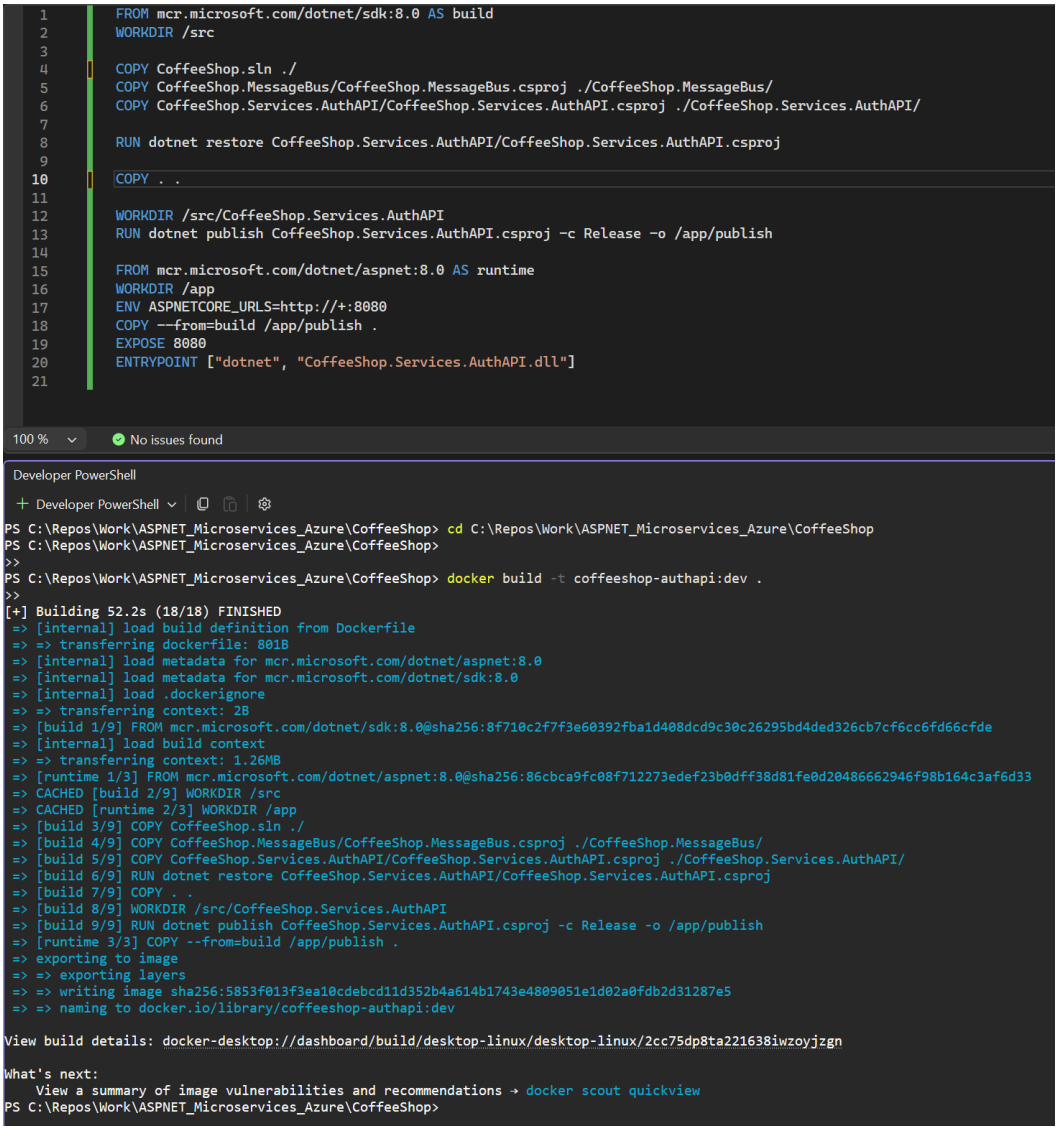


Рисунок 3.9 – Побудова Docker-образу на основі відповідного Dockerfile

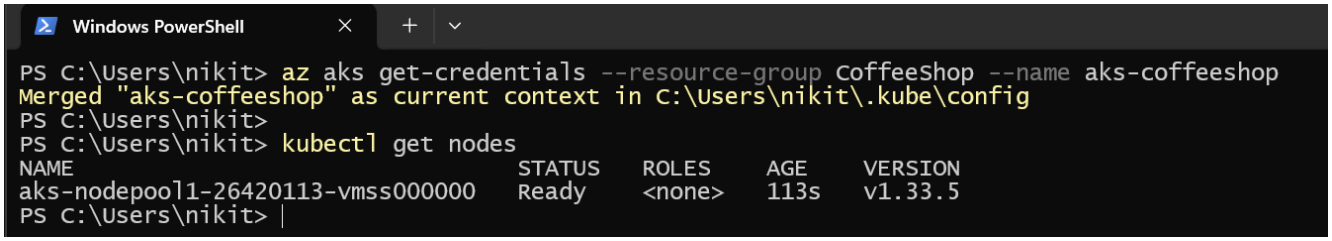


Рисунок 3.10 – Отримання облікових даних доступу до кластера AKS та перевірка стану вузлів за допомогою kubectl

```
PS C:\Users\nikit> az provider register --namespace Microsoft.ContainerService
Registering is still on-going. You can monitor using 'az provider show -n Microsoft.ContainerService'
PS C:\Users\nikit> az provider show --namespace Microsoft.ContainerService --query "registrationState" -o tsv
Registering
PS C:\Users\nikit> az aks create `
>>   --resource-group CoffeeShop `
>>   --name aks-coffeeshop `
>>   --location switzerlandnorth `
>>   --node-count 1 `
>>   --node-vm-size Standard_B2s `
>>   --attach-acr coffeeshopacrstud `
>>   --generate-ssh-keys
| | CreatingAgentPools: 1/1 nodes completed ..
```

Рисунок 3.11 – Створення кластера Azure Kubernetes Service з прив’язкою до реєстру контейнерів за допомогою Azure CLI

```
Developer PowerShell
+ Developer PowerShell
PS C:\Repos\Work\ASPNET_Microservices_Azure\CoffeeShop\deploy\authapi> cd C:\Repos\Work\ASPNET_Microservices_Azure\CoffeeShop\deploy\authapi
PS C:\Repos\Work\ASPNET_Microservices_Azure\CoffeeShop\deploy\authapi>
>>
PS C:\Repos\Work\ASPNET_Microservices_Azure\CoffeeShop\deploy\authapi> helm upgrade --install authapi .
>>
Release "authapi" does not exist. Installing it now.
NAME: authapi
LAST DEPLOYED: Wed Dec 17 13:50:12 2025
NAMESPACE: default
STATUS: deployed
REVISION: 1
DESCRIPTION: Install complete
TEST SUITE: None
PS C:\Repos\Work\ASPNET_Microservices_Azure\CoffeeShop\deploy\authapi>
>>
PS C:\Repos\Work\ASPNET_Microservices_Azure\CoffeeShop\deploy\authapi> kubectl get pods
>>
NAME                       READY   STATUS              RESTARTS   AGE
authapi-687d55b465-wwz88   0/1     ContainerCreating   0          0s
PS C:\Repos\Work\ASPNET_Microservices_Azure\CoffeeShop\deploy\authapi> kubectl get svc authapi
>>
NAME      TYPE           CLUSTER-IP    EXTERNAL-IP   PORT(S)        AGE
authapi   LoadBalancer   10.0.187.91   <pending>     80:31301/TCP   1s
PS C:\Repos\Work\ASPNET_Microservices_Azure\CoffeeShop\deploy\authapi> kubectl get pods
>>
NAME                       READY   STATUS    RESTARTS      AGE
authapi-687d55b465-wwz88   1/1     Running   2 (34s ago)   78s
PS C:\Repos\Work\ASPNET_Microservices_Azure\CoffeeShop\deploy\authapi> kubectl get svc authapi
>>
NAME      TYPE           CLUSTER-IP    EXTERNAL-IP     PORT(S)        AGE
authapi   LoadBalancer   10.0.187.91   74.161.115.19   80:31301/TCP   78s
PS C:\Repos\Work\ASPNET_Microservices_Azure\CoffeeShop\deploy\authapi>
>>
```

Рисунок 3.12 – Розгортання мікросервісу authapi за допомогою Helm та перевірка створених pod’ів і сервісу типу LoadBalancer

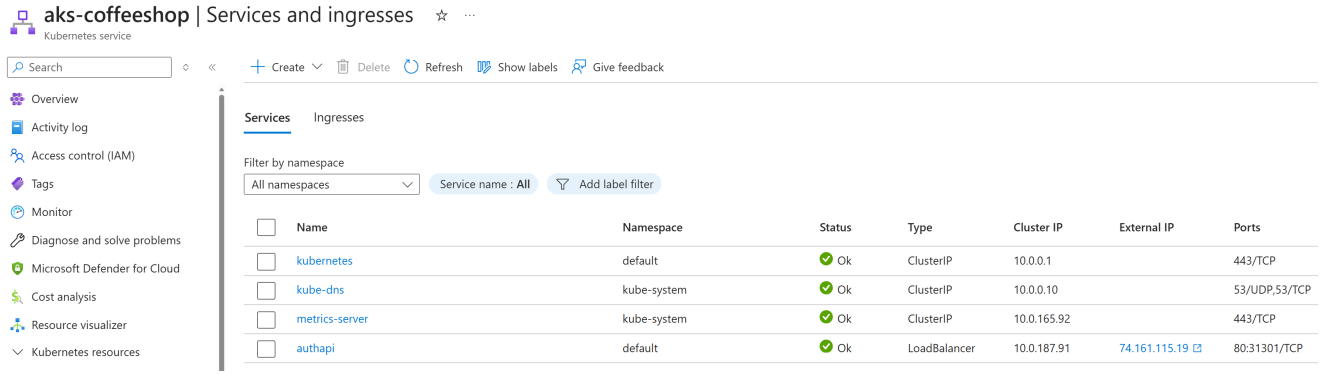


Рисунок 3.13 – Сервіс authapi типу LoadBalancer у кластері aks-coffeeshop, що надає зовнішній доступ до мікросервісу через публічну IP-адресу

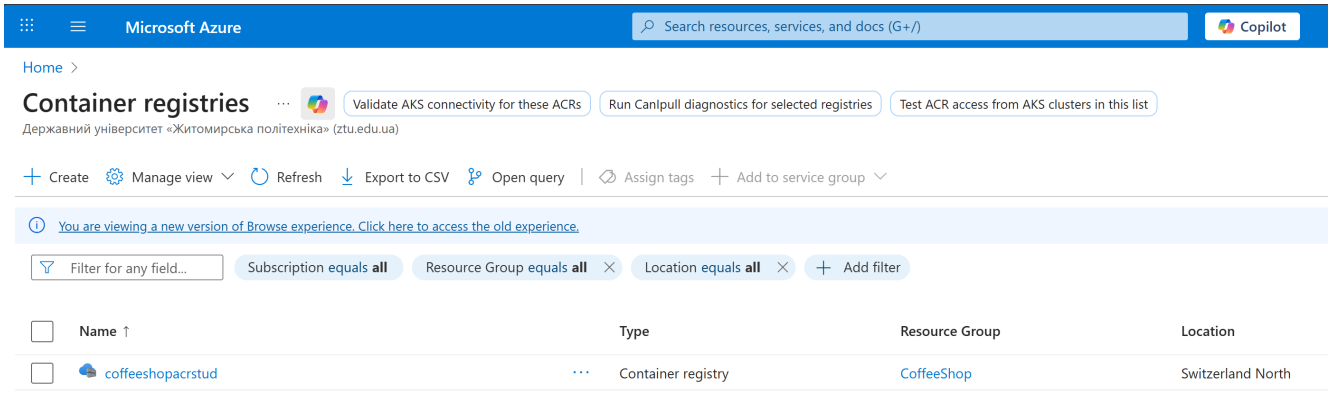


Рисунок 3.14 – Реєстр контейнерів Azure (Azure Container Registry) для зберігання образів мікросервісів системи «CoffeeShop»

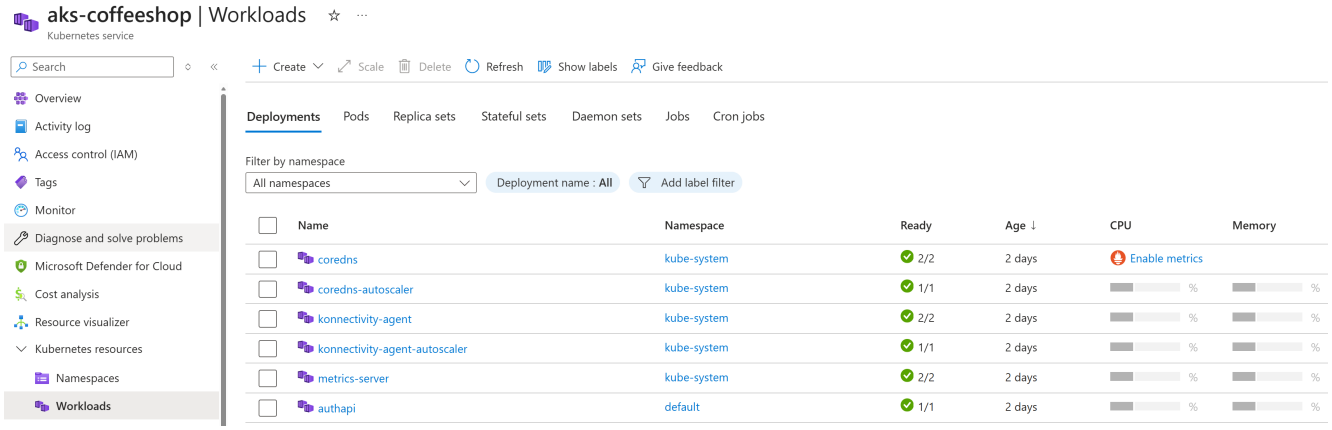


Рисунок 3.15 – Розгортання сервісу та системних компонентів у кластері Azure Kubernetes Service aks-coffeeshop

Окремим аспектом уніфікації є використання спільних «базових» chart'ів або chart-library, на які наслідуються конкретні сервіси «CoffeeShop», що дозволяє винести типові налаштування безпеки та спостережуваності (resource limits, securityContext, sidecar-контейнери з логуванням, анотації для Prometheus тощо) в один спільний шаблон. Для кожного середовища (dev, test, prod) використовуються окремі файли значень (values.dev.yaml, values.prod.yaml), у яких відрізняються лише змінювані параметри – кількість реплік, рівень логування, класи сховищ, доменні імена ingress – тоді як структура Kubernetes-ресурсів залишається сталою, що підвищує передбачуваність розгортань і полегшує аналіз інцидентів.

3.3 Побудова конвеєра CI/CD в Azure DevOps Pipelines

Для автоматизації процесів збірки, тестування та розгортання мікросервісної системи «CoffeeShop» використовується конвеєр CI/CD, реалізований у сервісі Azure DevOps Pipelines. Пайплайни описуються у вигляді YAML-конфігурацій, що зберігаються безпосередньо в репозиторії Azure Repos разом із вихідним кодом і Helm-чартами, що забезпечує прозорість змін конвеєра та застосування тих самих практик код-рев'ю, що й для прикладного коду. Загальна структура конвеєра включає три логічні етапи: етап безпосередньої збірки (CI), етап безпекового аналізу та етап розгортання в кластер Azure Kubernetes Service з використанням Helm.

Етап CI: збірка Docker-образу та публікація в Azure Container Registry

На етапі безперервної інтеграції (CI) конвеєр запускається при кожному push у визначені гілки репозиторію (наприклад, main та release/*) або при створенні pull-request, забезпечуючи автоматичну збірку та перевірку змін. Для кожного мікросервісу ASP.NET Core 8 виконується побудова Docker-образу на основі відповідного Dockerfile, запуск модульних тестів і, за потреби, базових інтеграційних тестів, що дозволяє виявити регресії на ранньому етапі до потрапляння коду в основну гілку. Ілюстрація, в якій представлено YAML-конфігурацію багатостадійного CI/CD-пайплайна в Azure Pipelines для проєкту «CoffeeShop» на рис. 3.16.

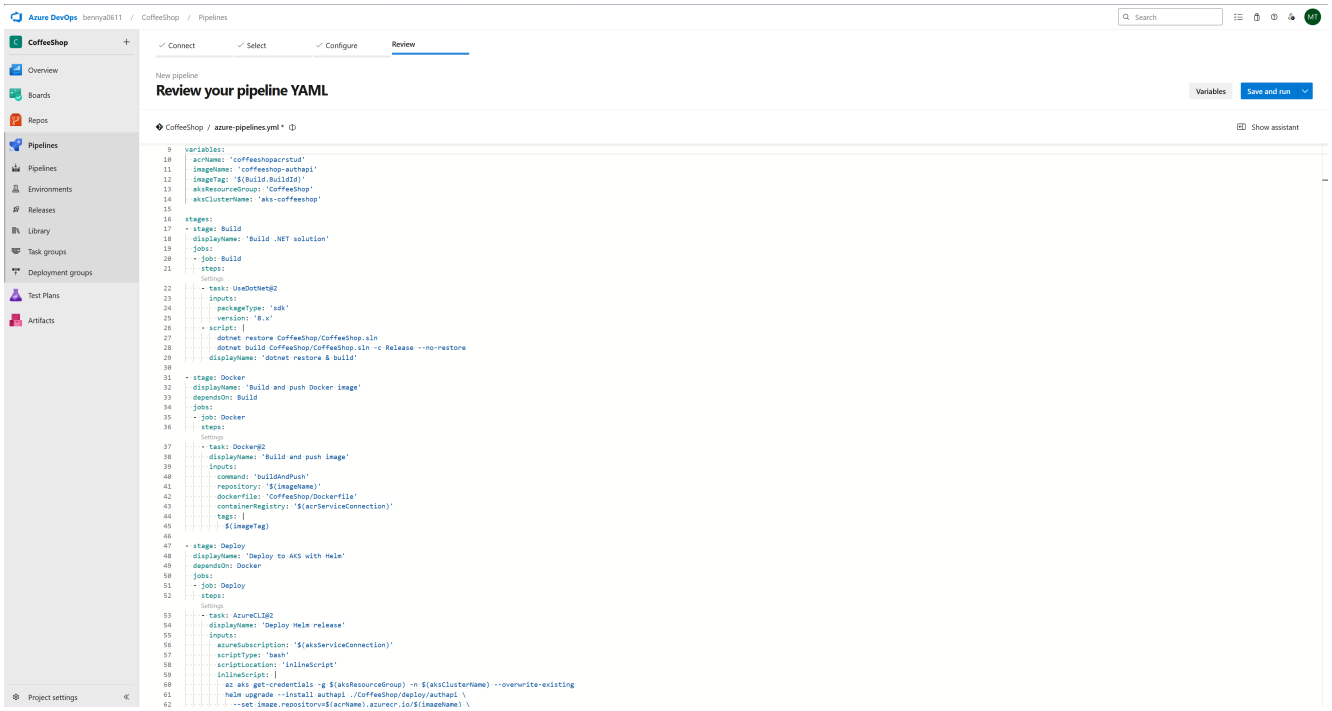


Рисунок 3.16 – Огляд YAML-конфігурації багатостадійного CI/CD-пайплайна в Azure Pipelines для проєкту «CoffeeShop»

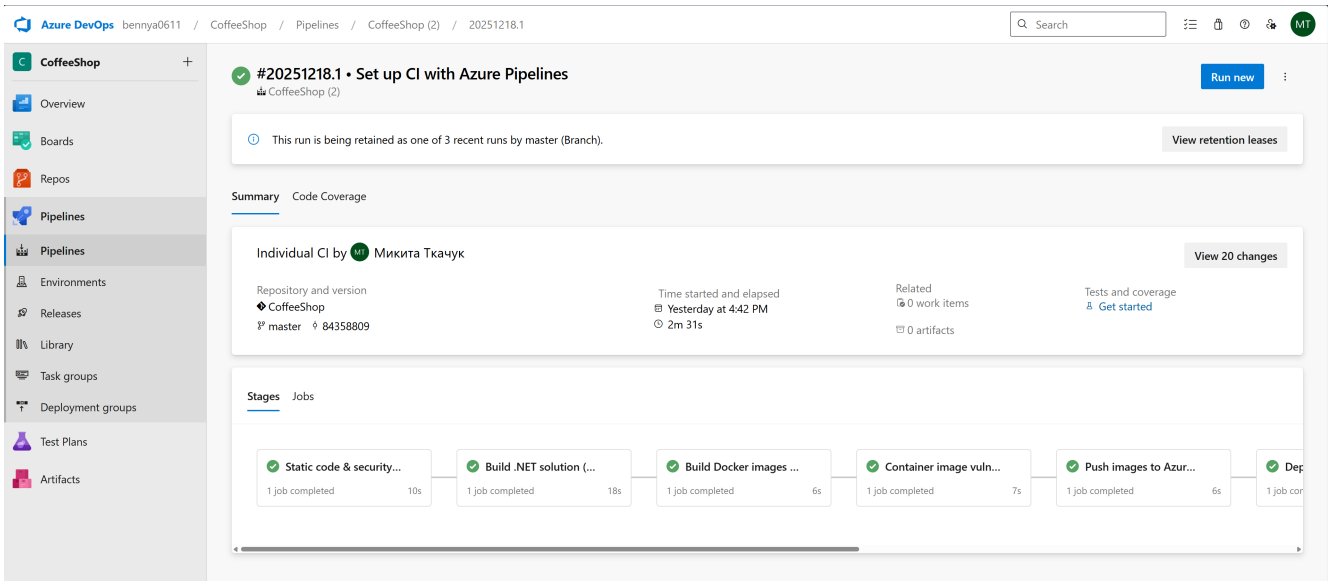


Рисунок 3.17 – Структура виконання CI/CD-пайплайна з послідовними стадіями аналізу, збірки, контейнеризації та розгортання

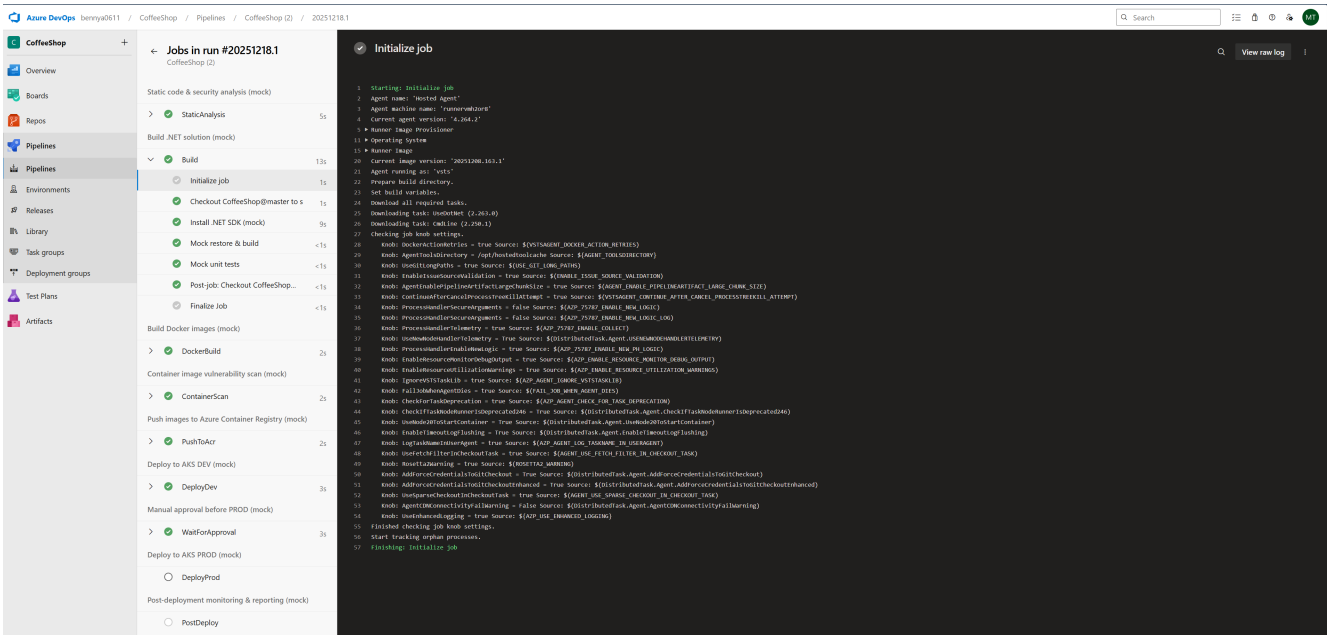


Рисунок 3.18 – Журнал виконання окремого job'а пайплайна в Azure Pipelines із детальною інформацією про середовище збірки

Після успішного проходження тестів Docker-образ тегується уніфікованим тегом (наприклад, номером збірки або значенням GitVersion) і публікується в приватному реєстрі Azure Container Registry за допомогою завдань типу Docker@2 або аналогічних контейнерних задач Azure Pipelines. При цьому автентифікація до ACR виконується через попередньо налаштований service connection, що може ґрунтуватися на керованій ідентичності або сервісному принципалі, а метадані про успішну публікацію образу (наприклад, тег і хеш) передаються в наступні етапи конвеєра або зберігаються як артефакт збірки для подальшого використання на етапі CD. Це забезпечує однозначну відповідність між версією коду, контейнерним образом і конфігурацією розгортання, що є ключовою вимогою до DevSecOps-процесів.

Етап Security: інтеграція Microsoft Defender for DevOps

Для підвищення рівня безпеки конвеєр CI/CD доповнюється окремим етапом аналізу коду та артефактів за допомогою Microsoft Defender for DevOps, який інтегрується з Azure DevOps через спеціальний конектор та розширення Microsoft Security DevOps. Після успішної збірки образу, але до етапу розгортання, пайплайн виконує завдання статичного аналізу коду (SAST), перевірки інфраструктурних

конфігурацій (IaC-сканування Helm-чартів та YAML-маніфестів) і сканування Docker-образу на наявність відомих вразливостей у базових образах та залежностях.

Результати перевірок публікуються в Microsoft Defender for Cloud, де агрегуються за репозиторіями та конвеєрами й враховуються у «secure score» DevOps-середовища, а критичні знахідки можуть блокувати подальше виконання пайплайна згідно з політиками організації. Для проєкту «CoffeeShop» це означає, що релізи, в яких виявлено вразливості високого рівня або критичні помилки конфігурації (наприклад, відсутність шифрування секретів, використання застарілих базових образів чи надто широких привілеїв контейнера), не потрапляють до production-кластера AKS, поки розробники не усунуть проблеми й не перезапустять конвеєр.

Етап CD: автоматизоване розгортання в AKS за допомогою Helm

На етапі безперервного розгортання (CD) конвеєр Azure DevOps Pipelines використовує оновлений Docker-образ із ACR та відповідний Helm-чарт для виконання helm upgrade у цільовому кластері Azure Kubernetes Service. Пайплайн завантажує файл параметрів середовища (values.prod.yaml або values.dev.yaml), у якому оновлюються значення тегів образів, кількість реплік і необхідні налаштування, після чого за допомогою завдань типу HelmDeploy або скриптів CLI виконується розгортання в потрібний namespace кластера. Для зменшення ризику підміни контейнерів у реєстрі ACR у маніфестах і Helm-чартах для продуктивного середовища вказується не лише тег образу, а й його криптографічний digest (sha256), що гарантує розгортання саме того варіанта образу, який пройшов перевірки безпеки в процесі CI. У разі, якщо зловмисник отримає доступ до реєстру й перебудує вразливий образ із тим самим тегом, Kubernetes не оновить розгортання, доки digest у маніфестах не буде змінено, що запобігає непомітному потраплянню зміненого образу до кластера. Автентифікація до AKS здійснюється через service connection, який прив’язаний до керованої ідентичності або сервісного принципала з обмеженими правами, що відповідає принципам least privilege та підходу Workload Identity, описаному у попередньому розділі.

Після застосування змін конвеєр може виконувати додаткові перевірки стану розгорнутих ресурсів: очікування успішного оновлення Deployment (усі pod’и в стані Ready), виконання smoke-тестів через зовнішню точку входу (Application Gateway/WAF) та запис результатів у систему моніторингу. Такий підхід дозволяє реалізувати повноцінну методологію DevSecOps для системи «CoffeeShop», у якій кожна зміна проходить через послідовні етапи збірки, безпекового аналізу й автоматизованого розгортання, мінімізуючи ручні дії та знижуючи ризик людських помилок. Ілюстрація, в якій представлено узагальнену схема DevSecOps-конвеєра для мікросервісної системи «CoffeeShop» в Azure DevOps Pipelines на рис. 3.19.

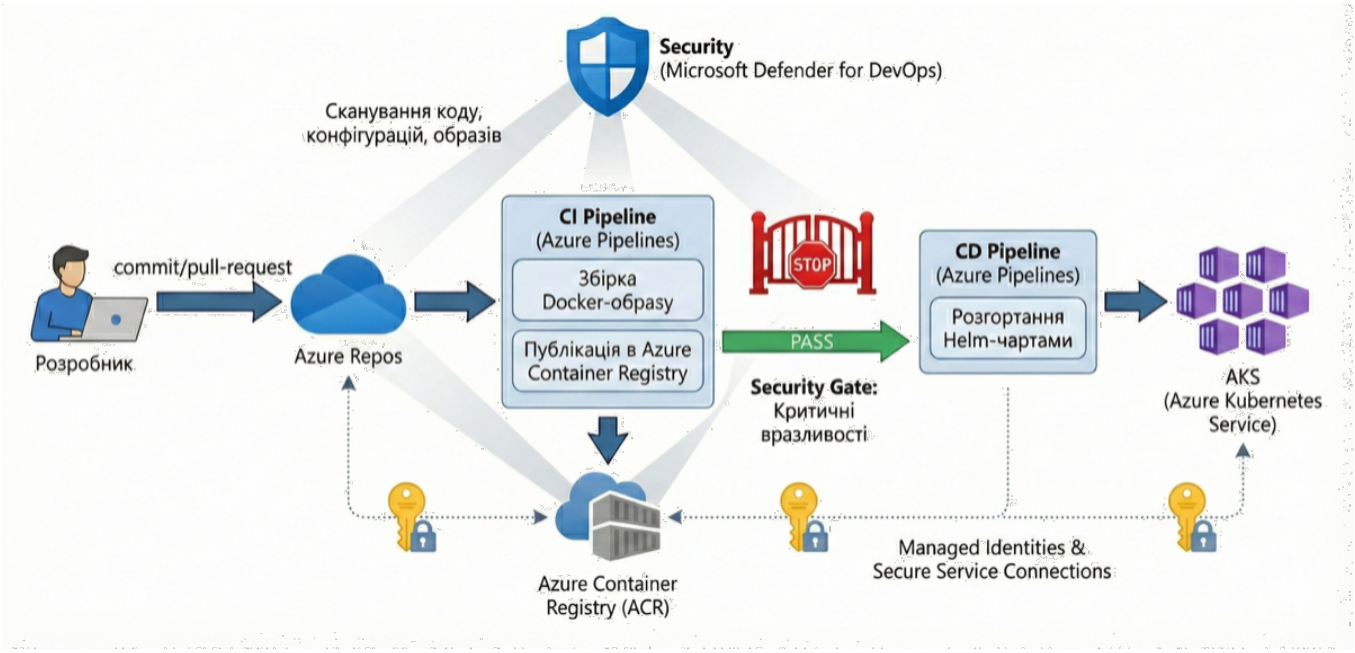


Рисунок 3.19 – Узагальнена схема DevSecOps-конвеєра для мікросервісної системи «CoffeeShop» в Azure DevOps Pipelines

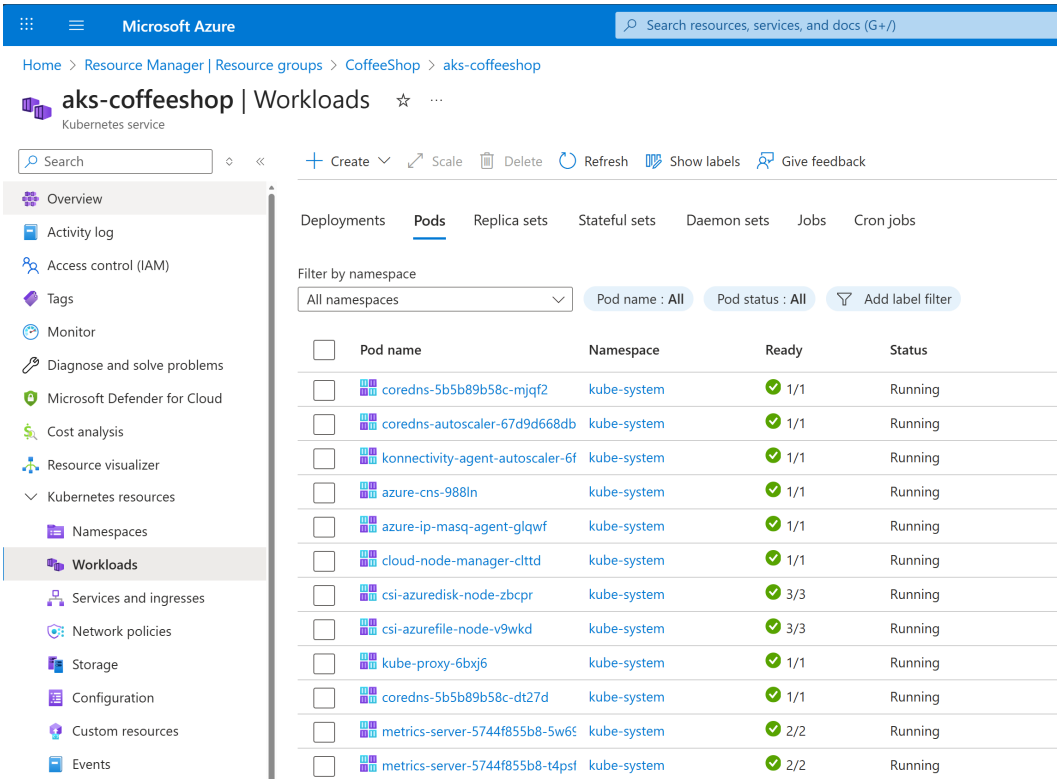


Рисунок 3.20 – Pod’и кластера aks-coffeeshop, включно з мікросервісами, розгорнутими через Helm-чарт

Висновки з розділу 3

У третьому розділі розроблено методологію безпечного розгортання мікросервісної системи «CoffeeShop» на основі практик DevSecOps з використанням Azure DevOps, Azure Container Registry та Azure Kubernetes Service. Описано організацію репозиторію в Azure Repos з гілковою стратегією Git і політиками гілок, що забезпечують контрольоване внесення змін, код-рев’ю та автоматичну валідацію збірки перед злиттям в основну гілку.

Сформовано підхід до створення уніфікованих Helm-чартів для мікросервісів ASP.NET Core 8, які описують типову структуру Kubernetes-ресурсів і використовують параметризовані файли значень для різних середовищ з інтеграцією з приватним реєстром Azure Container Registry. На цій основі побудовано конвеєр CI/CD в Azure DevOps Pipelines зі стадіями збірки й публікації Docker-образів, безпекового аналізу та автоматизованого розгортання в кластер AKS за допомогою Helm. Сукупність цих рішень формує відтворюваний DevSecOps-процес.

РОЗДІЛ 4. РЕАЛІЗАЦІЯ КОМПЛЕКСНОЇ СИСТЕМИ МОНІТОРИНГУ ТА ВИЯВЛЕННЯ ЗАГРОЗ

4.1 Забезпечення відповідності політикам безпеки (Governance)

Для забезпечення відповідності конфігурацій кластера Azure Kubernetes Service та пов’язаних ресурсів корпоративним і галузевим вимогам безпеки в системі «CoffeeShop» використовується сервіс Azure Policy, який дозволяє централізовано визначати, призначати й контролювати політики для ресурсів у межах підписок і груп керування. У рамках Azure Policy застосовуються як окремі політики (policy definitions), що описують окремі вимоги безпеки, так і ініціативи (policy initiatives), які групують політики за певними стандартами, зокрема ISO 27001, PCI DSS, NIST SP 800-53, Azure Security Benchmark. Для системи «CoffeeShop» доцільно використовувати ініціативи Azure Security Benchmark та ISO 27001 як базовий набір загальних вимог до безпеки, а також окремі політики з ініціативи PCI DSS, що стосуються шифрування даних, журналювання доступу та контролю мережевих з’єднань, оскільки вони є релевантними для будь-яких платіжних і клієнтських даних. Це дає змогу не просто окремо перевіряти налаштування ресурсів, а й оцінювати загальний рівень відповідності хмарної інфраструктури вимогам конкретного фреймворку, отримуючи агреговані показники стану відповідності та рекомендації щодо усунення виявлених проблем. Ілюстрація, в якій представлено оглядову панель Azure Policy з показниками Policy Compliance, Initiative Compliance та Overall resource compliance на рис. 4.1.

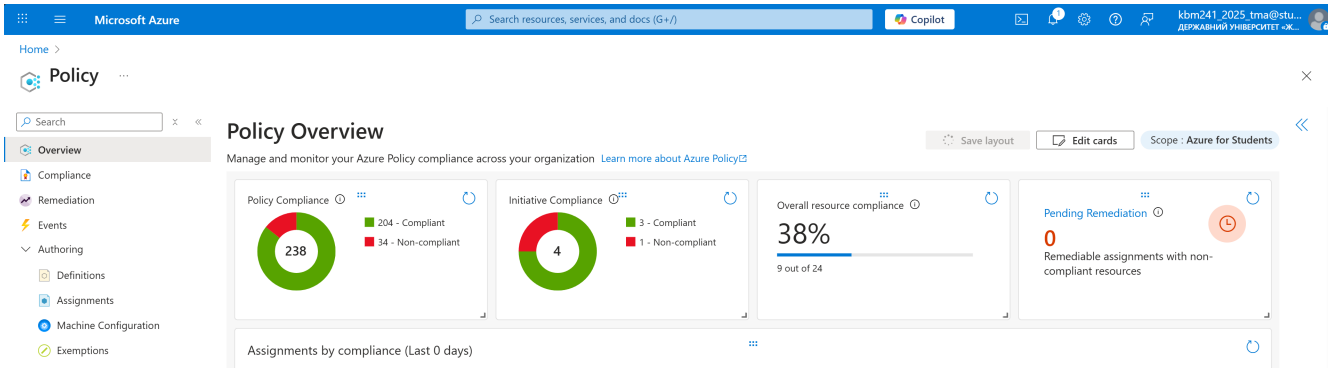


Рисунок 4.1 – Оглядова панель Azure Policy з показниками Policy Compliance, Initiative Compliance та Overall resource compliance для підписки, у якій розгорнута система «CoffeeShop»

Azure Policy підтримує кілька типів ефектів, за допомогою яких визначається реакція на невідповідні ресурси: Audit (лише фіксація порушення в журналі відповідності), Deny (блокування створення або модифікації ресурсу), DeployIfNotExists (розгортання або налаштування додаткових компонентів, якщо вони відсутні), Modify/Append (автоматична модифікація параметрів ресурсу під час створення), Disabled (тимчасове вимкнення політики без видалення її визначення). Такий підхід дозволяє впроваджувати політики поетапно: спочатку у режимі Audit для виявлення поточних відхилень, далі – з ефектами Deny та DeployIfNotExists/Modify для примусового забезпечення відповідності нових і змінених ресурсів, що добре узгоджується з принципами безперервного удосконалення безпеки. На практиці для невиробничих середовищ доцільно починати з ефекту Audit, щоб побачити фактичний стан конфігурацій без ризику порушення роботи сервісів, а вже для production-кластера застосовувати Deny та DeployIfNotExists/Modify, примусово впроваджуючи критично важливі налаштування, наприклад шифрування дисків, вимогу TLS або заборону привілейованих контейнерів.

У контексті Kubernetes ці ефекти застосовуються до кластера AKS через механізм Azure Policy for Kubernetes, який під капотом розгортає у кластері контролер на базі Gatekeeper (Open Policy Agent) та потрібні CustomResourceDefinition для опису обмежень CustomResourceDefinition для опису

обмежень. Під час створення або оновлення pod’ів та інших об’єктів Kubernetes запити до API-сервера перехоплюються admission-контролером Gatekeeper, що порівнює маніфести з визначеними обмеженнями (constraints) і, залежно від ефекту політики, дозволяє операцію, блокує її, або реєструє порушення без блокування. Для системи «CoffeeShop» доцільно використовувати політики, що забороняють запуск привілейованих контейнерів, вимагають задання ресурсних лімітів і requests, обмежують використання нерегламентованих реєстрів образів, перевіряють включення шифрування для сховищ та логів, а також наявність обов’язкових анотацій для інтеграції з моніторингом і журналюванням. У поєднанні з Helm-чартами та мережевими політиками це створює додатковий запобіжний контур: навіть у випадку помилки розробника або неправильних параметрів у файлах values некоректні конфігурації не будуть прийняті кластером, що зменшує ризик появи вразливих або некерованих робочих навантажень. Реалізація інтеграції базується на admission-webhook’ах Kubernetes: коли до API-сервера надходить запит на створення чи зміну об’єкта, Gatekeeper викликається як зовнішній валідатор, розгорнутий у кластері у вигляді Deployment і набору CustomResourceDefinition (ConstraintTemplate та Constraint). Це дозволяє застосовувати політики Azure Policy як до нових, так і до вже наявних ресурсів через періодичний скан стану кластера, результати якого потрапляють до звіту відповідності в Azure Policy. Ілюстрація, в якій представлено схему застосування ініціатив Azure Policy (наприклад, для стандарту ISO 27001) та компонента Azure Policy for Kubernetes (Gatekeeper) для контролю конфігурації кластера AKS і робочих навантажень системи «CoffeeShop» на рівні ефектів Audit, Deny та DeployIfNotExists/Modify на рис. 4.2.

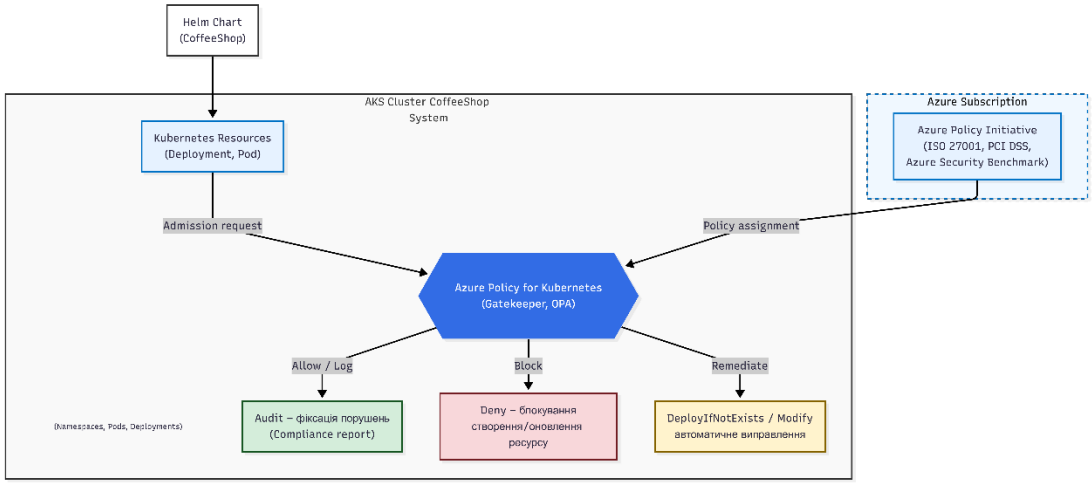


Рисунок 4.2 – Схема застосування ініціатив Azure Policy та компонента Azure Policy for Kubernetes для контролю конфігурації кластера AKS і робочих навантажень системи «CoffeeShop» на рівні ефектів Audit, Deny та DeployIfNotExists/Modify

4.2 Моніторинг стану безпеки та виявлення вразливостей (Runtime Security)

Використання Microsoft Defender for Containers для захисту робочих навантажень

Для захисту контейнеризованих робочих навантажень системи «CoffeeShop» у кластері Azure Kubernetes Service використовується Microsoft Defender for Containers, який є частиною платформи Microsoft Defender for Cloud і надає засоби як для аналізу безпекової позиції, так і для захисту під час виконання. Сервіс забезпечує безагентне сканування образів контейнерів у Azure Container Registry на предмет відомих вразливостей, аналіз конфігурацій кластерів AKS та робочих навантажень на відповідність кращим практикам, а також постійний моніторинг подій Kubernetes для виявлення підозрілої активності. Ілюстрація, в якій представлено панель Microsoft Defender for Cloud із оцінкою secure score та рекомендаціями для ресурсів на рис. 4.3.

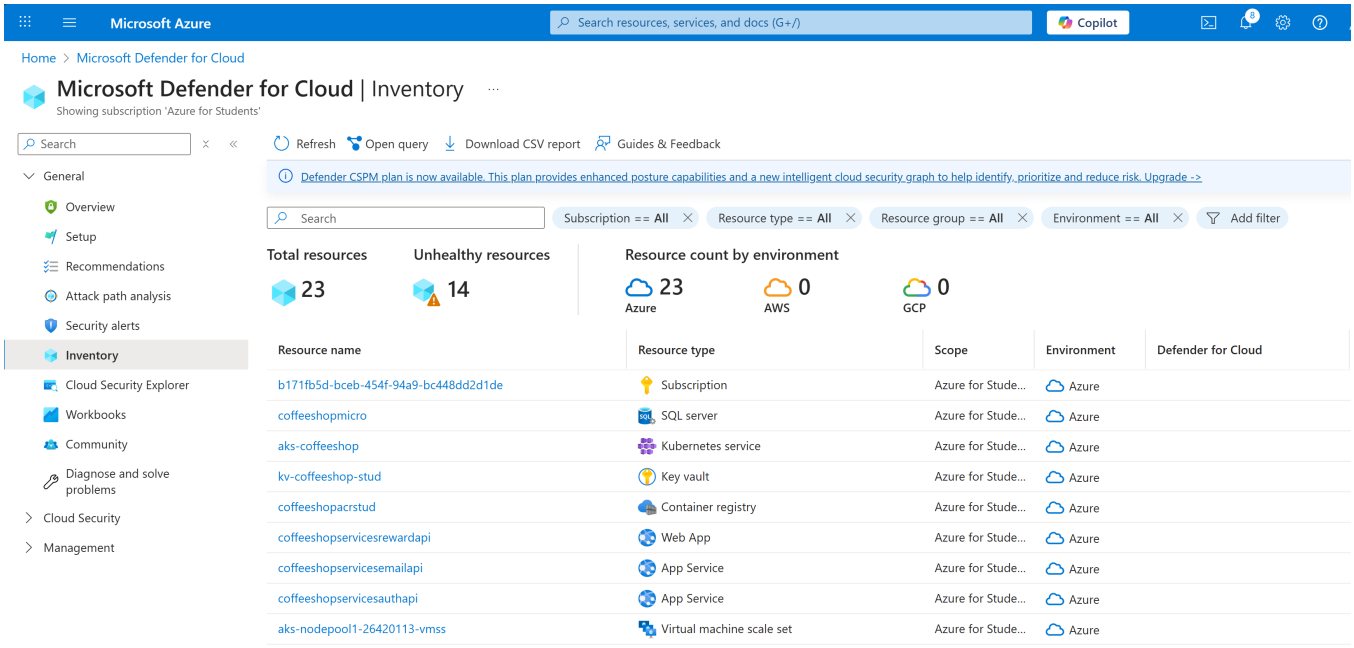


Рисунок 4.3 – Панель Microsoft Defender for Cloud із оцінкою secure score та рекомендаціями для ресурсів системи «CoffeeShop»

У контексті системи «CoffeeShop» Defender for Containers дозволяє відслідковувати, які саме образи мікросервісів розгорнуті в кластері, які з них містять невиправлені вразливості, чи дотримуються вони вимог щодо обмеження привілеїв контейнерів, ресурсних лімітів, використання лише довірених реєстрів і відсутності привілейованих ролей у Kubernetes. На основі зібраних даних сервіс генерує попередження у випадку виявлення небезпечних подій у runtime – наприклад, створення високопривілейованих ролей, спроби запуску контейнерів з доступом до чутливих host-namespaces або аномальної мережевої активності з pod'ів, що дозволяє оперативно реагувати на потенційні атаки та інциденти. Ілюстрація, в якій представлено панель Microsoft Defender for Cloud із переліком сповіщень безпеки (Security alerts) на рис. 4.4.

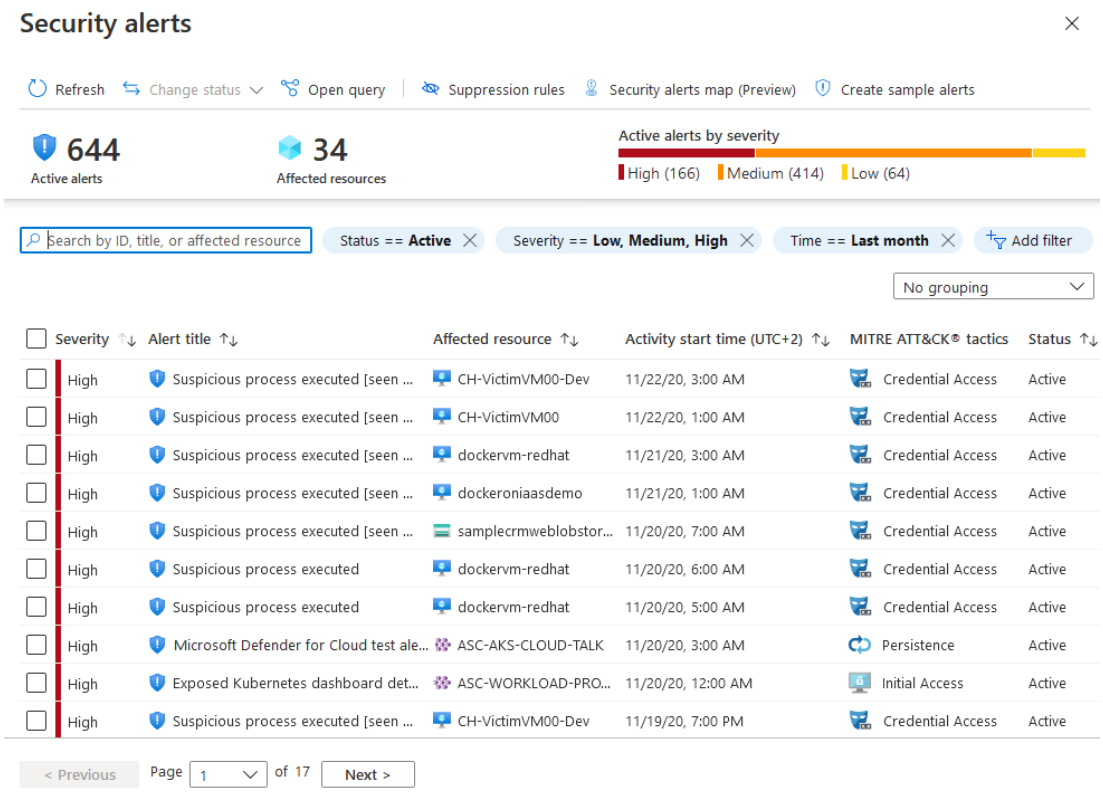


Рисунок 4.4 – Панель Microsoft Defender for Cloud із переліком сповіщень безпеки (Security alerts)

Аналіз рекомендацій безпеки в Microsoft Defender for Cloud

Microsoft Defender for Cloud агрегує результати аналізу всіх ресурсів, пов'язаних із системою «CoffeeShop» (кластер AKS, реєстри контейнерів, сховища даних, Key Vault, мережеві компоненти), та формує перелік рекомендацій, які відображаються в єдиній консолі разом із інтегрованим показником secure score. Для контейнерних середовищ ці рекомендації включають, зокрема, вимогу увімкнути профіль Defender for Containers для всіх кластерів AKS, усунути вразливості в запущених контейнерних образах, забезпечити, щоб контейнери не працювали від імені користувача root, мали встановлені ресурсні ліміти та використовували лише дозволені реєстри. Ілюстрація, в якій представлено виявлені вразливості контейнерних образів системи «CoffeeShop» у Microsoft Defender for Containers на рис. 4.5.

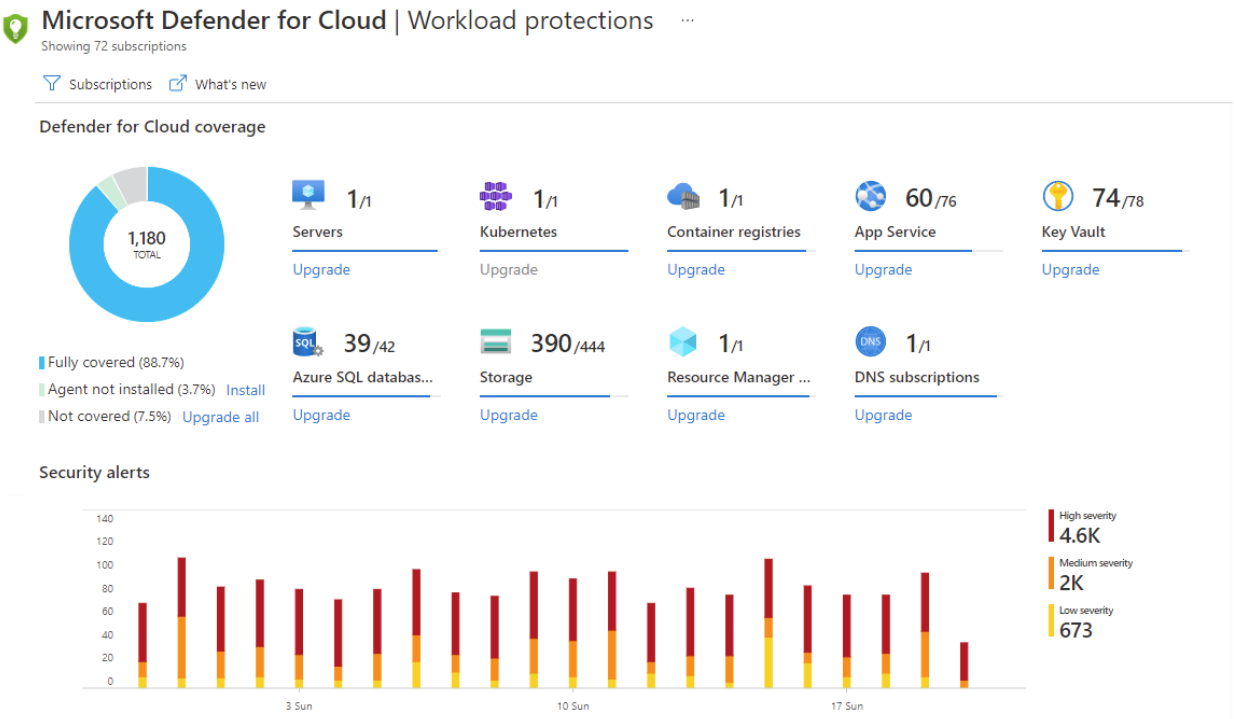


Рисунок 4.5 – Виявлені вразливості контейнерних образів системи «CoffeeShop» у Microsoft Defender for Containers

У разі з системою «CoffeeShop» ці рекомендації слугують основою для циклу безперервного вдосконалення: вони інтегруються у процес роботи DevSecOps-команди, яка регулярно переглядає список активних рекомендацій, пріоритизує їх відповідно до критичності та впроваджує зміни в Helm-чарти, політики Azure Policy або конфігурацію кластера AKS. Таким чином, Microsoft Defender for Containers та Defender for Cloud не лише забезпечують виявлення вразливостей і загроз під час виконання, але й виступають інструментами для системного підвищення рівня безпеки контейнеризованих робочих навантажень упродовж усього життєвого циклу системи. Ілюстрація, в якій представлено схему моніторингу безпеки контейнерних навантажень системи «CoffeeShop» за допомогою Microsoft Defender for Containers та Microsoft Defender for Cloud на рис. 4.6.

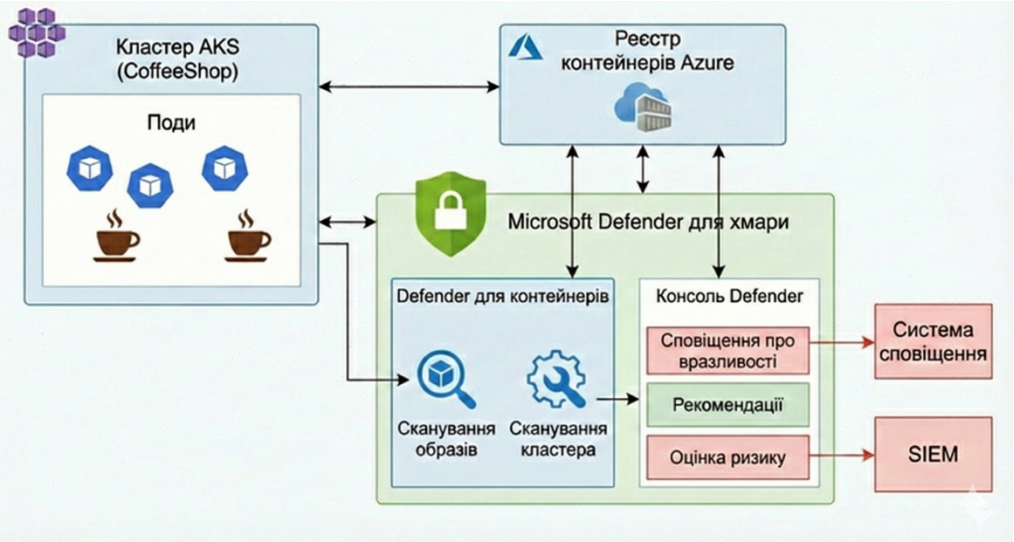


Рисунок 4.6 – Схема моніторингу безпеки контейнерних навантажень системи «CoffeeShop» за допомогою Microsoft Defender for Containers та Microsoft Defender for Cloud

4.3 Побудова системи моніторінгу (Observability)

Збір метрик та логів контейнерів за допомогою Azure Monitor Container Insights

Для забезпечення спостережуваності за роботою мікросервісної системи «CoffeeShop» використовується Azure Monitor з модулем Container Insights, який надає централізований збір метрик і логів з кластерів Azure Kubernetes Service. На вузлах кластера встановлюється агент моніторингу, що зчитує телеметрію з kubelet, контейнерного runtime та стандартних потоків виводу контейнерів, після чого передає її до робочої області Log Analytics, де дані зберігаються та доступні для запитів мовою Kusto Query Language (KQL).

У контексті системи «CoffeeShop» це дозволяє аналізувати завантаження CPU та пам’яті pod’ів, тривалість перезапусків контейнерів, кількість помилок HTTP-запитів на рівні сервісів, а також переглядати журнали застосунків ASP.NET Core 8 незалежно від того, на якому вузлі вони виконуються. На основі цих даних створюються дашборди продуктивності та оповіщення (alerts), які спрацьовують при перевищенні порогових значень (наприклад, тривалий високий рівень помилок 5xx, нестача пам’яті в певному мікросервісі або аномально часті перезапуски

pod'ів), що дозволяє оперативно реагувати на інциденти та планувати масштабування. Ілюстрація, в якій представлено збір метрик та логів мікросервісної системи «CoffeeShop» за допомогою Azure Monitor Application Insights на рис. 4.7.

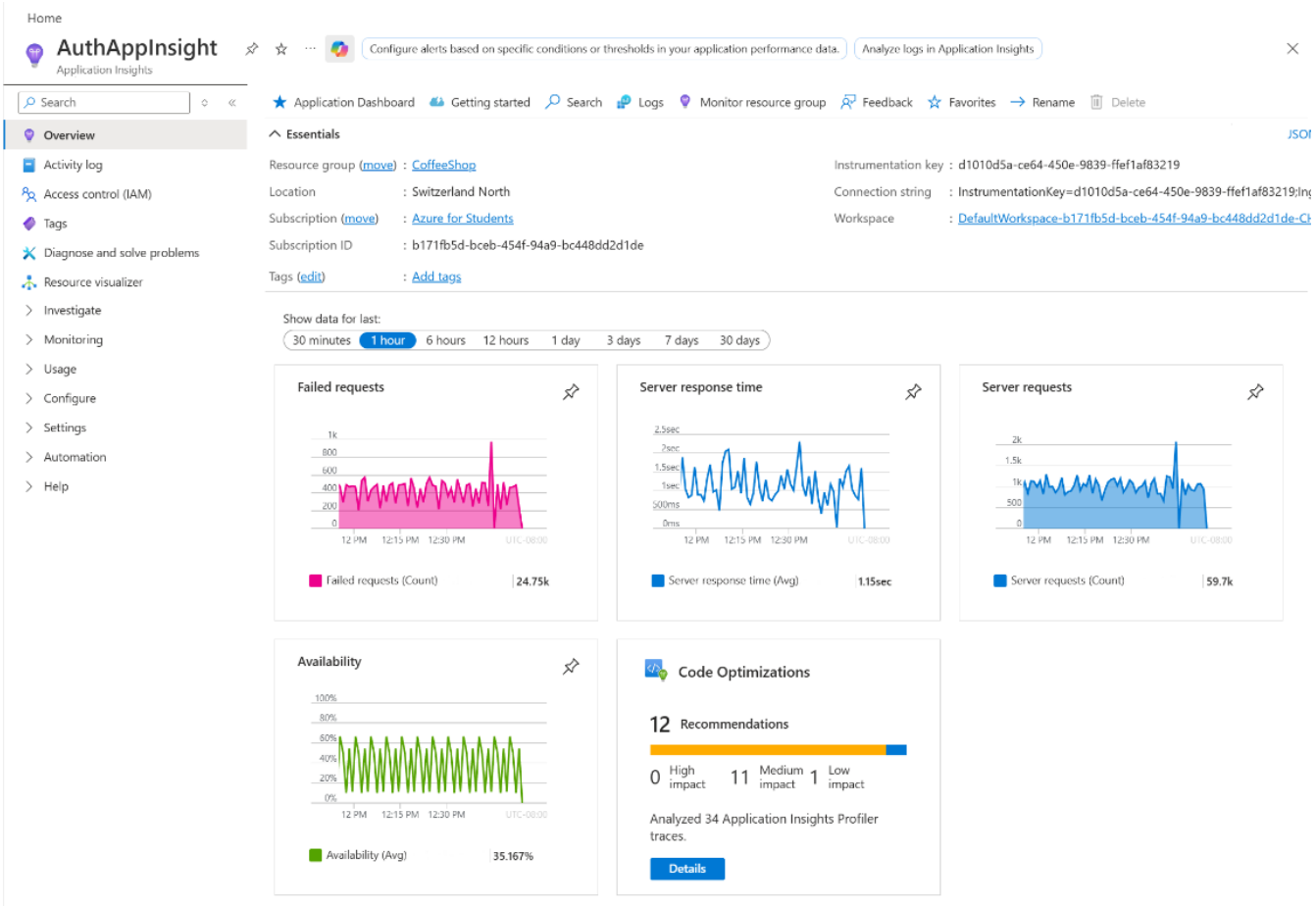


Рисунок 4.7 – Збір метрик та логів мікросервісної системи «CoffeeShop» за допомогою Azure Monitor Application Insights

Трасування розподілених транзакцій за допомогою Application Insights

Для аналізу поведінки розподілених транзакцій між мікросервісами використовується сервіс Application Insights, який інтегрується з ASP.NET Core 8 через відповідні SDK та автоматично збирає дані про запити, залежності, винятки й час відгуку. Завдяки підтримці розподіленого трасування Application Insights дозволяє відстежувати проходження одного логічного запиту користувача через кілька мікросервісів (наприклад, від фронтенду до сервісу кошика, сервісу замовлень і сервісу оплати), пов'язуючи телеметрію за допомогою кореляційних

ідентифікаторів. Ілюстрація, в якій представлено вікно Failures в Application Insights для ресурсу AuthAppInsight, що показує наявність помилкових запитів та дає змогу деталізувати збої за операціями й типами винятків на рис. 4.8.

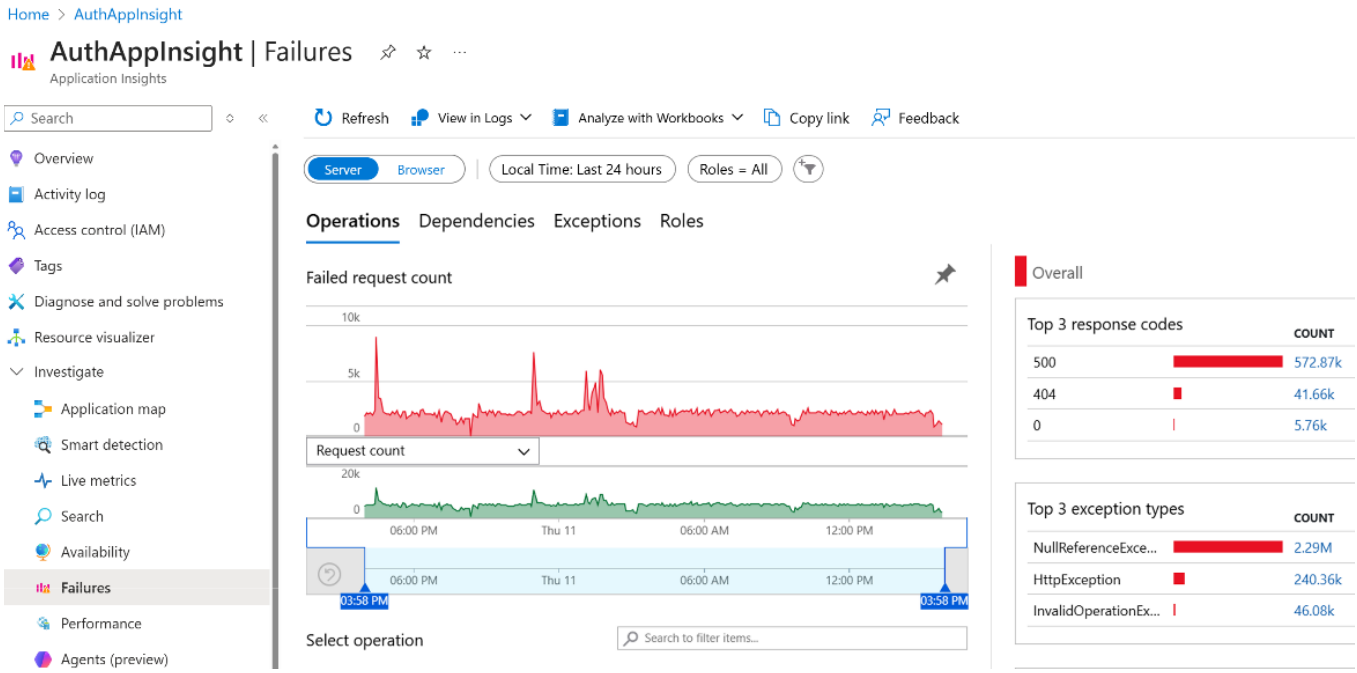


Рисунок 4.8 – Вікно Failures в Application Insights для ресурсу AuthAppInsight, що показує наявність помилкових запитів та дає змогу деталізувати збої за операціями й типами винятків

У системі «CoffeeShop» це дає змогу швидко виявляти «вузькі місця» в ланцюжку обробки замовлення, визначати, на якому саме мікросервісі виникають затримки або помилки, а також аналізувати вплив змін у коді чи конфігурації на загальний час відгуку користувацьких операцій. Додатково Application Insights може використовуватися для налаштування адаптивних оповіщень, що реагують не лише на абсолютні пороги метрик, а й на відхилення від звичного базового рівня, що підвищує чутливість системи моніторингу до нетипової поведінки.

Висновки з розділу 4

У четвертому розділі було розглянуто комплексний підхід до моніторингу, контролю відповідності та виявлення загроз для мікросервісної системи «CoffeeShop» у хмарі Microsoft Azure . Показано, що використання Azure Policy для Kubernetes дозволяє на рівні governance забезпечити дотримання обов'язкових вимог безпеки до кластерів AKS та робочих навантажень, автоматично блокуючи або аудіючи некоректні конфігурації, зокрема запуск привілейованих контейнерів, відсутність ресурсних лімітів та використання недозволених образів.

Окрему увагу приділено засобам runtime-безпеки: Microsoft Defender for Containers і Microsoft Defender for Cloud забезпечують сканування контейнерних образів та кластерів AKS на наявність вразливостей, моніторинг подій безпеки під час виконання та формування рекомендацій щодо покращення безпекової позиції, які інтегруються у процеси DevSecOps команди. Доповненням до цього є побудова системи спостережуваності на базі Azure Monitor Container Insights та Application Insights, що забезпечує збір метрик, логів і трас розподілених транзакцій, дозволяючи своєчасно виявляти проблеми з продуктивністю й надійністю та корелювати їх із потенційними інцидентами безпеки. Сукупність цих механізмів формує цілісну систему контролю й моніторингу, яка підтримує концепцію Zero Trust та забезпечує необхідний рівень прозорості й захищеності роботи мікросервісної cloud-native системи.

ВИСНОВКИ

У роботі сформульовано мету – розроблення та обґрунтування еталонної cloud-native архітектури мікросервісної інформаційної системи на базі екосистеми Microsoft Azure із застосуванням кращих практик безпеки для розгортання в середовищі Kubernetes. Для досягнення цієї мети було поставлено низку взаємопов’язаних завдань, результати виконання яких підсумовано нижче.

Перше завдання полягало в аналізі сучасних підходів до побудови та захисту мікросервісних cloud-native систем і в обґрунтуванні вибору технологічного стеку для реалізації демонстраційного застосунку. Це завдання виконано в розділі 1, де розглянуто еволюцію архітектур вебдодатків від моноліту до мікросервісів, проаналізовано переваги та ризики таких систем, обґрунтовано використання Kubernetes як оркестратора, Azure Kubernetes Service як керованого сервісу, Helm як засобу стандартизації розгортання, а також стеку безпеки Microsoft Azure (Zero Trust, Microsoft Defender for Cloud, Azure Policy). На основі аналізу сформульовано вимоги до демонстраційної системи «CoffeeShop» та окреслено її функціональні й нефункціональні характеристики.

Друге завдання стосувалося проєктування захищеної інфраструктури для цільової системи в хмарній платформі Microsoft Azure. У розділі 2 виконано це завдання шляхом розроблення мережевої топології з використанням Azure Application Gateway і Web Application Firewall для захисту периметра, а також мікросегментації внутрішнього трафіку в кластері AKS на основі Azure CNI та мережевих політик (Cilium). Спроєктовано модель зберігання даних із використанням Azure SQL Database для структурованих даних та Azure Blob Storage для мультимедійних об’єктів, а також організовано асинхронну комунікацію між мікросервісами через Azure Service Bus. Окремо розроблено модель управління ідентичністю та секретами, яка включає Microsoft Entra Workload Identity/Managed Identity, Azure Key Vault, Azure Container Registry та рольову модель доступу на основі Azure RBAC і Kubernetes RBAC.

Третє завдання передбачало розроблення методології безпечного розгортання (DevSecOps) для мікросервісної системи «CoffeeShop» на базі інструментів Azure DevOps. У розділі 3 це завдання реалізовано через організацію репозиторію в Azure

Repos із гілковою стратегією, політиками гілок і обов’язковим код-рев’ю, проєктування уніфікованих Helm-чартів для мікросервісів ASP.NET Core 8 та побудову CI/CD-конвеєра в Azure DevOps Pipelines. Розроблений конвеєр включає етап CI (збірка Docker-образів і публікація в Azure Container Registry), етап Security (інтеграція засобів аналізу безпеки на основі Microsoft Defender for DevOps для коду, інфраструктурних конфігурацій та контейнерних образів) і етап CD (автоматизоване розгортання в кластер AKS за допомогою Helm із використанням безпечних механізмів автентифікації). Це забезпечило відтворюваний DevSecOps-процес, у якому кожна зміна проходить через послідовні стадії перевірки якості й безпеки.

Четверте завдання було спрямоване на побудову комплексної системи моніторингу та виявлення загроз для розгорнутої cloud-native архітектури. У розділі 4 показано, як за допомогою Azure Policy реалізується рівень governance і контроль відповідності конфігурацій кластера AKS та робочих навантажень обов’язковим вимогам безпеки. Розглянуто використання Microsoft Defender for Containers та Microsoft Defender for Cloud для аналізу вразливостей контейнерних образів, моніторингу подій безпеки під час виконання та отримання рекомендацій щодо покращення безпекової позиції. Крім того, побудовано систему спостережуваності на базі Azure Monitor Container Insights і Application Insights, яка забезпечує збір метрик, логів і трас розподілених транзакцій, а також налаштування дашбордів і оповіщень для оперативного виявлення проблем із продуктивністю, надійністю та потенційних інцидентів безпеки.

Таким чином, усі поставлені в роботі завдання виконано, а сформована еталонна архітектура мікросервісної systemи «CoffeeShop» демонструє практичне застосування сучасних підходів до побудови, розгортання та захисту cloud-native інформаційних систем на базі екосистеми Microsoft Azure. Отримані результати можуть бути використані як методичний і практичний орієнтир для впровадження подібних рішень у промислових проєктах, а також слугувати основою для подальших досліджень у напрямі автоматизації безпеки й спостережуваності в контейнеризованих середовищах.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Richardson C. Microservices patterns: With examples in Java. Manning Publications, 2018. URL: https://microservices.io/patterns/monolithic.html (дата звернення: 16.12.2025).
2. Newman S. Building Microservices: Designing Fine-Grained Systems. O'Reilly Media, 2021. URL: https://martinfowler.com/articles/microservices.html (дата звернення: 16.12.2025).
3. Microsoft. N-tier architecture style. URL: https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/n-tier (дата звернення: 16.12.2025).
4. Microsoft. Choose an application architecture. URL: https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/architectural-principles (дата звернення: 16.12.2025).
5. Newman S. Building Microservices: Designing Fine-Grained Systems. O'Reilly Media, 2021. URL: https://martinfowler.com/articles/microservices.html (дата звернення: 16.12.2025).
6. Cloud Native Computing Foundation. Cloud Native Definition. URL: https://github.com/cncf/toc/blob/main/DEFINITION.md (дата звернення: 16.12.2025).
7. Erl T. SOA: Principles of Service Design. Prentice Hall, 2007. URL: https://www.oracle.com/technical-resources/articles/javase/soa-design-principles.html (дата звернення: 16.12.2025).
8. Richardson C. Microservices patterns: With examples in Java. Manning Publications, 2018. URL: https://microservices.io/patterns/microservices.html (дата звернення: 16.12.2025).
9. Merkel D. Docker: Lightweight Linux Containers for Consistent Development and Deployment. Linux Journal, 2014.

URL: https://www.docker.com/resources/what-container (дата звернення: 16.12.2025).

10. The Linux Foundation. Kubernetes Documentation: Concepts. URL: https://kubernetes.io/docs/concepts/ (дата звернення: 16.12.2025).
11. Cloud Native Computing Foundation. Cloud Native Definition. URL: https://github.com/cncf/toc/blob/main/DEFINITION.md (дата звернення: 16.12.2025).
12. Microsoft. Introduction to cloud-native applications on Azure. URL: https://learn.microsoft.com/en-us/dotnet/architecture/cloud-native/ (дата звернення: 16.12.2025).
13. NIST. NIST SP 800-204A: Building Secure Microservices-based Applications Using Service-Mesh Architecture. URL: https://csrc.nist.gov/publications/detail/sp/800-204a/final (дата звернення: 16.12.2025).
14. AWS. Security in microservices. URL: https://docs.aws.amazon.com/whitepapers/latest/microservices-on-aws/security-in-microservices.html (дата звернення: 16.12.2025).
15. Microsoft. What is Azure Kubernetes Service (AKS)? URL: https://learn.microsoft.com/en-us/azure/aks/what-is-aks (дата звернення: 16.12.2025).
16. Microsoft. Compare Azure Kubernetes Service (AKS) and self-managed Kubernetes. URL: https://learn.microsoft.com/en-us/azure/aks/compare-aks-vs-self-managed (дата звернення: 16.12.2025).
17. The Linux Foundation. Kubernetes Production Best Practices. URL: https://kubernetes.io/docs/setup/production-environment/ (дата звернення: 16.12.2025).
18. Microsoft. Kubernetes core concepts for Azure Kubernetes Service (AKS). URL: https://learn.microsoft.com/en-us/azure/aks/concepts-clusters-workloads#cluster-and-workload-management (дата звернення: 16.12.2025).

19. Microsoft. What is Azure Kubernetes Service (AKS)? URL: https://learn.microsoft.com/en-us/azure/aks/what-is-aks (дата звернення: 16.12.2025).

20. Microsoft. Azure Kubernetes Service (AKS) documentation – Security, authentication, and RBAC. URL: https://learn.microsoft.com/en-us/azure/aks/concepts-identity (дата звернення: 16.12.2025).

21. Microsoft. Azure Kubernetes Service (AKS) pricing. URL: https://azure.microsoft.com/en-us/pricing/details/kubernetes-service/ (дата звернення: 16.12.2025).

22. Microsoft. Best practices for cluster operators and developers on AKS. URL: https://learn.microsoft.com/en-us/azure/aks/operator-best-practices-cluster-security (дата звернення: 16.12.2025).

23. Helm – The package manager for Kubernetes. Official documentation. URL: https://helm.sh/docs/intro/using_helm/ (дата звернення: 16.12.2025).

24. Microsoft. Use Helm to deploy applications to Azure Kubernetes Service (AKS). URL: https://learn.microsoft.com/en-us/azure/aks/quickstart-helm (дата звернення: 16.12.2025).

25. Helm Best Practices. CNCF. URL: https://helm.sh/docs/topics/chart_best_practices/ (дата звернення: 16.12.2025).

26. Microsoft. Azure Container Registry as a Helm repository. URL: https://learn.microsoft.com/en-us/azure/container-registry/container-registry-helm-repos (дата звернення: 16.12.2025).

27. Microsoft. Store Helm charts using OCI artifacts in Azure Container Registry. URL: https://learn.microsoft.com/en-us/azure/container-registry/container-registry-helm-oci (дата звернення: 16.12.2025).

28. Microsoft. Use Helm to deploy applications to Azure Kubernetes Service (AKS). URL: https://learn.microsoft.com/en-us/azure/aks/quickstart-helm (дата звернення: 16.12.2025).

29. CNCF. Helm Security Best Practices. URL: https://helm.sh/docs/topics/security/ (дата звернення: 16.12.2025).

30. Helm – Rollback a release. Official documentation. URL: https://helm.sh/docs/helm/helm_rollback/ (дата звернення: 16.12.2025).

31. Artifact Hub – Find, install and publish Kubernetes packages. URL: https://artifacthub.io/ (дата звернення: 16.12.2025).

32. Microsoft. What is Zero Trust? URL: https://learn.microsoft.com/en-us/security/zero-trust/zero-trust-overview (дата звернення: 16.12.2025).

33. Microsoft. Zero Trust security in Azure. URL: https://learn.microsoft.com/en-us/security/zero-trust/zero-trust-azure (дата звернення: 16.12.2025).

34. Microsoft. Zero Trust for cloud-native applications. URL: https://learn.microsoft.com/en-us/security/zero-trust/zero-trust-deployment-cloudapps (дата звернення: 16.12.2025).

35. Microsoft. Secure containerized applications using Zero Trust principles. URL: https://learn.microsoft.com/en-us/security/zero-trust/deploy/deploy-containers (дата звернення: 16.12.2025).

36. Microsoft. What is Microsoft Defender for Cloud? URL: https://learn.microsoft.com/en-us/azure/defender-for-cloud/defender-for-cloud-introduction (дата звернення: 16.12.2025).

37. Microsoft. Defender for Cloud security posture management. URL: https://learn.microsoft.com/en-us/azure/defender-for-cloud/secure-score-security-controls (дата звернення: 16.12.2025).

38. Microsoft. Defender for Cloud workload protection for containers. URL: https://learn.microsoft.com/en-us/azure/defender-for-cloud/concept-container-security (дата звернення: 16.12.2025).

39. Microsoft. Azure Policy overview. URL: https://learn.microsoft.com/en-us/azure/governance/policy/overview (дата звернення: 16.12.2025).

40. Microsoft. Understand Azure Policy for Kubernetes clusters. URL: https://learn.microsoft.com/en-us/azure/governance/policy/concepts/policy-for-kubernetes (дата звернення: 16.12.2025).

41. OWASP. OWASP Top Ten Web Application Security Risks. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 16.12.2025).

42. Microsoft. What is Azure Application Gateway? URL: https://learn.microsoft.com/en-us/azure/application-gateway/overview (дата звернення: 16.12.2025).

43. Microsoft. Web application firewall on Azure Application Gateway. URL: https://learn.microsoft.com/en-us/azure/web-application-firewall/ag/ag-overview (дата звернення: 16.12.2025).

44. Microsoft. Configure end to end SSL with Application Gateway. URL: https://learn.microsoft.com/en-us/azure/application-gateway/ssl-overview (дата звернення: 16.12.2025).

45. Microsoft. Store TLS/SSL certificates in Azure Key Vault. URL: https://learn.microsoft.com/en-us/azure/application-gateway/key-vault-certs (дата звернення: 16.12.2025).

46. Microsoft. Autoscaling and zone redundancy in Application Gateway v2. URL: https://learn.microsoft.com/en-us/azure/application-gateway/application-gateway-autoscaling-zone-redundant (дата звернення: 16.12.2025).

47. Microsoft. Configure Azure CNI networking in AKS. URL: https://learn.microsoft.com/en-us/azure/aks/configure-azure-cni (дата звернення: 16.12.2025).

ДОДАТКИ

Додаток А

Порівняння AKS та self-hosted Kubernetes

Характеристика	AKS (керований сервіс)	Self-hosted Kubernetes
Розгортання control plane	Надано й обслуговується Azure	Повна відповідальність команди
Оновлення та патчі	Автоматизовані/керовані через AKS	Ручне планування та виконання
Інтеграція з Entra ID, ACR, Monitor, Key Vault	Вбудована підтримка, мінімальна конфігурація	Потрібна окрема інтеграція й налаштування
Витрати на експлуатацію	Зменшені операційні витрати	Вищі витрати на адміністрування
Фокус команди	На розробці та робочих навантаженнях	На підтримці інфраструктури кластера

Додаток Б

Переваги використання Helm для розгортання додатків у Kubernetes

Аспект	Без Helm	З використанням Helm
Опис ресурсів	Окремі YAML-маніфести	Структурований chart з шаблонами
Повторне використання	Обмежене, копіювання файлів	Параметризовані значення, спільні шаблони
Управління версіями	Ручне відстеження	Вбудована історія релізів і rollback
Узгодженість середовищ	Високий ризик відмінностей	Єдині chart’и для dev/test/prod
Інтеграція з CI/CD та ACR	Скрипти під кожен застосунок, окремі реєстри	Типові кроки helm install/upgrade із приватного реєстру ACR

Додаток В

Порівняння синхронної та асинхронної взаємодії в системі «CoffeeShop»

Аспект	**Синхронна взаємодія (прямий HTTP-запит)**	**Асинхронна взаємодія через Azure Service Bus**
Зв’язність компонентів	Сильна: клієнт очікує негайної відповіді від бекенду	Слабка: клієнт лише публікує подію, обробка виконується окремо
Стійкість до пікових навантажень	Низька: перевантаження бекенду → зростання затримок	Вища: черги згладжують сплески навантаження
Обробка довготривалих операцій	Ускладнена, ризик тайм-аутів	Зручна: операції виконуються у фоні worker-сервісами
Ризики повторної обробки	Контролюються на рівні транзакцій БД	Керуються через idempotency + dead-letter черги
Вимоги до доступності окремих сервісів	Критичні: відмова бекенду → помилка запиту клієнта	Гнучкіші: тимчасова недоступність worker’а не блокує прийом замовлень

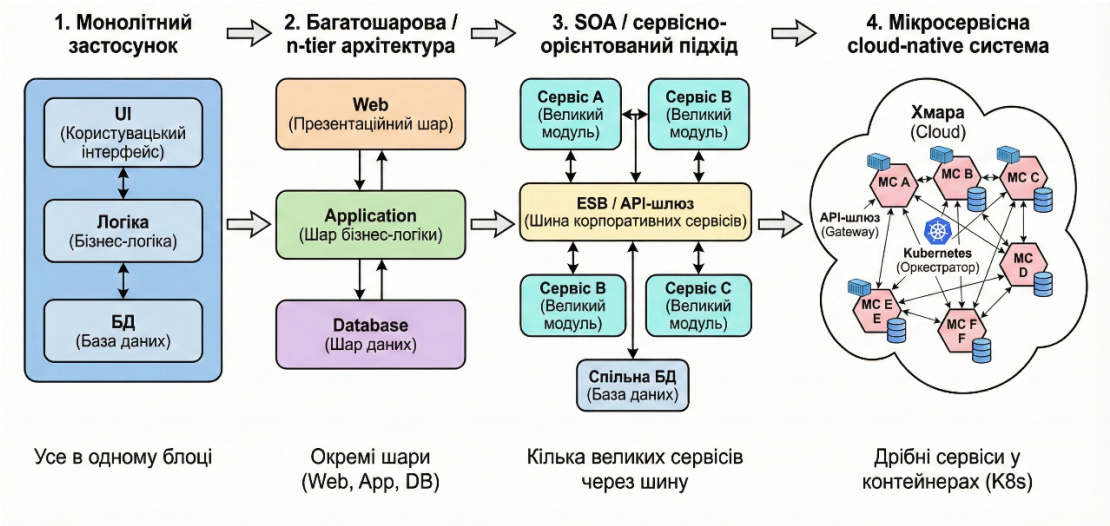
Додаток Г

Приклади ролей та прав доступу в системі «CoffeeShop»

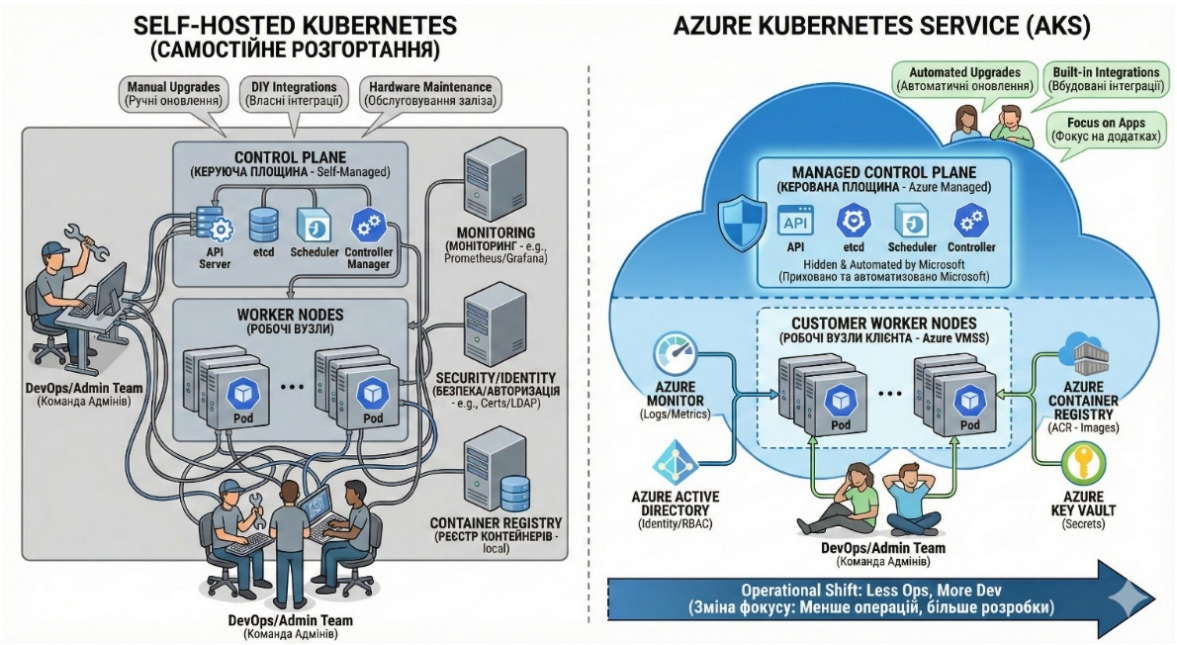
Роль / суб’єкт	**Рівень доступу в Azure**	**Рівень доступу в AKS/RBAC**
Адміністратор платформи	Повний доступ до ресурсної групи з AKS, ACR, Key Vault	Кластерна роль з можливістю керування кластерами, namespace та політиками
DevOps / SRE	Керування AKS, ACR, Key Vault, Service Bus у межах RG	Розгортання релізів, перегляд логів, оновлення конфігурацій в prod/dev
Розробник	Обмежений доступ до dev-ресурсів (AKS dev, ACR dev)	Доступ до namespace dev, створення/оновлення Deployment, перегляд логів
CI/CD конвеєр	Доступ до ACR, AKS, іноді Key Vault для секретів деплою	Сервісний account з правами на helm install/upgrade у визначених namespace
Сервісні ідентичності	Доступ лише до конкретних БД, сховищ, Service Bus, KV	Лише використання секретів/конфігів через Workload Identity, без адмін-прав

Додаток Д

Еволюція архітектури вебдодатків від моноліту до мікросервісних cloud-native систем

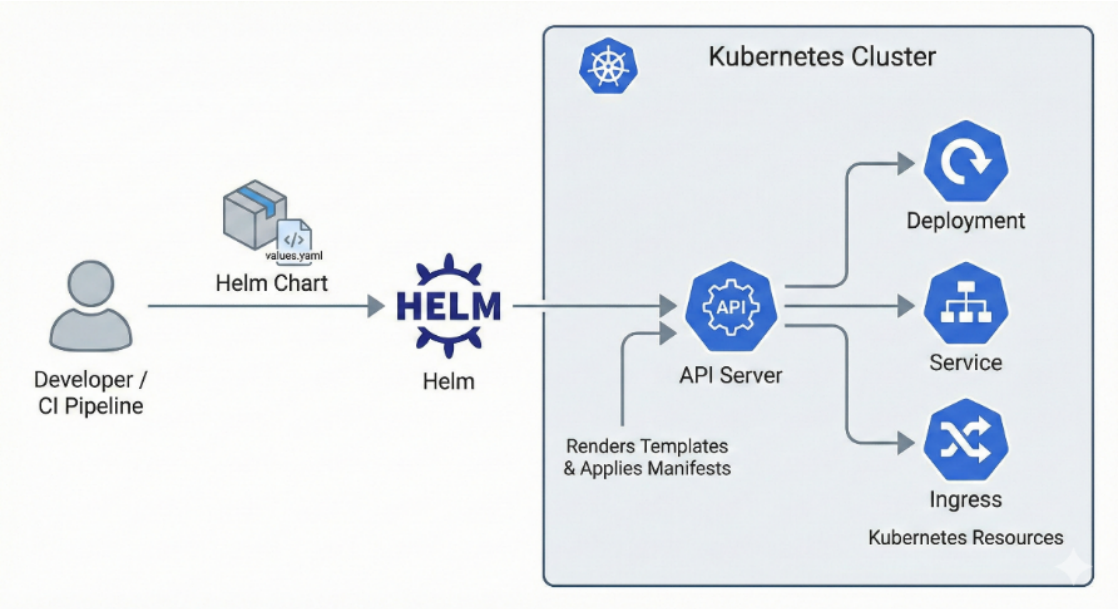


Порівняння архітектури self-hosted кластера Kubernetes та керованого сервісу Azure Kubernetes Service

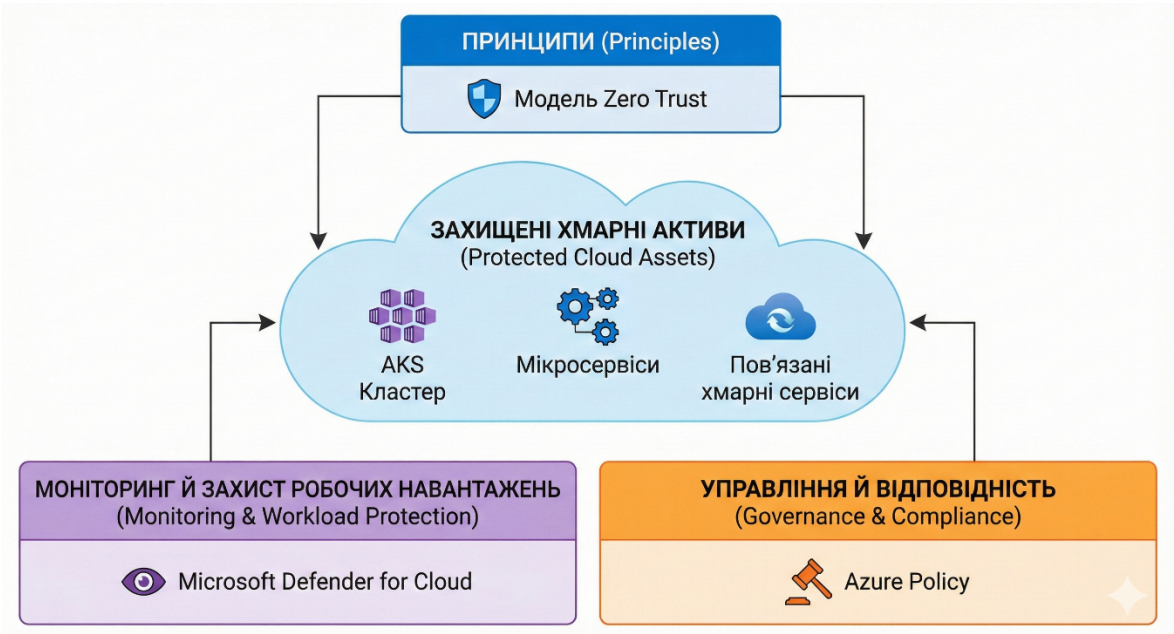


Додаток Е

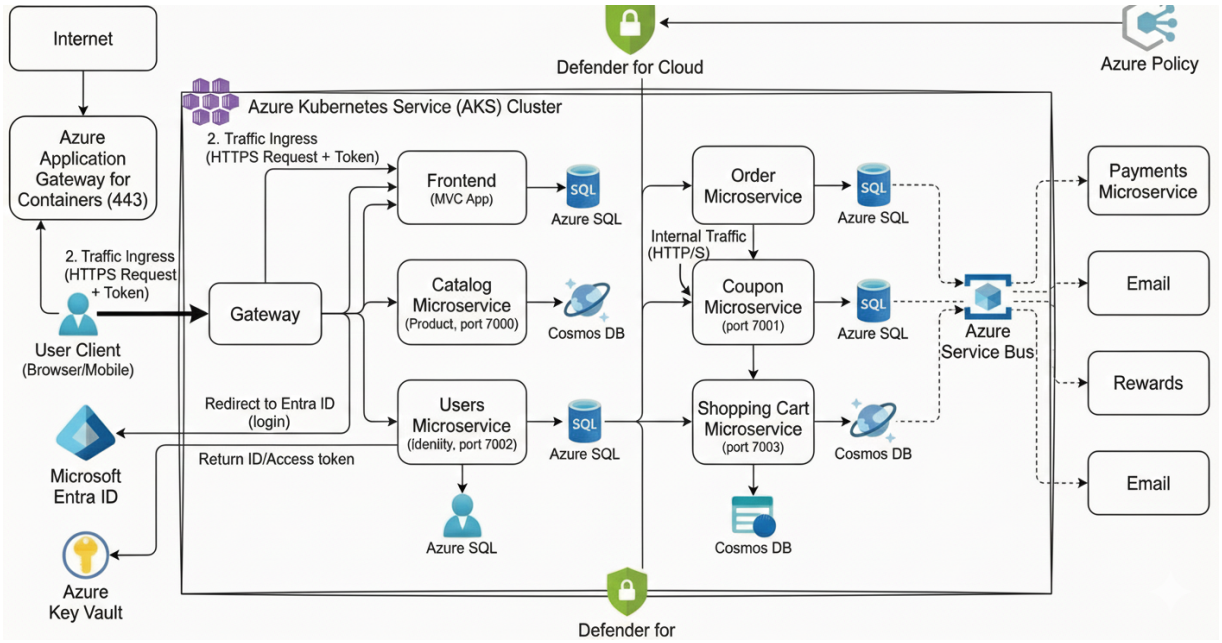
Схема процесу розгортання мікросервісного застосунку в Kubernetes за допомогою пакетного менеджера Helm



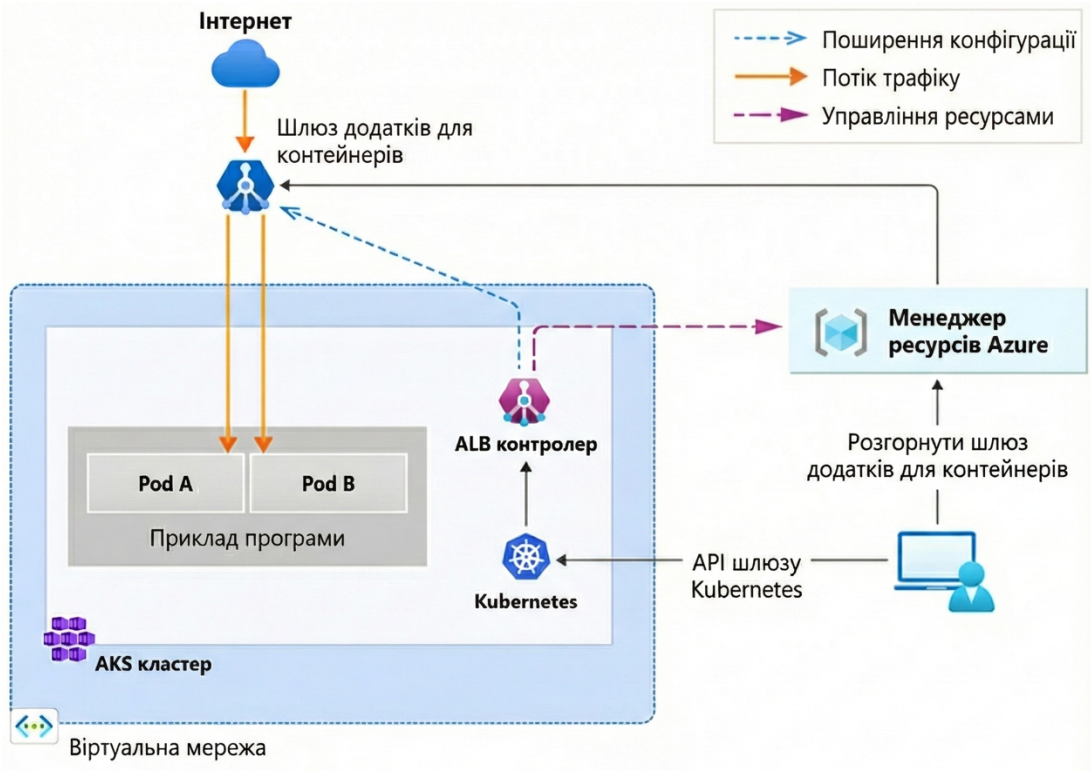
Узагальнена схема стеку безпеки Microsoft Azure для мікросервісної cloud-native системи, що поєднує принципи Zero Trust, Microsoft Defender for Cloud та Azure Policy



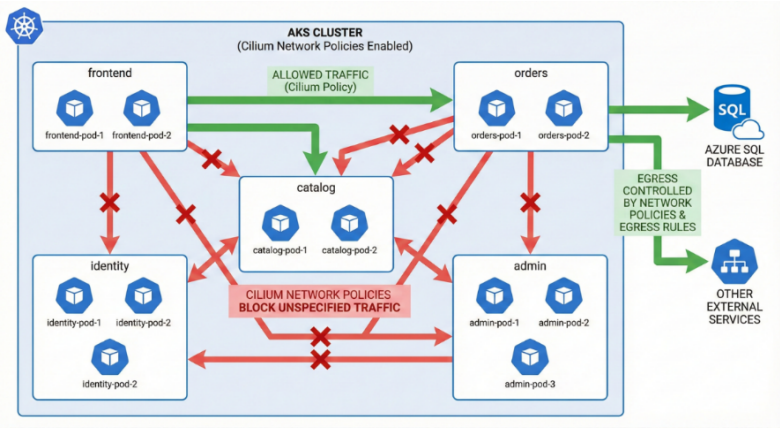
Високорівнева архітектура системи «CoffeeShop» у середовищі Azure Kubernetes Service



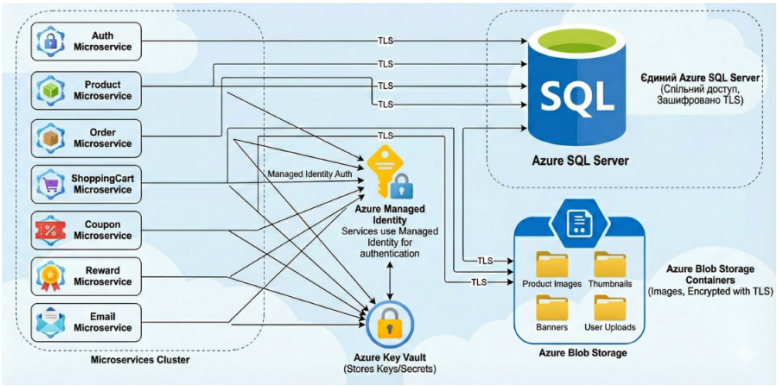
Використання Azure Application Gateway for Containers для публікації сервісів кластера AKS



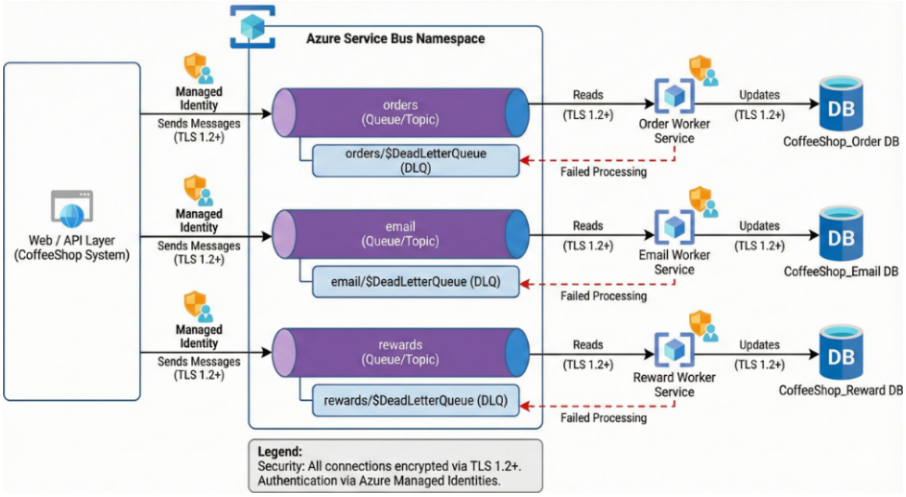
Внутрішня мережна сегментація кластера Azure Kubernetes Service для системи «CoffeeShop» із використанням Azure CNI та мережевих політик Cilium



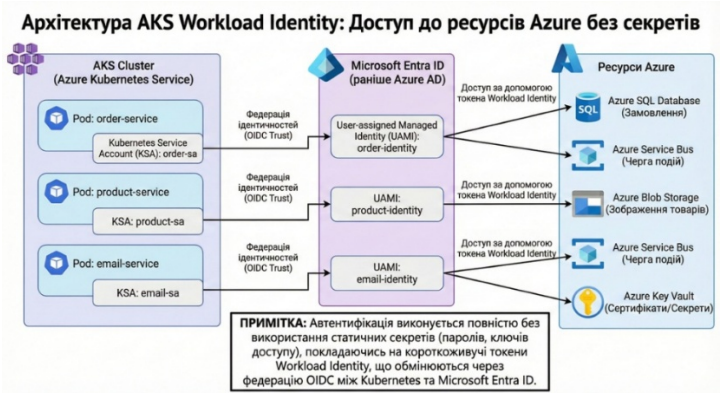
Логічна схема зберігання даних системи «CoffeeShop» на базі Azure SQL Database та Azure Blob Storage



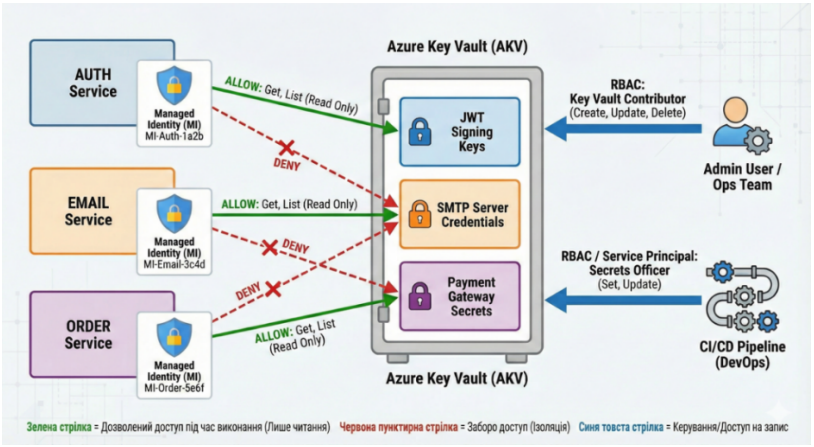
Організація асинхронної обробки подій у системі «CoffeeShop» за допомогою Azure Service Bus queues/topics та робітників-споживачів



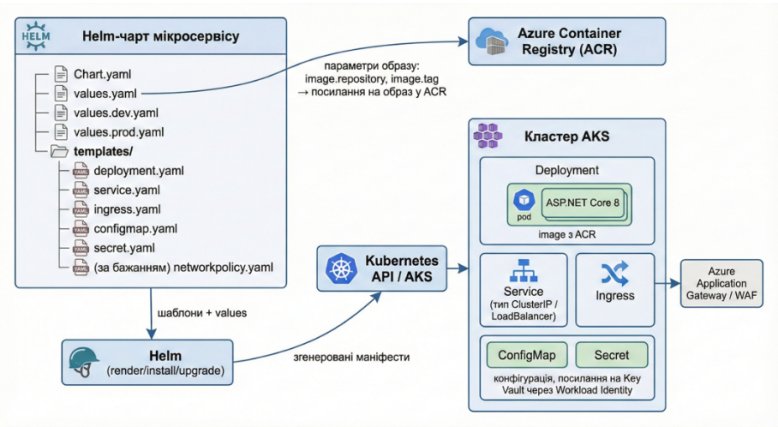
Використання Microsoft Entra Workload Identity для доступу мікросервісів системи «CoffeeShop» до ресурсів Azure без статичних секретів



Взаємодія керованих ідентичностей мікросервісів системи «CoffeeShop» з Azure Key Vault при доступі до критичних секретів



Структура уніфікованого Helm-чарту та схема розгортання мікросервісу ASP.NET Core 8 системи «CoffeeShop» у кластері Azure Kubernetes Service із використанням Azure Container Registry та параметризованих файлів значень



Додаток К

Узагальнена схема DevSecOps-конвеєра для мікросервісної системи «CoffeeShop» в Azure DevOps Pipelines

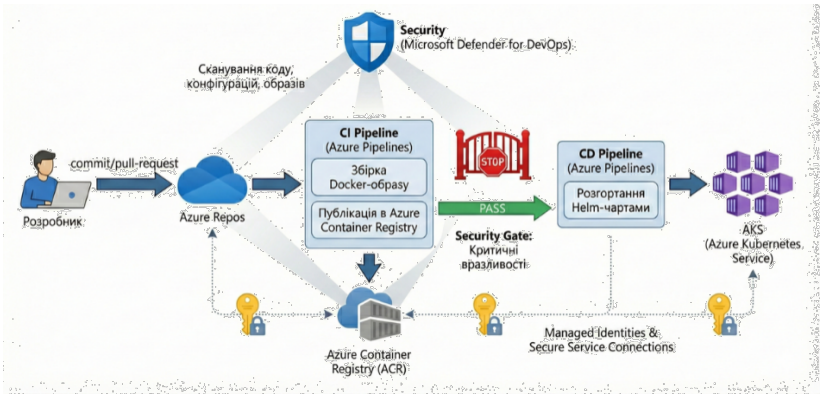


Схема застосування ініціатив Azure Policy (наприклад, для стандарту ISO 27001) та компонента Azure Policy for Kubernetes (Gatekeeper) для контролю конфігурації кластера AKS і робочих навантажень системи «CoffeeShop» на рівні ефектів Audit, Deny та DeployIfNotExists/Modify

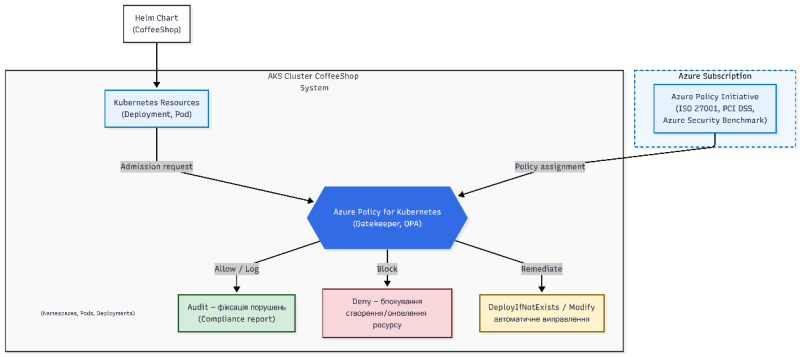
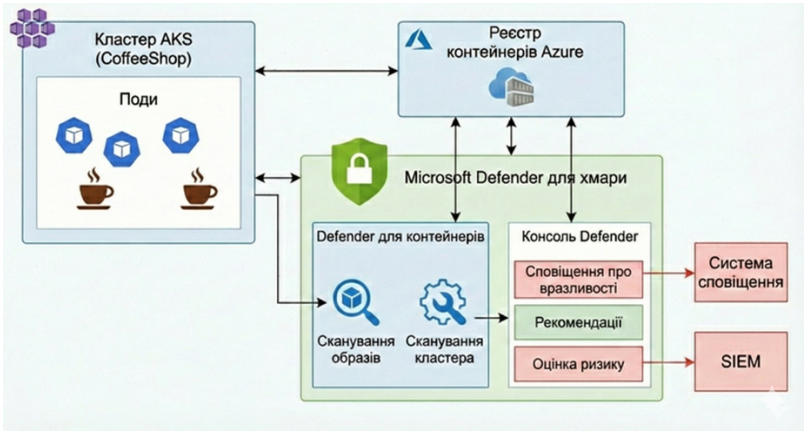


Схема моніторингу безпеки контейнерних навантажень системи «CoffeeShop» за допомогою Microsoft Defender for Containers та Microsoft Defender for Cloud



YAML-конфігурації багатостадійного CI/CD-пайплайна в Azure Pipelines

```
trigger:
  branches:
    include:
      - main
pool:
  vmImage: 'ubuntu-latest'
variables:
  acrName: 'coffeeshopacrstud'
  imageName: 'coffeeshop-authapi'
  imageTag: '$(Build.BuildId)'
  aksResourceGroup: 'CoffeeShop'
  aksClusterName: 'aks-coffeeshop'
stages:
- stage: Build
  displayName: 'Build .NET solution'
  jobs:
  - job: Build
    steps:
    - task: UseDotNet@2
      inputs:
        packageType: 'sdk'
        version: '8.x'
    - script: |
        dotnet restore CoffeeShop/CoffeeShop.sln
        dotnet build CoffeeShop/CoffeeShop.sln -c Release --no-restore
      displayName: 'dotnet restore & build'
- stage: Docker
  displayName: 'Build and push Docker image'
  dependsOn: Build
  jobs:
  - job: Docker
    steps:
    - task: Docker@2
      displayName: 'Build and push image'
      inputs:
        command: 'buildAndPush'
        repository: '$(imageName)'
        dockerfile: 'CoffeeShop/Dockerfile'
        containerRegistry: '$(acrServiceConnection)'
```

```
        tags: |
          $(imageTag)
- stage: Deploy
  displayName: 'Deploy to AKS with Helm'
  dependsOn: Docker
  jobs:
  - job: Deploy
    steps:
    - task: AzureCLI@2
      displayName: 'Deploy Helm release'
      inputs:
        azureSubscription: '$(aksServiceConnection)'
        scriptType: 'bash'
        scriptLocation: 'inlineScript'
        inlineScript: |
          az aks get-credentials -g $(aksResourceGroup) -n $(aksClusterName) --overwrite-existing
          helm upgrade --install authapi ./CoffeeShop/deploy/authapi \
            --set image.repository=$(acrName).azurecr.io/$(imageName) \
            --set image.tag=$(imageTag)
```