

Міністерство освіти і науки України  
Харківський національний університет радіоелектроніки

Факультет  
Кафедра

Комп'ютерної інженерії та управління  
Комп'ютерних інтелектуальних технологій та систем

## КВАЛІФІКАЦІЙНА РОБОТА Пояснювальна записка

рівень вищої освіти

другий (магістерський)

Інтеграція мультитенантних рішень у хмарні інформаційні системи  
інтелектуальних безперервних виробництв  
(тема)

Виконав:

здобувач II курсу, групи КІТм-23-1

Олександр ТРИПОЛЬСВ

(власне ім'я, прізвище)

Спеціальність 123 Комп'ютерна інженерія

(код і повна назва спеціальності)

Тип програми освітньо-професійна

(освітньо-професійна або освітньо-наукова)

Освітня програма Комп'ютерні  
інтелектуальні технології

(повна назва освітньої програми)

Керівник доц. каф. КІТС Наталія СЕРДЮК

(посада, власне ім'я, прізвище)

Допускається до захисту

Зав. кафедри

Олег РУДЕНКО

(підпис)

(власне ім'я, прізвище)

2024 р.

## Харківський національний університет радіоелектроніки

Факультет	Комп'ютерної інженерії та управління
Кафедра	Комп'ютерних інтелектуальних технологій та систем
Рівень вищої освіти	другий (магістерський)
Спеціальність	123 Комп'ютерна інженерія
Тип програми	освітньо-професійна
Освітня програма	Комп'ютерні інтелектуальні технології

ЗАТВЕРДЖУЮ:

Зав. кафедри \_\_\_\_\_

(підпис)

«\_\_\_\_\_» \_\_\_\_\_ 2024р.

### ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

- здобувачеві Трипольєву Олександровичу
1. Тема роботи (проекту) Інтеграція мультитенантних рішень у хмарні інформаційні системи інтелектуальних безперервних виробництв  
затверджена наказом університету від "28" жовтня 2024 р. № 1256 Ст \_\_\_\_\_
  2. Термін подання учнем роботи до екзаменаційної комісії 20.01.2024р.
  3. Вихідні дані до роботи (проекту) \_\_\_\_\_
    - 1) ознаки мультитенантної архітектури;
    - 2) використання механізмів Deployment, Service, Horizontal Pod Autoscaler для динамічного масштабування додатків;
    - 3) хмарні платформи AWS, Azure, GCP
  4. Перелік питань, що потрібно опрацювати в роботі:
    - Огляд стану проблеми та постановка задачі
    - Аналіз літератури за напрямком дослідження
    - Огляд сучасних методів та технологій для побудови мультитенантних рішень
    - Визначення ключових переваг і викликів мультитенантності в контексті інтелектуальних виробництв
    - Розробка компонентів архітектури хмарної інформаційної системи з підтримкою мультитенантності
    - Впровадження запропонованої архітектури в реальному середовищі
    - Підготовка презентаційного матеріалу

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням кафедри)  
16 слайдів презентаційного матеріалу

6. Консультанти розділів роботи (п.6 включається до завдання за наявністю консультантів згідно до наказу, зазначеному у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

### КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Видача та узгодження теми проєкту	28.10.2024	Виконано
2	Огляд стану проблеми та постановка задачі	30.11-29.11.2024	Виконано
3	Аналіз літератури за напрямком дослідження	30.11- 07.12.2024	Виконано
4	Аналіз існуючих підходів до реалізації мультитенантної архітектури в хмарних системах	08.12-10.12.2024	Виконано
5	Розробка компонентів архітектури	11.12-16.12.2024	Виконано
6	Впровадження запропонованої архітектури	17.12-24.12.2024	Виконано
7	Тестування мультитенантної системи на відповідність вимогам інтелектуальних безперервних виробництв	25.12-29.12.2024	Виконано
8	Підготовка ПЗ та презентаційного матеріалу	30.12-23.01.2025	Виконано
9	Подання до ЕК	20.01.2025	Виконано
10	Захист проєкту	22.01.2025	

Дата видачі завдання «28» жовтня 2024 р.

Здобувач

\_\_\_\_\_ (підпис)

Керівник роботи

\_\_\_\_\_ (підпис)

доцент Н. М. Сердюк

(посада, ініціали, прізвище)

## РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 79 сторінок, 10 рисунків, 2 таблиці, 1 додаток, 13 джерел.

МУЛЬТИТЕНАНТНІСТЬ, ТЕНАНТИ, АРХІТЕКТУРА, ІНТЕГРАЦІЯ, ХМАРНІ СЕРВІСИ, ІНТЕЛЕКТУАЛЬНІ ВИРОБНИЧІ СИСТЕМИ, DOCKER, KUBERNETES, МАСШТАБУВАННЯ, АВТОМАТИЗАЦІЯ

Метою цієї кваліфікаційної роботи є розробка та впровадження мультитенантних рішень у хмарні інформаційні системи для підвищення продуктивності, гнучкості та ефективного управління інтелектуальними безперервними виробництвами.

Розробка хмарної інформаційної системи з підтримкою мультитенантності є складним і багатокомпонентним процесом, що потребує інтеграції сучасних технологій для забезпечення гнучкості, продуктивності та безпеки. Мультитенантна архітектура дозволяє ефективно використовувати ресурси, забезпечуючи одночасну роботу декількох клієнтів з ізоляцією їх даних та процесів. Це досягається завдяки використанню контейнеризації додатків за допомогою Docker, а також автоматизації та масштабування через Kubernetes.

Інтеграція з хмарними сервісами, такими як AWS, Azure та GCP, надає системі доступ до потужних обчислювальних ресурсів і сервісів зберігання, що дозволяє додаткам адаптуватися до зростаючих навантажень і вимог ринку. Окрім цього, хмарні сервіси надають інструменти для аналізу даних, автоматизації процесів та інтеграції з сервісами машинного навчання, що дозволяє побудувати інтелектуальні виробничі системи з високою продуктивністю.

## ABSTRACT

Explanatory note of the qualification work: 79 pages, 10 figures, 2 tables, 2 appendices, 13 sources.

MULTITENANCY, TENANTS, ARCHITECTURE, INTEGRATION, CLOUD SERVICES, INTELLIGENT PRODUCTION SYSTEMS, DOCKER, KUBERNETES, SCALING, AUTOMATION

The purpose of this qualification work is to develop and implement multitenant solutions in cloud information systems to enhance productivity, flexibility, and efficient management of intelligent continuous production systems.

The development of a cloud information system with multitenant support is a complex and multi-component process that requires the integration of modern technologies to ensure flexibility, performance, and security. Multitenant architecture allows efficient resource utilization by enabling the simultaneous operation of multiple clients while ensuring the isolation of their data and processes. This is achieved through the use of application containerization via Docker, as well as automation and scaling through Kubernetes.

Integration with cloud services such as AWS, Azure, and GCP provides the system with access to powerful computing resources and storage services, enabling applications to adapt to increasing workloads and market demands. Additionally, these cloud services offer tools for data analysis, process automation, and integration with machine learning services, facilitating the creation of high-performance intelligent production systems.

Міністерство освіти і науки України  
Харківський національний університет радіоелектроніки

Факультет  
Кафедра

Комп'ютерної інженерії та управління  
Комп'ютерних інтелектуальних технологій та систем

## **АНОТАЦІЯ**

### **КВАЛІФІКАЦІЙНОЇ РОБОТИ**

рівень вищої освіти

другий (магістерський)

Інтеграція мультитенантних рішень у хмарні інформаційні системи  
інтелектуальних безперервних виробництв  
(тема)

Виконав:

здобувач II курсу, групи КІТм-23-1

**Олександр ТРИПОЛЬСВ**

(власне ім'я, прізвище)

Спеціальність 123 Комп'ютерна інженерія

(код і повна назва спеціальності)

Тип програми освітньо-професійна

(освітньо-професійна або освітньо-наукова)

Освітня програма Комп'ютерні  
інтелектуальні технології

(повна назва освітньої програми)

Керівник доц. каф. КІТС Наталія СЕРДЮК

(посада, власне ім'я, прізвище)

2024

## АНОТАЦІЯ

Трипольєв О.В. Інтеграція мультитенантних рішень у хмарні інформаційні системи інтелектуальних безперервних виробництв. – Магістерська кваліфікаційна робота.

Інтелектуальні безперервні виробництва відкривають значні можливості для автоматизації та оптимізації процесів. Завдяки впровадженню технологій штучного інтелекту та машинного навчання, підприємства можуть прогнозувати несправності обладнання, оптимізувати використання ресурсів і знижувати енерговитрати. Ці технології також дозволяють автоматизувати рутинні операції, що звільняє людські ресурси для виконання більш складних і творчих завдань.

Хмарні обчислення та мультитенантність відіграють ключову роль у підвищенні ефективності інтелектуальних безперервних виробництв, забезпечуючи гнучкість, масштабованість та оптимізацію ресурсів. Хмарні обчислення дозволяють підприємствам інтегрувати передові технології, такі як аналітика великих даних, штучний інтелект і машинне навчання, без необхідності створювати та підтримувати власну обчислювальну інфраструктуру. Це значно знижує витрати на ІТ-інфраструктуру та обслуговування, оскільки всі ресурси доступні через Інтернет і можуть бути масштабовані відповідно до потреб виробництва.

Метою цієї кваліфікаційної роботи є розробка та впровадження мультитенантних рішень у хмарні інформаційні системи для підвищення продуктивності, гнучкості та ефективного управління інтелектуальними безперервними виробництвами.

У першому розділі був проведений аналіз предметної області та постановка задачі дослідження, а саме: сучасні виклики і можливості в

інтелектуальних безперервних виробництвах та роль хмарних обчислень та мультитенантності в підвищенні ефективності виробництв.

У другому розділі були визначені ключові компоненти хмарної системи з мультитенантністю: модулі управління тенантами, обробки даних, обчислювальний модуль, безпекові модулі та інтерфейс користувача. Кожен з цих модулів відповідає за окремі аспекти роботи системи, забезпечуючи високу ефективність, безпеку та ізоляцію даних кожного тенанта. Архітектура побудована на сучасних принципах контейнеризації та хмарних обчислень, що дозволяє масштабувати систему, підтримувати відмовостійкість і швидко реагувати на зміну вимог.

У третьому розділі був детально розглянутий процес контейнеризації додатків за допомогою Docker та налаштування Kubernetes для оркестрації контейнерів. Kubernetes дозволяє автоматично масштабувати систему, забезпечувати балансування навантаження і відновлення сервісів при збоях. Було розглянуто використання таких механізмів, як Deployment, Service, Horizontal Pod Autoscaler для динамічного масштабування додатків. Kubernetes також підтримує ізоляцію ресурсів для кожного тенанта за допомогою Namespace, що дозволяє зберігати їх дані й процеси окремо.

Для забезпечення масштабованої інфраструктури в четвертому розділі було розглянуто інтеграцію з основними хмарними платформами AWS, Azure, GCP. Це дозволяє додаткам використовувати керовані сервіси для зберігання даних, таких як Amazon S3, Azure Blob Storage та Google Cloud Storage. Також розглянули використання керованих баз даних, таких як Amazon RDS або Azure SQL Database, і можливість використання інструментів для аналізу даних і машинного навчання, таких як Google BigQuery або Azure Machine Learning. Також були розглянуті механізми безпеки, які використовуються для захисту даних у хмарних системах. Це включало налаштування шифрування даних (як при зберіганні, так і при передачі), використання Role-Based Access Control (RBAC) для управління правами доступу користувачів і подів, а також

використання інструментів управління доступом, таких як AWS IAM або Azure AD. Налаштування мережевої безпеки через Network Policies у Kubernetes забезпечує контроль за мережевим трафіком між подами. Безпека є важливим аспектом таких систем, і було розглянуто засоби забезпечення захисту даних у хмарі, включаючи шифрування, контроль доступу через IAM, RBAC та мережеві політики в Kubernetes. Завдяки цьому можна гарантувати захист даних навіть у складних мультитенантних середовищах, де різні клієнти використовують одну і ту ж інфраструктуру.

Таким чином, запропонована архітектура хмарної інформаційної системи з мультитенантністю є надійним, гнучким і безпечним рішенням для сучасних виробничих процесів, забезпечуючи масштабованість та ефективне управління ресурсами в умовах інтенсивних обчислювальних і даних навантажень.

МУЛЬТИТЕНАНТНІСТЬ, ТЕНАНТИ, АРХІТЕКТУРА, ІНТЕГРАЦІЯ, ХМАРНІ СЕРВІСИ, ІНТЕЛЕКТУАЛЬНІ ВИРОБНИЧІ СИСТЕМИ, DOCKER, KUBERNETES, МАСШТАБУВАННЯ, АВТОМАТИЗАЦІЯ

Публікації здобувача за темою роботи:

1. Сердюк Н.М., Вольгуст М.С. Трипольев О. Використання можливостей Cloud Computing при обробці інформації у безперервних виробництвах: тези доповідей XIV Всеукр. НПК – Дніпро, 21 березня 2024 р., стор 177.

2. Trypoliev O., Volhust M., Serdiuk N. Features of cloud computing in big data processing /Матеріали XVII Всеукр. НПWEBK – Кривий Ріг, 26-28 березня 2024 р. – 283 с. Стор 28.

3. Сердюк Н.М., Трипольев О.В. Особливості розробки архітектури хмарної інформаційної системи з підтримкою мультитенантності. Комп'ютерні ігри і мультимедіа як інноваційний підхід до комунікації: тези доповідей IV Всеукр. НПК – Одеса, 2024.

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AI – штучний інтелект

AWS – Amazon Web Services

CPS – кіберфізичні системи

GCP – Google Cloud Platform

IaC – інфраструктуру як код

IoT – Інтернет речей

IBC – Інтелектуальні виробничі системи

ML – машинне навчання

## ЗМІСТ

Вступ	13
1 Аналіз предметної області та постановка задачі дослідження	15
1.1 Опис сучасних викликів і можливостей в інтелектуальних безперервних виробництвах	15
1.2 Роль хмарних обчислень та мультитенантності в підвищенні ефективності виробництв	22
1.3 Постановка задачі дослідження	26
2 Аналіз існуючих підходів до реалізації мультитенантної архітектури в хмарних системах	27
2.1 Огляд сучасних методів та технологій для побудови мультитенантних рішень	27
2.2 Визначення ключових переваг і викликів мультитенантності в контексті інтелектуальних виробництв	28
2.3 Огляд особливостей інтелектуальних виробничих систем	30
2.4 Роль автоматизації та інтелектуальних алгоритмів у підвищенні ефективності виробництв	31
3 Розробка архітектури хмарної інформаційної системи з підтримкою мультитенантності	37
3.1 Основні компоненти архітектури хмарної інформаційної системи з підтримкою мультитенантності	37
3.2 Інтеграція інтелектуальних модулів у мультитенантну архітектуру	43
4 Впровадження запропонованої архітектури в реальному середовищі	46
4.1 Основні етапи впровадження та їх особливості	46

4.2 Створення та розгортання додатків у контейнерах Docker	48
4.3 Створення та налаштування Kubernetes-кластера	52
4.3.1 Розгортання додатків у Kubernetes	53
4.3.2 Налаштування масштабування додатків	55
4.4 Забезпечення відмовостійкості та автоматичне відновлення	55
4.5 Налаштування безпеки та ізоляції	56
4.6 Інтеграція контейнеризованих додатків з хмарними сервісами	57
4.7 Забезпечення безпеки та управління доступом	60
4.8 Методика тестування мультитенантної системи на відповідність вимогам інтелектуальних безперервних виробництв	63
4.9 Порівняльний аналіз хмарних платформ для інтеграції інформаційної системи з підтримкою мультитенантності	66
Висновки	68
Перелік використаних джерел	70
Додаток А. Слайди презентації	72

## ВСТУП

Мультитенантні хмарні інформаційні системи відіграють важливу роль у сучасних технологічних процесах, особливо в інтелектуальних безперервних виробництвах. Використання хмарних рішень дозволяє підприємствам значно оптимізувати ресурси, підвищити продуктивність та забезпечити гнучкість управління бізнес-процесами. У цьому контексті мультитенантність є ключовою технологією, що дозволяє одночасно обслуговувати кількох клієнтів (тенантів) за допомогою однієї інфраструктури, зберігаючи ізоляцію їх даних та процесів. Такий підхід забезпечує значну економію витрат на обчислювальні ресурси та спрощує управління системою, що особливо важливо для виробничих процесів з високим рівнем автоматизації.

Актуальність дослідження мультитенантних рішень для інтелектуальних виробництв визначається зростаючими вимогами до обробки великих обсягів даних і забезпечення безперервної роботи систем в умовах високих навантажень. Інтелектуальні виробничі процеси, що використовують штучний інтелект та машинне навчання, потребують надійних і гнучких платформ, здатних масштабуватися залежно від потреб ринку. Мультитенантні хмарні рішення дозволяють оптимізувати ці процеси за рахунок автоматизації управління ресурсами та інтеграції з сучасними інструментами для аналізу даних.

Попри активний розвиток хмарних технологій, мультитенантна архітектура залишається складною для впровадження у виробничих середовищах через ряд викликів, пов'язаних із забезпеченням безпеки даних, управлінням ресурсами та оптимізацією продуктивності. Багато підприємств стикаються з проблемами ізоляції даних різних клієнтів, що ускладнює забезпечення їх конфіденційності і захисту. Крім того, розподіл ресурсів між кількома клієнтами може впливати на загальну продуктивність системи,

особливо при різкому збільшенні навантаження. Відповідно, розробка ефективної архітектури, яка здатна вирішувати ці проблеми та одночасно забезпечувати гнучкість, є актуальним завданням для сучасних хмарних систем.

Метою даного дослідження є розробка та впровадження мультитенантної архітектури для хмарних систем, які використовуються в інтелектуальних виробництвах. Наукова частина дослідження спрямована на розробку моделі архітектури, що забезпечує ефективне управління ресурсами, ізоляцію даних клієнтів та підтримку масштабованості у хмарному середовищі. Практична частина дослідження полягає в інтеграції мультитенантної архітектури в інтелектуальні виробничі системи, що дозволить підприємствам підвищити продуктивність, знизити витрати на інфраструктуру та забезпечити безпеку даних у хмарному середовищі.

У результаті цього дослідження буде розроблена система, яка дозволить підприємствам ефективно впроваджувати хмарні рішення, що відповідають сучасним вимогам безперервних виробничих процесів. Це відкриє нові можливості для оптимізації ресурсів, автоматизації процесів і зростання конкурентоспроможності на ринку.

## 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

### 1.1 Опис сучасних викликів і можливостей в інтелектуальних безперервних виробництвах

Інтелектуальні безперервні виробництва стикаються з низкою сучасних викликів, які водночас відкривають нові можливості для підвищення ефективності та конкурентоспроможності підприємств.

Одним з основних викликів є необхідність інтеграції складних технологій, таких як штучний інтелект (AI), машинне навчання (ML) та Інтернет речей (IoT), у вже існуючі виробничі процеси. Ці технології вимагають значних інвестицій та технічної експертизи, що може бути проблематичним для підприємств з обмеженими ресурсами. Крім того, безперервні виробництва характеризуються високими вимогами до стабільності та надійності систем, оскільки навіть незначні простої можуть призвести до значних фінансових втрат. Інтеграція нових технологій може створювати додаткові ризики для безперервності операцій, що потребує ретельного планування та тестування.

Іншим викликом є управління та аналіз великих обсягів даних, які генеруються на кожному етапі виробничого процесу. Збір, зберігання та обробка цих даних потребують значних обчислювальних ресурсів і спеціалізованих інструментів для забезпечення своєчасної і точної аналітики. Це також вимагає високого рівня кібербезпеки, щоб захистити чутливі дані від потенційних загроз, включаючи кібератаки та несанкціонований доступ. Умови безперервних виробництв роблять такі загрози ще більш критичними, оскільки будь-який інцидент може вплинути на весь виробничий ланцюг.

Попри ці виклики, інтелектуальні безперервні виробництва відкривають значні можливості для автоматизації та оптимізації процесів. Завдяки впровадженню технологій штучного інтелекту та машинного навчання, підприємства можуть прогнозувати несправності обладнання, оптимізувати використання ресурсів і знижувати енерговитрати. Ці технології також дозволяють автоматизувати рутинні операції, що звільняє людські ресурси для виконання більш складних і творчих завдань.

Інша значуща можливість полягає в інтеграції хмарних обчислень, що забезпечує високу масштабованість і гнучкість. Хмарні платформи дозволяють підприємствам використовувати передові аналітичні інструменти, не вкладаючи значні кошти в інфраструктуру. Вони також сприяють спрощенню управління виробничими процесами, оскільки централізоване управління і моніторинг можна здійснювати віддалено і в реальному часі. Це особливо важливо для глобальних компаній, які мають розподілені виробничі підрозділи в різних регіонах. Також, впровадження Інтернету речей (IoT) у виробництво відкриває нові перспективи для підвищення точності і якості продукції. IoT-сенсори можуть безперервно збирати дані про стан обладнання, умови навколишнього середовища і продукцію, що дозволяє оперативно реагувати на будь-які відхилення та покращувати якість продукції. Це також сприяє зниженню витрат на обслуговування і ремонт, оскільки можливості прогнозувальної аналітики дозволяють планувати технічне обслуговування на основі реальних потреб, а не за розкладом.

Загалом, інтелектуальні безперервні виробництва мають величезний потенціал для підвищення продуктивності, якості та гнучкості виробничих процесів. Однак, для реалізації цього потенціалу необхідно вирішити низку викликів, пов'язаних з інтеграцією нових технологій, управлінням великими даними та забезпеченням кібербезпеки.

## 1.2 Роль хмарних обчислень та мультитенантності в підвищенні ефективності виробництв

Хмарні обчислення та мультитенантність відіграють ключову роль у підвищенні ефективності інтелектуальних безперервних виробництв, забезпечуючи гнучкість, масштабованість та оптимізацію ресурсів.

Хмарні обчислення дозволяють підприємствам інтегрувати передові технології, такі як аналітика великих даних, штучний інтелект і машинне навчання, без необхідності створювати та підтримувати власну обчислювальну інфраструктуру. Це значно знижує витрати на ІТ-інфраструктуру та обслуговування, оскільки всі ресурси доступні через Інтернет і можуть бути масштабовані відповідно до потреб виробництва. Завдяки хмарним обчисленням, підприємства можуть швидко адаптуватися до змін у виробничому процесі, підвищувати продуктивність і мінімізувати простой. Наприклад, хмарні платформи надають інструменти для аналізу даних у реальному часі, що дозволяє оперативно виявляти проблеми і приймати обґрунтовані рішення на основі точних даних. Це особливо важливо для безперервних виробництв, де навіть незначні збої можуть мати серйозні наслідки.

Хмарні рішення також забезпечують централізоване управління та моніторинг виробничих процесів, що дозволяє контролювати різні аспекти виробництва з будь-якого місця, де є доступ до Інтернету. Це сприяє підвищенню ефективності управління, оскільки дозволяє керівникам швидко реагувати на зміни, оптимізувати ресурси та координувати дії між різними підрозділами підприємства. Завдяки можливостям хмарних обчислень, підприємства можуть інтегрувати різні системи і процеси, що покращує синергію між різними елементами виробничого циклу.

Мультитенантність, як архітектурний підхід у хмарних обчисленнях, дозволяє підприємствам оптимізувати використання ресурсів, надаючи кільком

користувачам (тенантам) доступ до одного екземпляру програмного забезпечення або інфраструктури. Це означає, що ресурси можуть використовуватися більш ефективно, що знижує загальні витрати на їх підтримку та управління. У контексті безперервних виробництв, мультитенантність дозволяє різним підрозділам підприємства, або навіть різним підприємствам, використовувати одну й ту саму інфраструктуру або додатки, зберігаючи при цьому ізоляцію даних і налаштувань.

Мультитенантність також сприяє підвищенню гнучкості та швидкості впровадження нових технологій. Завдяки централізованому управлінню оновленнями та змінами в системі, підприємства можуть швидше впроваджувати нові функції та адаптуватися до нових виробничих умов. Це особливо важливо в умовах, коли технології швидко змінюються, і підприємства повинні бути здатні оперативно реагувати на ці зміни, щоб залишатися конкурентоспроможними.

При розгляді різних моделей мультитенантності рекомендується спочатку враховувати, як ви визначаєте клієнтів для організації, які бізнес-драйвери та як ви плануєте масштабувати рішення.

Спершу необхідно визначити клієнта для вашої організації. Є дві найпоширеніші моделі.

Бізнес для бізнесу (B2B). Якщо клієнти є іншими організаціями, швидше за все, ви порівнюєте клієтів з цими клієнтами. Однак враховуйте, чи ваші клієнти мають підрозділи (команди або відділи) і наявність присутності в декількох країнах або регіонах. Може знадобитися порівняти одного клієнта з кількома клієнтами, якщо існують різні вимоги для цих підгруп. Аналогічним чином, клієнт може зберегти два екземпляри служби, щоб вони могли зберігати свої середовища розробки та робочого середовища окремо один від одного. Як правило, один клієнт має декілька користувачів. Наприклад, всі співробітники клієнта є користувачами одного клієнта.

Бізнес для споживачів (B2C). Якщо вашими клієнтами є споживачі, часто буває складніше зв'язати клієнтів, орендарів та користувачів. У деяких сценаріях, кожен споживач може бути окремим клієнтом. Проте, розгляньте, чи може ваше рішення використовуватись сім'ями, групами друзів, клубів, асоціацій або іншими групами, які можуть знадобитися для спільного доступу до даних та управління ними. Наприклад, служба потокової передачі музики може підтримувати як окремих користувачів, так і сім'ї, і вона може обробляти кожен із цих типів облікових записів по-різному, коли вона відокремлює їх від клієнтів.

Потім необхідно визначити, який клієнт означає для конкретного рішення, і чи слід розрізняти логічні клієнти та розгортання.

Наприклад, розглянемо службу потокової передачі музики. Спочатку можна створити рішення, яке може легко обробляти тисячі (або десятки тисяч) користувачів. Однак у міру зростання може виникнути необхідність дублювати рішення або деякі компоненти, щоб масштабуватись до нового попиту на клієнтів. Це означає, що необхідно визначити, як призначати конкретних клієнтів певним екземплярам рішення. Ви можете призначити клієнтів випадково або географічно або заповнити один екземпляр, а потім запустити інший (упаковка кошика). Тим не менш, ймовірно, потрібно зберегти запис клієнтів та інфраструктуру своїх даних та додатків, щоб можна було спрямовувати свій трафік у правильну інфраструктуру. У цьому прикладі ви можете представити кожного клієнта як окремого клієнта, а потім зіставити користувачів з розгортанням, що містить дані. У вас є зіставлення "один до багатьох" між клієнтами та розгортаннями, і ви можете переміщати клієнтів між розгортаннями на свій розсуд.

На відміну від цього розглянемо компанію, яка створює хмарне програмне забезпечення для юридичних фірм. Клієнти можуть наполягати на наявності власної виділеної інфраструктури підтримки своїх стандартів відповідності. Тому необхідно підготуватися до розгортання та управління

різними екземплярами рішення із самого початку. У цьому прикладі розгортання завжди містить один клієнт, а клієнт зіставляється із власним виділеним розгортанням.

Ключова різниця між клієнтами та розгортаннями полягає в тому, як застосовується ізоляція. Якщо кілька клієнтів спільно використовують одне розгортання (набір інфраструктури), зазвичай використовується код програми та ідентифікатор клієнта, який знаходиться у базі даних для поділу даних кожного клієнта. Якщо у вас є клієнти з власними виділеними розгортаннями, вони мають власну інфраструктуру, тому вам може бути менш важливо знати, що вона працює в мультитенантному середовищі.

Розгортання іноді називаються супертенантами чи мітками.

При отриманні запиту для конкретного клієнта необхідно зіставити його з розгортанням, у якому зберігаються дані клієнта, як показано нижче (рис. 1.1).

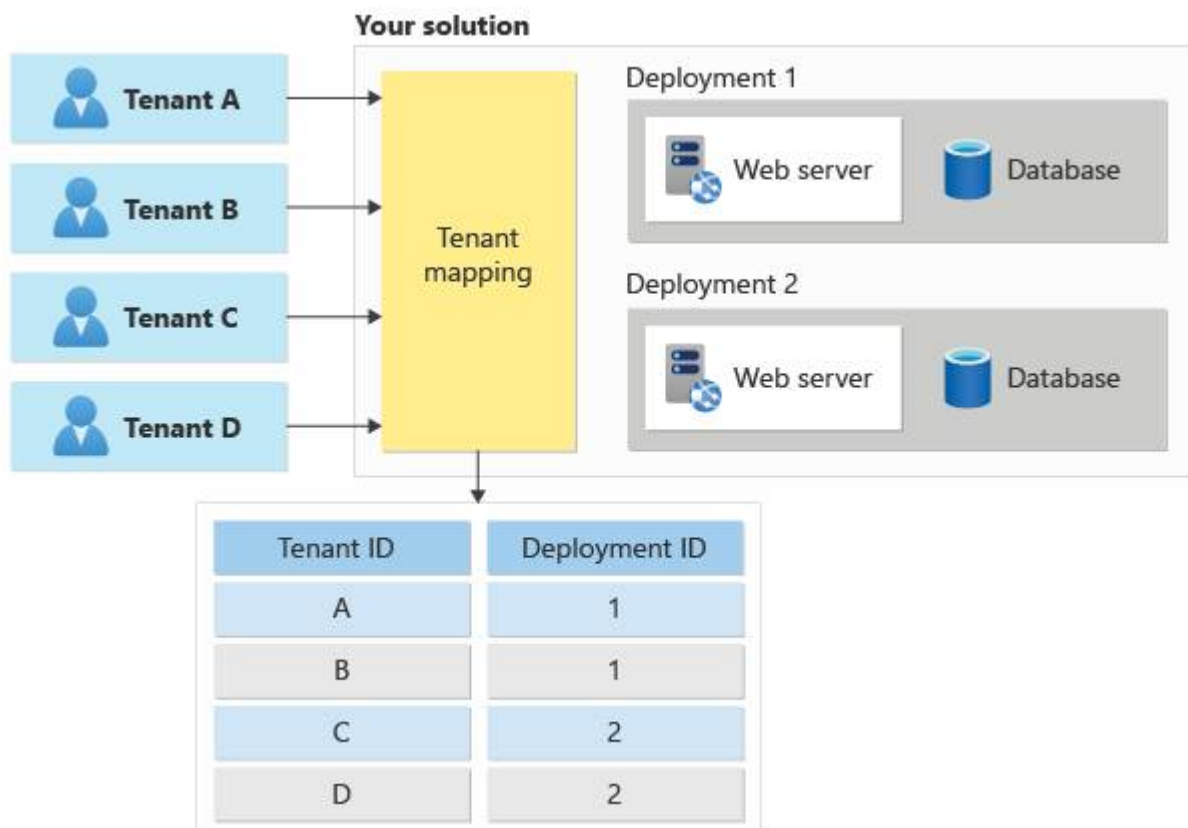


Рис.1.1 - Розгортання, у якому зберігаються дані клієнта

Однією з найбільших міркувань проектування мультитенантної архітектури є рівень ізоляції, необхідний кожному клієнту. Ізоляція може означати різне:

- наявність однієї спільної інфраструктури з окремими екземплярами програми та окремими базами даних кожного клієнта;
- спільне використання деяких загальних ресурсів, але збереження інших ресурсів є окремим для кожного клієнта;
- зберігання даних на окремій фізичній інфраструктурі. У хмарі ця конфігурація може потребувати окремих ресурсів Azure для кожного клієнта. У надзвичайних ситуаціях це може означати навіть розгортання окремої фізичної інфраструктури за допомогою виділених вузлів.

Замість того щоб думати про ізоляцію як дискретну властивість, слід думати про це як про те, що він знаходиться на континуумі. Ви можете розгорнути компоненти своєї архітектури, які більш менш ізольовані, ніж інші компоненти тієї ж архітектури, залежно від ваших вимог. На наступному рисунку показаний спектр можливостей ізоляції.

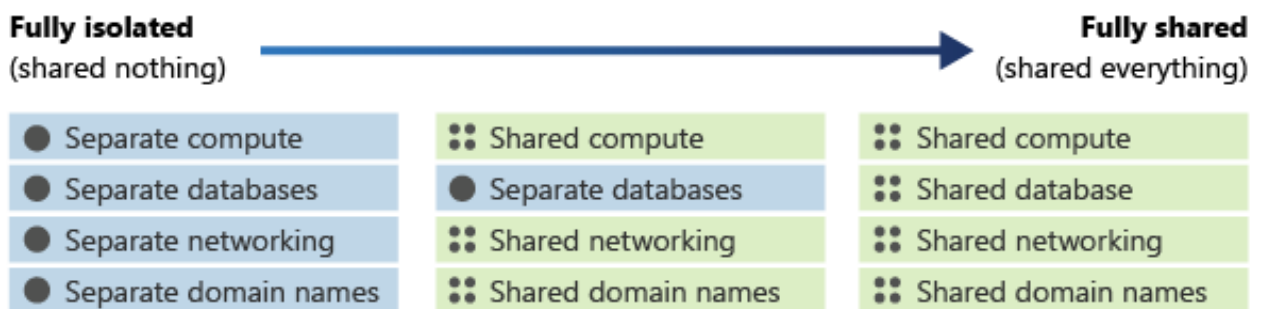


Рис.1.2 - Спектр можливостей ізоляції

Рівень ізоляції впливає на багато аспектів архітектури, включаючи такі як:

- безпека. У разі спільного використання інфраструктури між кількома клієнтами необхідно бути особливо обережним, щоб не отримувати доступ до даних з одного клієнта при поверненні відповідей іншому. Потрібна надійна основа для стратегії ідентифікації, і вам необхідно розглянути як клієнт, так і посвідчення користувача в процесі авторизації;

- вартість. Загальна інфраструктура може використовуватися кількома клієнтами, тож це менш дорого;

- продуктивність. Якщо ви надаєте загальний доступ до інфраструктури, продуктивність вашої системи може знизитись, оскільки її використовують більше клієнтів, оскільки ресурси можуть бути використані швидше. Проблеми з продуктивністю можуть погіршитися клієнтами з незвичайними шаблонами використання;

- надійність. Якщо ви використовуєте один набір загальної інфраструктури, проблема з одним компонентом може спричинити збій для всіх клієнтів;

- швидкість реагування потреби окремого клієнта. При розгортанні інфраструктури, виділеної для одного клієнта, можна адаптувати конфігурацію для ресурсів відповідно до вимог конкретного клієнта. Ви можете розглянути цю можливість у моделі ціноутворення, дозволяючи клієнтам платити більше за ізольовані розгортання.

Архітектура рішення може вплинути на доступні варіанти ізоляції. Наприклад, розглянемо трирівневу архітектуру рішення:

1. Рівень інтерфейсу користувача може бути загальним мультитенантним веб-додатком, при цьому всі клієнти отримують доступ до одного імені вузла.

2. Середній рівень може бути загальним рівнем програми із загальними чергами повідомлень.

3. Рівень даних можна ізолювати бази даних, таблиці або контейнери великих двійкових об'єктів.

Можна використовувати різні рівні ізоляції на кожному рівні. Треба прийняти рішення про загальні та ізольовані можливості, включаючи витрати, складність, вимоги клієнтів та кількість ресурсів, які можна розгорнути, перш ніж досягти квот та обмежень.

Після встановлення вимог можна оцінити їх за деякими загальними моделями оренди та відповідними шаблонами розгортання.

Після встановлення вимог можна оцінити їх за деякими загальними моделями оренди та відповідними шаблонами розгортання. У моделі автоматичного розгортання з одним клієнтом розгортається виділений набір інфраструктури для кожного клієнта, як показано в цьому прикладі:

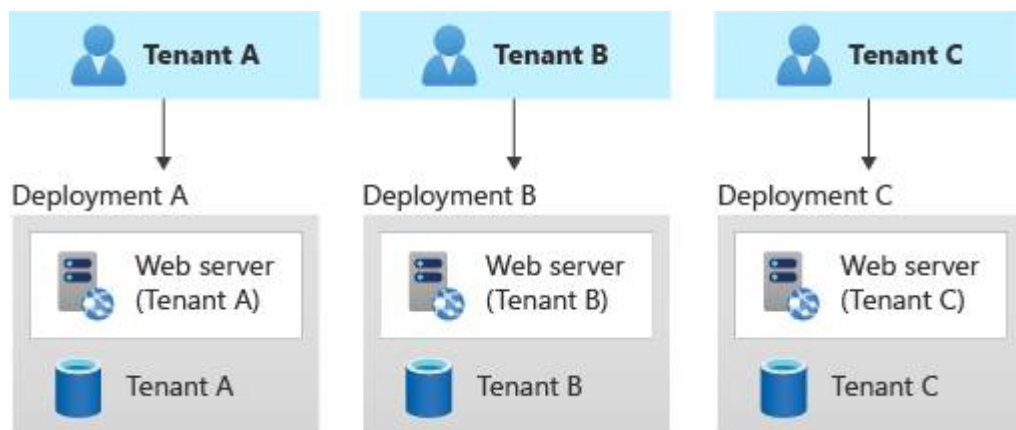


Рис.1.3 - Виділений набір інфраструктури для кожного клієнта

Програма відповідає за ініціювання та координацію розгортання ресурсів кожного клієнта. Як правило, рішення, які використовують цю модель, використовують інфраструктуру як код (IaC) або API Azure Resource Manager. Цей підхід можна використовувати, коли необхідно підготувати окремі інфраструктури для кожного клієнта. При плануванні розгортання рекомендується використовувати шаблон позначок розгортання.

Ключовою перевагою цього підходу є ізоляція даних для кожного клієнта, що знижує ризик випадкового витоку. Цей захист може бути важливим

для деяких клієнтів, які мають високі витрати на відповідність нормативним вимогам. Крім того, клієнти навряд чи впливають на продуктивність системи один одного, проблема, яка іноді називається галасливою проблемою сусіда. Оновлення та зміни можна поступово розгортати по всіх клієнтах, що знижує ймовірність збою всієї системи.

Якщо використовується цей підхід, він забезпечує низьку економічність, так як ви не розділяєте інфраструктуру між клієнтами. Якщо для одного клієнта потрібна певна вартість інфраструктури, то 100 клієнтів, ймовірно, вимагають 100 разів, що коштує. Крім того, поточне обслуговування (наприклад, застосування нових оновлень конфігурації або програмного забезпечення), ймовірно, триватиме багато часу. Треба розглянути можливість автоматизації робочих процесів та можливість поступового застосування змін у середовищах. Крім того, слід розглянути інші операції крос-розгортання, такі як звіти та аналітика з усього флоту.

Можна повністю розглянути мультитенантне розгортання, де всі компоненти спільно використовуються. У вас є лише один набір інфраструктури для розгортання та обслуговування, а всі клієнти використовують її, як показано на наступній схемі (рис.1.4):

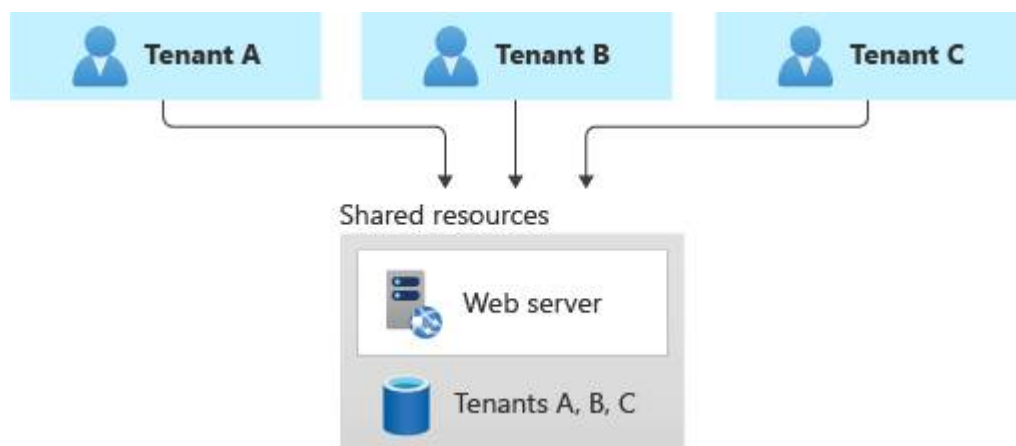


Рис.1.4 - Один набір інфраструктури для розгортання та обслуговування

Ця модель є привабливою, оскільки операційна система із загальними компонентами є менш дорогою, ніж використання окремих ресурсів для кожного клієнта. Навіть якщо необхідно розгорнути більш високі рівні або номери SKU ресурсів для обліку підвищеного навантаження, загальна вартість розгортання часто нижча за вартість набору ресурсів з одним клієнтом. Крім того, якщо користувачеві або клієнту необхідно перемістити дані в інший клієнт, можливо, можна оновити ідентифікатори та ключі клієнта, і може не знадобитися перенести дані між двома окремими розгортаннями.

#### Ризики:

- може знадобитися керувати сегментуванням даних. Крім того, може знадобитися турбуватися про наслідки, які можуть мати окремі клієнти у загальній системі. Наприклад, якщо великий клієнт намагається виконати важкий запит або операцію, це може вплинути на інших клієнтів;

- обслуговування може бути простішим з одним розгортанням, оскільки необхідно оновити лише один набір ресурсів. Однак це часто ризиковано, оскільки зміни можуть вплинути на всю клієнтську базу;

- може знадобитися розглянути можливість масштабування. Наприклад, якщо використовуєте обліковий запис зберігання в рамках рішення зі збільшенням масштабу, кількість запитів до цього облікового запису зберігання може досягти обмеження того, що може обробляти обліковий запис зберігання. Щоб уникнути обмеження квоти ресурсів, можна розглянути можливість розгортання пулу декількох екземплярів ресурсів.

Важливою перевагою мультитенантності є її здатність забезпечити масштабованість системи без значних капіталовкладень. Це дозволяє підприємствам ефективно управляти виробничими процесами на різних рівнях, від окремих виробничих ліній до цілих фабрик, без необхідності розгортати нові системи для кожного рівня управління. Завдяки цьому можна досягти більш високого рівня автоматизації та інтеграції, що безпосередньо впливає на підвищення ефективності та зниження витрат.

У підсумку, хмарні обчислення та мультитенантність забезпечують інтелектуальним безперервним виробництвам значні переваги, включаючи зниження витрат на інфраструктуру, підвищення гнучкості та масштабованості, а також оптимізацію управління та моніторингу виробничих процесів. Ці технології відкривають нові можливості для підприємств, дозволяючи їм ефективніше використовувати свої ресурси, швидше впроваджувати інновації та підвищувати загальну продуктивність виробництва.

### 1.3 Постановка задачі дослідження

Метою цієї кваліфікаційної роботи є розробка та впровадження мультитенантних рішень у хмарні інформаційні системи для підвищення продуктивності, гнучкості та ефективного управління інтелектуальними безперервними виробництвами. Для вирішення поставленої мети треба реалізувати наступні завдання:

- повести аналіз сучасних методів та технологій для побудови мультитенантних рішень;
- запропонувати архітектуру системи з урахуванням вимог до продуктивності, масштабованості та безпеки;
- провести інтеграцію інтелектуальних модулів у мультитенантну архітектуру;
- оцінити вплив інтеграції на загальну ефективність та адаптивність системи.

## 2 АНАЛІЗ ІСНУЮЧИХ ПІДХОДІВ ДО РЕАЛІЗАЦІЇ МУЛЬТИТЕНАНТНОЇ АРХІТЕКТУРИ В ХМАРНИХ СИСТЕМАХ

### 2.1 Огляд сучасних методів та технологій для побудови мультитенантних рішень

Мультитенантність у хмарних обчисленнях дозволяє забезпечити ефективне використання ресурсів, підтримуючи одночасний доступ до однієї системи для кількох клієнтів (тенантів), при цьому ізолюючи їхні дані і налаштування. Існує кілька підходів до реалізації мультитенантної архітектури, кожен з яких має свої особливості.

Перша визначається підходом із спільною інфраструктурою (Shared Infrastructure). У цьому підході всі тенанти використовують ту саму інфраструктуру, включаючи сервери, бази даних та програмне забезпечення. Всі дані зберігаються в одній базі даних, але ізольовані через схеми або логічні розділи. Це дозволяє максимально ефективно використовувати ресурси, але також вимагає високого рівня управління безпекою і конфіденційністю даних. Такий підхід є найбільш економічним і масштабованим, проте ризики пов'язані з можливими витокami даних між тенантами.

Другий підхід із виділеними базами даних (Dedicated Database). У цьому підході кожен тенант має власну базу даних, але всі вони розгорнуті на спільній інфраструктурі. Це забезпечує кращу ізоляцію даних і полегшує управління доступом, проте може бути дорожчим через необхідність підтримки окремих баз для кожного клієнта. Цей підхід є оптимальним для великих організацій або клієнтів з підвищеними вимогами до безпеки.

Третій підхід із виділеною інфраструктурою (Dedicated Infrastructure) передбачає надання кожному тенанту окремої інфраструктури (віртуальних машин, баз даних, серверів). Він забезпечує максимальну ізоляцію і безпеку,

але є найменш економічним з точки зору використання ресурсів. Зазвичай цей підхід використовується для дуже специфічних потреб або для клієнтів із суворими вимогами до безпеки та конфіденційності.

Четвертий - з використанням контейнеризації. Контейнеризація, зокрема Docker та Kubernetes, дозволяє ізолювати додатки та середовища виконання на рівні операційної системи, що робить можливим створення легких і ефективних мультитенантних рішень. Цей підхід дозволяє швидко розгортати нові середовища для тенантів, забезпечуючи високу масштабованість і гнучкість. Контейнери надають високу ізоляцію між додатками, що покращує безпеку та дозволяє ефективніше використовувати ресурси.

## 2.2 Визначення ключових переваг і викликів мультитенантності в контексті інтелектуальних виробництв

Мультитенантність пропонує значні переваги в контексті інтелектуальних безперервних виробництв, зокрема вищу ефективність використання ресурсів та зниження витрат. Завдяки спільній інфраструктурі, підприємства можуть значно зменшити свої витрати на ІТ-інфраструктуру, зберігаючи при цьому високий рівень продуктивності. Мультитенантність дозволяє швидко масштабувати систему, додаючи нових клієнтів або підрозділи без необхідності значних інвестицій у додаткове обладнання. Крім того, цей підхід сприяє покращенню управління та моніторингу, оскільки всі операції здійснюються з єдиного центру управління, що спрощує адміністрування і контроль за ресурсами.

Незважаючи на численні переваги, мультитенантність також несе в собі низку викликів, особливо у сфері безпеки і конфіденційності даних. Спільне використання ресурсів між кількома клієнтами підвищує ризик витоку даних або несанкціонованого доступу. Це вимагає впровадження складних механізмів шифрування, управління доступом і моніторингу, що може значно ускладнити

архітектуру системи. Крім того, управління продуктивністю у мультитенантному середовищі може бути складним, оскільки потреби різних тенантів можуть значно відрізнятись, що потребує розробки гнучких і адаптивних рішень для балансування навантаження. Ще одним викликом є забезпечення відповідності правовим і регуляторним вимогам, особливо у випадках, коли дані різних тенантів мають зберігатися у різних юрисдикціях або підлягають різним правовим нормам.

Таким чином, мультитенантність у хмарних інформаційних системах для інтелектуальних виробництв є потужним інструментом для підвищення ефективності, але потребує ретельного підходу до розробки архітектури, безпеки та управління ресурсами.

З огляду на специфіку інтелектуальних безперервних виробництв та можливості сучасних технологій, найдоцільнішим підходом для реалізації мультитенантної архітектури є підхід з використанням контейнеризації. Цей підхід поєднує в собі гнучкість, ефективність використання ресурсів та високий рівень ізоляції, що особливо важливо для складних виробничих середовищ.

Контейнери дозволяють швидко розгортати і масштабувати додатки для кожного тенанта. У випадку з інтелектуальними виробництвами, де потреби в ресурсах можуть змінюватися залежно від обсягу виробництва або кількості підключених пристроїв, можливість швидкого масштабування є критично важливою. Також вони дозволяють реалізувати ефективність використання ресурсів. Контейнери легкі, займають менше ресурсів порівняно з віртуальними машинами, що дозволяє більш ефективно використовувати наявні обчислювальні потужності. Це допомагає знизити загальні витрати на інфраструктуру, що особливо важливо для підприємств з великими обсягами даних.

Контейнери забезпечують чітку ізоляцію середовищ виконання, що важливо для підтримки конфіденційності і безпеки даних між різними

тенантами. Це дозволяє знизити ризики, пов'язані з витоком даних або несанкціонованим доступом. Використання Kubernetes (K8s) як оркестратора контейнерів дозволяє інтегрувати контейнери з різними хмарними платформами, такими як AWS, Azure чи Google Cloud. Це забезпечує високу доступність і надійність системи, а також полегшує управління мультитенантною архітектурою.

### 2.3 Огляд особливостей інтелектуальних виробничих систем

Інтелектуальні виробничі системи (ІВС) — це сучасні виробничі платформи, які інтегрують передові технології автоматизації, збору та аналізу даних, а також штучного інтелекту для оптимізації та управління виробничими процесами. Ці системи дозволяють підприємствам значно підвищити продуктивність, якість продукції та гнучкість виробництва, забезпечуючи при цьому зниження витрат і підвищення конкурентоспроможності на ринку. Однією з ключових особливостей ІВС є їх здатність забезпечувати безперервний моніторинг і управління виробничими процесами в режимі реального часу. Завдяки інтеграції сенсорних мереж, Інтернету речей (ІоТ) і кіберфізичних систем (СРS), ІВС збирають величезні обсяги даних про стан обладнання, якість продукції, використання ресурсів і інші важливі параметри. Ці дані використовуються для негайного аналізу та прийняття рішень, що дозволяє оперативно реагувати на будь-які відхилення або проблеми у виробничому процесі.

Інша важлива особливість інтелектуальних виробничих систем — їх висока адаптивність. ІВС можуть автоматично налаштовувати виробничі процеси залежно від змін у середовищі або вимог ринку. Це дозволяє підприємствам швидко реагувати на зміну попиту, впроваджувати нові продукти без суттєвих затримок і адаптуватися до нових умов виробництва без зниження продуктивності. Наприклад, у виробництвах із високим рівнем

персоналізації продукції, ІВС дозволяють швидко переналаштовувати обладнання під нові специфікації продукту.

ІВС також мають високу ступінь інтеграції з іншими бізнес-системами підприємства, такими як ERP (системи управління ресурсами підприємства) та SCM (системи управління ланцюгами постачання). Це забезпечує безперервний обмін даними між виробничим підрозділом та іншими департаментами компанії, що дозволяє більш точно планувати виробничі цикли, оптимізувати запаси і знижувати витрати. Інтеграція з хмарними обчисленнями та великими даними (Big Data) дозволяє підприємствам використовувати передові аналітичні інструменти для прогнозування попиту, управління ланцюгами постачання та підвищення загальної ефективності виробництва.

#### 2.4 Роль автоматизації та інтелектуальних алгоритмів у підвищенні ефективності виробництв

Автоматизація є одним із ключових компонентів інтелектуальних виробничих систем, що дозволяє підвищити продуктивність, скоротити час на виконання завдань і знизити людські помилки. Автоматизовані системи управління виробництвом використовують різні програмно-апаратні засоби для автоматизації операцій, починаючи від обробки сировини і закінчуючи складанням і пакуванням готової продукції. Завдяки цьому виробничі процеси стають більш стабільними і передбачуваними, що забезпечує підвищення якості продукції та зниження витрат на виробництво.

Інтелектуальні алгоритми, включаючи алгоритми машинного навчання і штучного інтелекту, відіграють критично важливу роль у підвищенні ефективності виробничих систем. Вони забезпечують можливість аналізу великих обсягів даних, зібраних з різних виробничих процесів, для виявлення прихованих закономірностей, прогнозування можливих проблем та оптимізації

використання ресурсів.

Автоматизація виробничих процесів, особливо у вигляді використання промислових роботів і автоматизованих конвеєрів, дозволяє значно знизити витрати на виробництво і підвищити продуктивність. Роботи здатні виконувати рутинні, важкі або небезпечні завдання з високою точністю і швидкістю, що знижує залежність від людської праці і підвищує безпеку на виробництві. Завдяки автоматизації зменшується час простоїв і втрат, пов'язаних з людським фактором, що особливо важливо в безперервних виробництвах, де зупинка процесу може мати суттєві фінансові наслідки.

Інтелектуальні алгоритми дозволяють вирішувати складні завдання оптимізації виробничих процесів. Наприклад, алгоритми машинного навчання можуть аналізувати історичні дані про роботу обладнання для прогнозування можливих відмов і планування технічного обслуговування. Це дозволяє знизити ризики несподіваних поломок і мінімізувати втрати, пов'язані з простоєм обладнання. Алгоритми оптимізації використовуються для покращення управління ресурсами на виробництві. Вони дозволяють автоматично налаштовувати параметри виробничих процесів, такі як швидкість лінії, температура або тиск, залежно від поточних умов або вимог до якості продукції. Це допомагає знизити витрати на енергію, зменшити відходи і підвищити загальну ефективність виробництва.

Однією з найважливіших функцій інтелектуальних алгоритмів є прогнозування. Використовуючи історичні дані і сучасні методи аналітики, такі як регресійний аналіз або нейронні мережі, можна прогнозувати майбутні події, наприклад, коливання попиту, зміну параметрів обладнання або можливі затримки в постачаннях. Це дозволяє заздалегідь підготуватися до змін, оптимізувати виробничі плани і зменшити ризики. Крім того, аналітичні системи можуть автоматично формувати звіти і пропозиції для керівництва підприємства на основі отриманих даних. Це дозволяє приймати більш обґрунтовані рішення, що позитивно впливає на стратегічне планування і

загальну ефективність бізнесу.

Інтелектуальні виробничі системи здатні інтегрувати різні частини виробничого процесу в єдину, добре скоординовану систему. Це включає координацію роботи окремих ліній або цехів, а також інтеграцію з системами постачання і розподілу продукції. Завдяки цьому підприємства можуть оперативно адаптуватися до змін у виробничому циклі, мінімізувати затримки і втрати на різних етапах виробництва, і забезпечувати стабільну якість продукції.

Інтелектуальні виробничі системи та автоматизація кардинально змінюють підхід до організації виробничих процесів, підвищуючи їхню ефективність, гнучкість і надійність. Завдяки інтеграції сучасних технологій, таких як автоматизація, штучний інтелект і аналітика великих даних, підприємства отримують можливість оптимізувати свої виробничі процеси, знижувати витрати і підвищувати якість продукції. Це робить інтелектуальні виробничі системи невід'ємною частиною сучасної промисловості, дозволяючи підприємствам залишатися конкурентоспроможними в умовах швидко змінюваного ринку.

Інтелектуальні виробничі системи функціонують в умовах, де ключовими критеріями ефективності є продуктивність, надійність, масштабованість і безпека. Ці вимоги визначають здатність системи забезпечувати безперебійну роботу, адаптуватися до змінних умов виробництва і захищати критичні дані від загроз.

Продуктивність є основоположним показником для інтелектуальних виробничих систем, оскільки від неї залежить швидкість та ефективність виконання виробничих процесів. Високий рівень продуктивності системи означає здатність обробляти великі обсяги даних у реальному часі, виконувати складні розрахунки для оптимізації процесів і забезпечувати швидке прийняття рішень.

Для досягнення необхідного рівня продуктивності ІВС повинна повинна

мати можливість обробляти дані і виконувати операції з мінімальною затримкою, щоб забезпечувати оперативний контроль і корекцію виробничих процесів. Це особливо важливо для безперервних виробництв, де навіть незначні затримки можуть призвести до серйозних втрат.

ІВС повинна підтримувати високий рівень пропускну здатності для обробки великих обсягів даних, що надходять від численних сенсорів і пристроїв, інтегрованих у виробничий процес, також вона повинна ефективно використовувати доступні обчислювальні ресурси для виконання завдань, мінімізуючи витрати на обладнання та енергію.

Надійність ІВС є критично важливою, оскільки вона визначає здатність системи стабільно функціонувати в умовах безперервного виробництва, де зупинка процесу може призвести до значних фінансових втрат.

Система повинна бути спроектована таким чином, щоб мінімізувати ризики відмови обладнання або програмного забезпечення. Це може бути досягнуто за рахунок використання резервних копій даних, дублікатів критично важливих компонентів і можливості автоматичного відновлення після відмови.

ІВС повинна забезпечувати безперервність роботи, навіть у разі виникнення несправностей або збоїв. Це може включати автоматичне перемикання на резервні системи або дублювання ключових процесів.

Система повинна мати можливість постійного моніторингу та діагностики свого стану для своєчасного виявлення і усунення потенційних проблем. Це знижує ризик несподіваних збоїв і забезпечує стабільну роботу виробничого процесу.

Масштабованість визначає здатність ІВС адаптуватися до зростаючих потреб виробництва або змін у його конфігурації без втрати ефективності чи продуктивності.

Система повинна мати можливість збільшувати свою продуктивність шляхом додавання нових обчислювальних ресурсів (серверів, пристроїв), що

дозволить обробляти більшу кількість даних або обслуговувати більше виробничих ліній.

ІВС повинна підтримувати можливість оновлення апаратного та програмного забезпечення для підвищення продуктивності існуючих ресурсів. Це дозволяє оптимізувати використання інфраструктури без необхідності значних інвестицій у нове обладнання. Система повинна бути спроектована з урахуванням можливості швидкого налаштування під нові завдання або зміни у виробничих процесах, що дозволить адаптуватися до потреб бізнесу без серйозних перебудов.

Безпека є однією з найважливіших вимог до інтелектуальних виробничих систем, особливо у контексті обробки конфіденційних даних та захисту від кіберзагроз.

Вимоги до безпеки включають:

- захист даних. ІВС повинна забезпечувати надійне шифрування даних як під час їх передачі, так і при зберіганні, щоб запобігти несанкціонованому доступу або витоку інформації. Це особливо важливо для захисту комерційної таємниці та іншої критично важливої інформації;

- управління доступом. Система повинна мати чітко визначені політики управління доступом, що регулюють права користувачів на доступ до різних частин системи і даних. Це включає використання багатофакторної автентифікації, ролей і політик доступу, що забезпечує захист від внутрішніх і зовнішніх загроз;

- кібербезпека. ІВС повинна бути оснащена засобами захисту від кіберзагроз, такими як міжмережеві екрани (firewalls), системи виявлення вторгнень (IDS/IPS) та антивірусні програми. Це допомагає запобігти атак на систему і мінімізувати ризики компрометації;

- відповідність регуляторним вимогам. Система повинна відповідати всім нормативним вимогам, які стосуються безпеки даних і захисту конфіденційної інформації, зокрема стандартам GDPR, ISO/IEC 27001 та

іншим.

Інтелектуальні виробничі системи вимагають високих стандартів щодо продуктивності, надійності, масштабованості та безпеки. Задоволення цих вимог є критичним для забезпечення стабільної роботи виробництва, швидкої адаптації до змін і захисту від зовнішніх та внутрішніх загроз. Таким чином, ці критерії слугують фундаментом для успішного впровадження та ефективної експлуатації інтелектуальних виробничих систем у сучасних умовах промисловості.

## 3 РОЗРОБКА АРХІТЕКТУРИ ХМАРНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ З ПІДТРИМКОЮ МУЛЬТИТЕНАНТНОСТІ

### 3.1 Основні компоненти архітектури хмарної інформаційної системи з підтримкою мультитенантності

Розробка архітектури хмарної інформаційної системи з підтримкою мультитенантності є важливим кроком для забезпечення ефективності, масштабованості та безпеки в інтелектуальних виробничих середовищах. Мультитенантна архітектура дозволяє одночасно обслуговувати кілька клієнтів (тенантів) в одній системі, зберігаючи ізоляцію даних і забезпечуючи оптимальне використання ресурсів.

Основні компоненти архітектури:

1) управління тенантами (Tenant Management):

- ідентифікація та автентифікація тенантів. Для кожного тенанта система повинна забезпечувати унікальний ідентифікатор та процедуру автентифікації. Це може бути реалізовано через системи управління ідентифікацією, такі як OAuth або SAML, що забезпечують безпечний доступ до системи;

- ізоляція даних та конфігурацій. Дані кожного тенанта повинні бути ізольовані від інших, щоб уникнути витоку інформації. Це може бути досягнуто за допомогою різних схем баз даних (наприклад, використання окремих схем або навіть окремих баз даних для кожного тенанта) або на рівні додатків через логічне розділення;

2) модуль управління даними (Data Management Layer):

- обробка і зберігання даних. Використання реляційних (SQL) або нереляційних (NoSQL) баз даних для зберігання даних тенантів. Модуль управління даними повинен забезпечувати ефективне зберігання і швидкий

доступ до даних, а також шифрування на рівні баз даних для підвищення безпеки;

- обробка запитів. Механізми обробки запитів повинні враховувати специфічні вимоги кожного тенанта, наприклад, через оптимізацію запитів до бази даних, кешування та балансування навантаження;

3) модуль обчислень (Computation Layer):

- масштабування ресурсів. Цей модуль відповідає за динамічне масштабування обчислювальних ресурсів, залежно від потреб кожного тенанта. Це може бути досягнуто через використання контейнеризації (Docker) та оркестрації контейнерів (Kubernetes), що дозволяє автоматично додавати або видаляти ресурси, залежно від навантаження;

- інтеграція з сервісами штучного інтелекту та машинного навчання. Модуль обчислень повинен забезпечувати можливість інтеграції з сервісами AI/ML для інтелектуального аналізу даних і автоматизації виробничих процесів;

4) модуль управління доступом та безпеки (Security and Access Management):

- управління правами доступу визначається тим, що всі користувачі системи повинні мати відповідні права доступу, залежно від їх ролі та статусу тенанта. Це включає реалізацію ролей та політик доступу для кожного тенанта, що забезпечує захист даних від несанкціонованого доступу;

- моніторинг і виявлення загроз — це реалізація інструментів моніторингу, таких як SIEM (Security Information and Event Management), для виявлення підозрілих активностей та запобігання можливим загрозам безпеці;

5) модуль інтерфейсу користувача (User Interface Layer):

- персоналізація інтерфейсу стосується того, що кожен тенант повинен мати можливість налаштовувати інтерфейс відповідно до своїх потреб. Це може включати налаштування дашбордів, звітів і панелей управління;

- доступність полягає в тому, що інтерфейс користувача повинен бути доступним через різні платформи та пристрої (веб, мобільні додатки), забезпечуючи зручний доступ до всіх функцій системи;

б) модуль інтеграції з зовнішніми системами (External Integration Layer):

- API і Webhooks. Для забезпечення інтеграції з іншими системами і сервісами, наприклад, ERP або SCM, необхідно реалізувати API та вебхуки, що дозволяють обмінюватися даними в режимі реального часу;

- сумісність з хмарними сервісами. Забезпечення сумісності з основними хмарними платформами (AWS, Azure, GCP), що дозволить використовувати сторонні сервіси для розширення функціональних можливостей системи.

У мультитенантній хмарній інформаційній системі потік даних повинен бути чітко визначений, щоб забезпечити ізоляцію, безпеку та ефективність обробки даних. Взаємодія між компонентами відбувається наступним чином.

Користувач (тенант) ініціює запит через інтерфейс користувача, який обробляється модулем управління доступом для перевірки прав і автентифікації.

Після автентифікації запит передається в модуль обчислень, де він може бути оброблений за допомогою спеціалізованих алгоритмів, інтегрованих сервісів AI/ML або стандартних обчислювальних процедур.

Оброблені запити направляються до модуля управління даними, який витягує або записує дані в базу даних, забезпечуючи ізоляцію на рівні тенантів.

Після завершення обробки результати передаються назад через модуль обчислень і відправляються користувачу через інтерфейс.

Для забезпечення надійності та відмовостійкості системи необхідно реалізувати механізми резервування обчислювальних ресурсів, баз даних і мережевих компонентів для забезпечення роботи системи у разі відмови основних елементів.

Також використання інструментів моніторингу для відстеження стану системи і виявлення можливих проблем на ранній стадії і Автоматичне відновлення. Реалізація механізмів автоматичного відновлення компонентів системи у разі збою, щоб мінімізувати час простою.

Архітектура хмарної інформаційної системи з підтримкою мультитенантності забезпечує гнучкість, масштабованість та безпеку, зберігаючи при цьому високу продуктивність та надійність (рис.3.1). Схема розроблена у редакторі Mermaid. Розробка такої архітектури включає модулі управління тенантами, даними, обчисленнями, безпекою, інтерфейсом користувача і інтеграцією з зовнішніми системами. Забезпечення належної взаємодії між цими модулями і впровадження механізмів для відмовостійкості гарантують ефективне функціонування системи в умовах інтелектуальних безперервних виробництв.

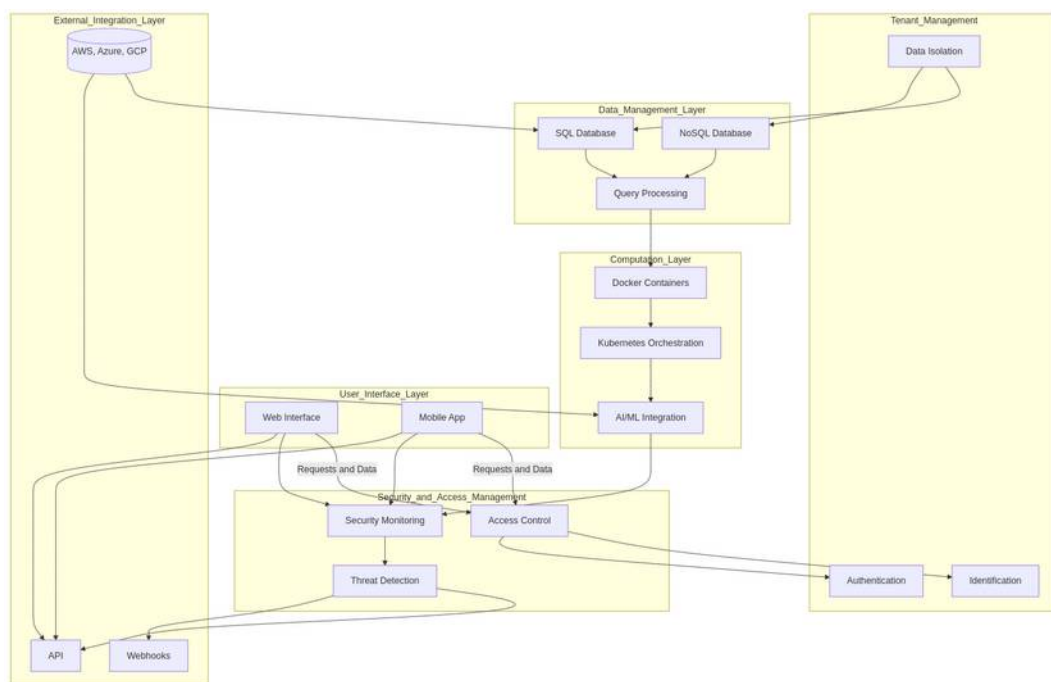


Рис.3.1 - Архітектура хмарної інформаційної системи з підтримкою мультитенантності

Опис схеми архітектури:

1) User Interface Layer (Шар інтерфейсу користувача) включає Web Interface (Веб-інтерфейс) і Mobile App (Мобільний додаток). Вони

представляють два основних способи взаємодії користувачів із системою. Користувачі можуть надсилати запити і отримувати дані через ці інтерфейси. Веб-інтерфейс підходить для роботи на комп'ютерах і планшетах, тоді як мобільний додаток забезпечує зручність доступу з мобільних пристроїв.

2) Tenant Management (Управління тенантами) складається з Identification (Ідентифікація) і Authentication (Автентифікація), що відповідають за перевірку користувачів і визначення їх належності до конкретного тенанта (користувача або організації). Цей модуль забезпечує безпечний доступ до системи і визначає права кожного тенанта, а також з Data Isolation (Ізоляція даних), яка гарантує, що дані кожного тенанта зберігаються і обробляються ізольовано від інших, забезпечуючи конфіденційність і безпеку.

3) Data Management Layer (Шар управління даними). Це SQL Database (Реляційна база даних) і NoSQL Database (Нереляційна база даних), які використовуються для зберігання даних тенантів. Реляційні бази даних підходять для структурованих даних, тоді як нереляційні забезпечують гнучкість у роботі з неструктурованими або варіативними даними. До цього шару входить також Query Processing (Обробка запитів), що відповідає за виконання і оптимізацію запитів до баз даних, забезпечуючи швидкий доступ до необхідної інформації.

4) Computation Layer (Шар обчислень). До цього шару входять Docker Containers (Контейнери Docker), які забезпечують ізольоване середовище для додатків, що дозволяє легко масштабувати та керувати ресурсами, також Kubernetes Orchestration (Оркестрація Kubernetes), що управляє розгортанням, масштабуванням і підтримкою контейнерів у хмарному середовищі, забезпечуючи гнучкість та надійність системи і AI/ML Integration (Інтеграція з AI/ML сервісами), яка дозволяє використовувати алгоритми штучного інтелекту та машинного навчання для аналізу даних і автоматизації виробничих процесів, що підвищує ефективність системи.

5) Security and Access Management (Управління безпекою та доступом). До цього шару входять:

- Access Control (Управління доступом), яке визначає політики доступу користувачів до різних частин системи на основі їх ролей і рівня доступу;
- Security Monitoring (Моніторинг безпеки), що постійно відстежує стан системи для виявлення підозрілої активності та потенційних загроз;
- Threat Detection (Виявлення загроз) - аналізує активність у системі для своєчасного виявлення і реагування на загрози, забезпечуючи захист даних і стабільність роботи;

6) External Integration Layer (Шар інтеграції із зовнішніми системами):

- API забезпечує інтерфейс для взаємодії з іншими системами та додатками, що дозволяє інтегрувати хмарну систему з іншими корпоративними рішеннями, такими як ERP, CRM тощо;
- Webhooks використовуються для передачі даних та подій у реальному часі між системами, що забезпечує гнучкість у реагуванні на зовнішні події;
- Cloud Compatibility (Сумісність з AWS, Azure, GCP) забезпечує можливість роботи з основними хмарними платформами, розширюючи можливості системи для використання додаткових сервісів і інфраструктури.

Ця архітектура хмарної інформаційної системи з підтримкою мультитенантності забезпечує комплексний підхід до управління даними, обчислювальними ресурсами та безпекою для інтелектуальних безперервних виробництв. Вона враховує необхідність ізоляції даних, гнучкості масштабування, високого рівня безпеки та інтеграції з іншими системами, що робить її ефективною і надійною для сучасних промислових умов.

### 3.2 Інтеграція інтелектуальних модулів у мультитенантну архітектуру

Інтеграція модулів штучного інтелекту (AI) та машинного навчання (ML) у мультитенантну архітектуру є важливим етапом у створенні інтелектуальних виробничих систем, які здатні автоматизувати і оптимізувати виробничі процеси на основі аналізу даних. Завдяки цим технологіям підприємства можуть автоматизувати складні процеси, покращити якість продукції, знизити витрати та підвищити гнучкість у відповідь на зміни ринкових умов. Інтелектуальні алгоритми стають невід'ємною частиною сучасного виробництва, забезпечуючи підтримку і оптимізацію всіх ключових аспектів виробничих процесів. Така інтеграція дозволяє значно підвищити ефективність системи, надаючи можливість швидкого прийняття рішень та адаптації до змін. Кроки інтеграції AI/ML модулів у мультитенантну архітектуру:

- аналіз вимог та вибір алгоритмів. На початковому етапі необхідно провести детальний аналіз виробничих процесів та визначити, які саме задачі можуть бути автоматизовані або оптимізовані за допомогою AI/ML. Це можуть бути задачі прогнозування попиту, управління запасами, оптимізації виробничих ліній або виявлення дефектів у продукції. Після цього обираються відповідні алгоритми машинного навчання, такі як класифікація, регресія, кластеризація або алгоритми глибокого навчання, залежно від конкретних потреб виробництва;

- підготовка даних. AI/ML модулі потребують якісних даних для навчання і роботи. Тому на цьому етапі проводиться збір, очищення та підготовка даних з виробничих систем. Дані можуть включати інформацію з сенсорів, історичні дані про роботу обладнання, дані про продуктивність, витрати матеріалів тощо. У мультитенантному середовищі важливо забезпечити, щоб дані кожного тенанта були чітко ізольовані, а також щоб алгоритми могли адаптуватися до специфічних потреб кожного тенанта;

- розробка та навчання моделей. Розробка моделей машинного навчання відбувається на основі підготовлених даних. Моделі навчаються на

історичних даних для того, щоб прогнозувати результати або автоматизувати певні процеси. У разі мультитенантної архітектури можливо налаштувати моделі таким чином, щоб вони могли працювати з даними від різних тенантів або навіть навчатися окремо для кожного тенанта.

- інтеграція моделей у систему. Після навчання моделі інтегруються у мультитенантну архітектуру. Це передбачає підключення моделей до основного модуля обчислень (Computation Layer) через API або спеціалізовані сервіси, такі як TensorFlow Serving, AWS SageMaker, або Azure Machine Learning. Моделі можуть бути розгорнуті у контейнерах Docker і управлятися за допомогою Kubernetes, що забезпечує їх масштабованість та надійність;

- реалізація інтерфейсів для взаємодії. Інтерфейси користувача (User Interface Layer) повинні бути адаптовані для відображення результатів роботи AI/ML моделей, таких як прогнози, рекомендації або попередження. Це дозволяє користувачам легко отримувати доступ до аналітики та приймати обґрунтовані рішення. Крім того, результати можуть передаватися іншим системам через зовнішні інтеграційні модулі (External Integration Layer), що дозволяє використовувати їх у плануванні, управлінні запасами або інших бізнес-процесах.

- моніторинг і оновлення моделей. AI/ML моделі потребують постійного моніторингу та періодичного оновлення. Це необхідно для забезпечення їх актуальності та точності. Система повинна включати механізми автоматичного оновлення моделей на основі нових даних, а також інструменти для оцінки їхньої продуктивності.

Інтелектуальні алгоритми відіграють ключову роль у підтримці та оптимізації виробничих процесів у рамках інтелектуальних виробничих систем. Їхня роль полягає у забезпеченні аналізу великих обсягів даних, автоматизації прийняття рішень і адаптації виробничих процесів до змінних умов.

## 4 ВПРОВАДЖЕННЯ ЗАПРОПОНОВАНОЇ АРХІТЕКТУРИ В РЕАЛЬНОМУ СЕРЕДОВИЩІ

### 4.1 Основні етапи впровадження та їх особливості

Впровадження запропонованої мультитенантної архітектури в реальному виробничому середовищі є складним процесом, який вимагає чіткої стратегії та використання сучасних хмарних технологій. Основні етапи цього процесу включають планування, підготовку інфраструктури, розгортання системи, інтеграцію, тестування та оптимізацію.

На початковому етапі важливо провести детальне планування процесу впровадження. Це включає:

- оцінка вимог. Необхідно точно визначити технічні вимоги до системи, включаючи потреби в ресурсах, пропускну здатність, вимоги до безпеки та масштабованості;
- аналіз існуючої інфраструктури. Оцінка поточної ІТ-інфраструктури підприємства, щоб визначити, які компоненти можна використовувати, а які потребують модернізації або заміни;
- розробка дорожньої карти впровадження. Створення поетапного плану дій з визначенням ключових етапів, ресурсів і термінів для кожного з них.

Другий етап – це підготовка інфраструктури. Перед тим, як розпочати розгортання системи, необхідно підготувати інфраструктуру. Визначтеся з хмарною платформою, яка найкраще відповідає потребам підприємства AWS, Microsoft Azure, Google Cloud Platform. Вибір залежить від необхідних сервісів, вимог до безпеки, сумісності та вартості.

Забезпечити необхідну мережеву інфраструктуру для підтримки надійного і безпечного підключення між хмарними сервісами та локальними системами.

Також необхідно налаштувати віртуальні машини, контейнери та інші обчислювальні ресурси на обраній хмарній платформі.

На етапі розгортання системи відбувається безпосереднє розгортання компонентів системи:

- розгортання контейнерів Docker;
- оркестрація за допомогою Kubernetes;
- розгортання баз даних. Налаштування реляційних (SQL) та нереляційних (NoSQL) баз даних для зберігання даних тенантів.

Після розгортання основних компонентів, необхідно інтегрувати систему з іншими бізнес-додатками та сервісами, а саме:

- інтеграція API. Це дозволить забезпечити двосторонній обмін даними між хмарною системою і корпоративними додатками;
- налаштування Webhooks. Це дозволить системі оперативно реагувати на зовнішні зміни або події, зберігаючи актуальність даних.
- сумісність із сторонніми сервісами (AWS, Azure, GCP), що дозволить використовувати додаткові сервіси для обробки даних, машинного навчання або аналізу.

Після завершення розгортання необхідно провести тестування системи.

Після завершення тестування та оптимізації система готова до повноцінного впровадження:

Для успішного розгортання та управління мультитенантною архітектурою інтелектуальних виробничих систем варто використовувати такі хмарні платформи та інструменти:

- Amazon Web Services (AWS) надає широкий спектр сервісів, включаючи Amazon EC2 для обчислень, Amazon RDS для баз даних, Amazon

SageMaker для машинного навчання, а також сервіси для оркестрації контейнерів (Amazon EKS);

- Microsoft Azure пропонує потужні інструменти для розгортання AI/ML моделей, таких як Azure Machine Learning, а також сервіси для управління базами даних (Azure SQL Database) і контейнеризації (Azure Kubernetes Service);

- Google Cloud Platform (GCP) включає сервіси для оркестрації контейнерів (Google Kubernetes Engine), управління базами даних (Google Cloud SQL, Google Cloud Datastore) та інструменти для машинного навчання (Google AI Platform).

Впровадження мультитенантної архітектури в реальному середовищі — це комплексний процес, який включає планування, підготовку інфраструктури, розгортання, інтеграцію з іншими системами, тестування та постійну підтримку. Використання хмарних платформ, таких як AWS, Azure або GCP, забезпечує гнучкість, масштабованість та безпеку, необхідні для успішного впровадження і подальшої експлуатації системи в умовах інтелектуальних виробничих процесів.

#### 4.2 Створення та розгортання додатків у контейнерах Docker

Для створення та розгортання додатків у контейнерах Docker, які будуть використовуватися в мультитенантному середовищі, необхідно виконати кілька ключових кроків. Цей процес включає налаштування середовища розробки, створення Docker-контейнерів для кожного тенанта, забезпечення ізоляції даних і налаштування контейнерів для інтеграції з іншими компонентами системи.

Перш за все, необхідно налаштувати середовище для роботи з Docker:

Потім налаштувати Docker Compose. Docker Compose дозволяє одночасно керувати кількома контейнерами, що корисно для мультитенантного середовища.

Кожен додаток, який буде працювати в мультитенантному середовищі, потрібно контейнеризувати:

Для кожного додатка створюється `Dockerfile`, який визначає середовище виконання, залежності та команди для розгортання додатка. Наприклад, для Node.js-дodatка `Dockerfile` може виглядати так:

```
``Dockerfile
# Використовуємо офіційний образ Node.js
FROM node:14

# Створюємо робочу директорію
WORKDIR /app

# Копіюємо файли package.json та package-lock.json
COPY package*.json ./

# Встановлюємо залежності
RUN npm install

# Копіюємо вихідні файли додатка
COPY . .

# Відкриваємо порт для прослуховування
EXPOSE 3000

# Команда для запуску додатка
CMD ["node", "app.js"]
````
```

Використовуємо команду ``docker build``, щоб створити Docker-образ додатка на основі ``Dockerfile``. Виконуємо команду в терміналі:

```
```bash
docker build -t myapp:latest .
```
```

Тут ``myapp`` — це ім'я образу, а ``latest`` — версія образу.

Якщо кожен тенант потребує різних конфігурацій (наприклад, змінні середовища), створіть окремі конфігураційні файли для кожного з них або використовуйте ``docker-compose.yml`` для визначення окремих служб для кожного тенанта.

Для мультитенантного середовища критично важливо забезпечити ізоляцію даних між різними тенантами. Визначаємо змінні середовища, які будуть налаштовуватися окремо для кожного тенанта. Наприклад, змінні для підключення до різних баз даних та використовуємо Docker Network для створення окремих віртуальних мереж для різних тенантів, щоб їхні контейнери не мали прямого доступу до контейнерів інших тенантів.

```
```bash
docker network create tenant1_network
docker network create tenant2_network
```
```

Якщо використовується одна база даних для кількох тенантів, забезпечте розмежування даних на рівні схеми або таблиць. Наприклад, кожен тенант може мати власну схему в базі даних.

Для управління кількома контейнерами одночасно і спрощення їх налаштування, використовуємо Docker Compose:

```
```yaml
version: '3'
services:
  tenant1_app:
    image: myapp:latest
    environment:
      - DB_HOST=tenant1_db
    networks:
      - tenant1_network

  tenant1_db:
    image: postgres:latest
    environment:
      - POSTGRES_DB=tenant1db
    networks:
      - tenant1_network

  tenant2_app:
    image: myapp:latest
    environment:
      - DB_HOST=tenant2_db
    networks:
      - tenant2_network

  tenant2_db:
    image: postgres:latest
    environment:
      - POSTGRES_DB=tenant2db
    networks:
```

```
- tenant2_network
```

```
networks:
  tenant1_network:
  tenant2_network:
  ...
```

Запускаємо всі контейнери за допомогою команди:

```
```bash
docker-compose up -d
```
```

Це дозволить одночасно запуснути всі необхідні сервіси для кожного тенанта.

Після того як контейнери будуть розгорнуті, потрібно інтегрувати їх з іншими компонентами системи, такими як бази даних, зовнішні API та сервіси управління безпекою.

Ці кроки дозволяють створити та розгорнути додатки в контейнерах Docker, налаштувати їх для роботи в мультитенантному середовищі та забезпечити ізоляцію даних між тенантами. Після цього можемо використовувати Docker Compose для зручного управління контейнерами та інтеграції з іншими сервісами.

### 4.3 Створення та налаштування Kubernetes-кластера

Створення кластера (хмарний варіант): Наприклад, для AWS EKS:

```
eksctl create cluster --name my-multitenant-cluster --region us-west-2 --nodes 3
```

Це команда для створення кластера з трьома вузлами.

Налаштування доступу. Після створення кластера, вам потрібно налаштувати `kubectl` — команду для взаємодії з Kubernetes-кластером:

```
aws eks --region us-west-2 update-kubeconfig --name my-multitenant-cluster
```

#### 4.3.1 Розгортання додатків у Kubernetes

Створення Deployment файлів. Для кожного додатка потрібно створити YAML-файли, що описують налаштування розгортання додатків. Наприклад, для розгортання додатка у кількох контейнерах з підтримкою балансування навантаження та масштабування:

```
deployment.yaml:  
  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: tenant1-app  
spec:  
  replicas: 3  
  selector:  
    matchLabels:  
      app: tenant1-app  
  template:  
    metadata:  
      labels:  
        app: tenant1-app  
    spec:  
      containers:  
        - name: tenant1-app-container
```

```
image: myapp:latest
ports:
  - containerPort: 8080
```

Це розгортає три репліки додатка для одного тенанта. Процес повторюється для інших тенантів.

Створення сервісів для доступу до додатків. Створіть файл конфігурації сервісу, щоб забезпечити доступ до додатків через балансування навантаження:

```
service.yaml:
apiVersion: v1
kind: Service
metadata:
  name: tenant1-service
spec:
  selector:
    app: tenant1-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
  type: LoadBalancer
```

Це створить балансувальник навантаження, який автоматично розподілятиме трафік на репліки додатка для кожного тенанта.

Запуск додатків у Kubernetes. Використовують команди `kubectl` для деплою додатків:

```
kubectl apply -f deployment.yaml
kubectl apply -f service.yaml
```

### 4.3.2 Налаштування масштабування додатків

Kubernetes дозволяє автоматично масштабувати додатки залежно від навантаження (автоскейлінг):

Горизонтальне масштабування (Horizontal Pod Autoscaler): Створіть файл для налаштування автоскейлінгу, наприклад:

```
hpa.yaml:  
apiVersion: autoscaling/v1  
kind: HorizontalPodAutoscaler  
metadata:  
  name: tenant1-app-hpa  
spec:  
  scaleTargetRef:  
    apiVersion: apps/v1  
    kind: Deployment  
    name: tenant1-app  
  minReplicas: 3  
  maxReplicas: 10  
  targetCPUUtilizationPercentage: 50
```

Цей файл налаштовує автоскейлінг додатка в залежності від завантаження CPU. Kubernetes збільшить кількість реплік до 10, якщо навантаження перевищить 50%.

Запуск HPA:

```
kubectl apply -f hpa.yaml
```

#### 4.4 Забезпечення відмовостійкості та автоматичне відновлення

Моніторинг і управління станом додатків. Kubernetes автоматично відстежує стан кожного контейнера та перевіряє його працездатність. Якщо контейнер виходить з ладу, Kubernetes автоматично його перезапускає. Це налаштовується через "проби" (Liveness та Readiness):

Прості Liveness та Readiness:

```
livenessProbe:  
  httpGet:  
    path: /health  
    port: 8080  
  initialDelaySeconds: 5  
  periodSeconds: 10  
readinessProbe:  
  httpGet:  
    path: /ready  
    port: 8080  
  initialDelaySeconds: 5  
  periodSeconds: 10
```

Це налаштування дозволяє Kubernetes перевіряти, чи додаток готовий приймати запити, та чи працює він коректно.

Якщо контейнер перестає працювати, Kubernetes автоматично перезапустить його і розподілить навантаження на інші репліки, забезпечуючи безперебійну роботу.

#### 4.5 Налаштування безпеки та ізоляції

Використовується namespace для ізоляції середовищ різних тенантів. Це дозволить ізолювати їх додатки та дані:

```
kubectl create namespace tenant1
```

Ролі та права доступу (Role-Based Access Control, RBAC): Налаштуйте політики доступу для обмеження прав кожного тенанта лише на їхні ресурси:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: tenant1
  name: tenant1-role
rules:
- apiGroups: [""]
  resources: ["pods", "services"]
  verbs: ["get", "list", "watch"]
```

Це дає тенанту доступ тільки до їхніх власних сервісів і подів.

Оркестрація контейнерів за допомогою Kubernetes дозволяє автоматизувати управління мультитенантними додатками, забезпечуючи високу продуктивність, масштабованість та відмовостійкість. За допомогою Kubernetes можна динамічно адаптувати систему до змінних умов навантаження, забезпечувати ізоляцію між тенантами і швидке відновлення після збоїв.

#### 4.6 Інтеграція контейнеризованих додатків з хмарними сервісами

Інтеграція контейнеризованих додатків з хмарними сервісами надає можливість використовувати масштабовану інфраструктуру для обробки даних, аналізу та зберігання. Для кожної хмарної платформи існують конкретні сервіси, які спрощують цей процес. Ось як можна інтегрувати додатки з основними хмарними платформами:

a) Amazon Web Services (AWS). Amazon RDS (Relational Database Service): AWS RDS забезпечує керувані бази даних, які можуть бути інтегровані з додатками в Kubernetes. Наприклад, ви можете підключити контейнеризований додаток до бази даних RDS за допомогою змінних середовища.

У Kubernetes використовуйте Secrets для зберігання конфіденційних даних, таких як облікові дані бази даних:

```
apiVersion: v1
kind: Secret
metadata:
  name: rds-credentials
type: Opaque
data:
  username: base64encoded-username
  password: base64encoded-password
```

Потім підключіть ці дані до пода:

```
env:
- name: DB_USERNAME
  valueFrom:
    secretKeyRef:
      name: rds-credentials
      key: username
- name: DB_PASSWORD
  valueFrom:
    secretKeyRef:
      name: rds-credentials
      key: password
```

S3 (Simple Storage Service). Для зберігання великих даних або файлів можна використовувати S3. Інтеграція з додатками може бути реалізована через SDK для обраної мови програмування (AWS SDK), або за допомогою S3 API в контейнерах. Для доступу додатків до S3 через Kubernetes ви можете налаштувати IAM роль з відповідними дозволами. У випадку AWS EKS, ви можете використовувати IAM Roles for Service Accounts (IRSA) для надання доступу до ресурсів S3 конкретним подам.

б) Microsoft Azure. Azure Blob Storage: Якщо ваш додаток потребує зберігання файлів або об'єктів, використовуйте Azure Blob Storage. Інтеграція з Kubernetes можлива через підключення контейнеризованих додатків до Blob Storage через Azure SDK або REST API. Приклад використання змінних середовища для налаштування доступу до Blob Storage:

```
env:
  - name: AZURE_STORAGE_ACCOUNT
    value: my-storage-account
  - name: AZURE_STORAGE_ACCESS_KEY
    valueFrom:
      secretKeyRef:
        name: azure-secrets
        key: storage-access-key
```

Azure SQL Database. Інтеграція може бути реалізована через підключення додатків до бази за допомогою керованого сервісу.

в) Google Cloud Platform (GCP). Google Cloud Storage пропонує сховище об'єктів, яке можна інтегрувати з додатками через Google Cloud SDK або API. Ви можете зберігати конфігурацію доступу через Kubernetes Secrets. BigQuery: Для аналітики та обробки великих даних використовуйте BigQuery, який підтримує інтеграцію через API для обробки і аналізу даних у масштабі.

## 4.7 Забезпечення безпеки та управління доступом

### Шифрування даних у Kubernetes:

- шифрування даних у стані спокою. Більшість хмарних платформ, таких як AWS, Azure та GCP, забезпечують автоматичне шифрування даних у стані спокою для таких сервісів, як RDS, Blob Storage чи S3. Крім того, можна вручну налаштувати шифрування для конфіденційних даних у Kubernetes, використовуючи Secrets для зберігання облікових даних або ключів шифрування;

- шифрування даних при передачі. Використовуйте HTTPS або TLS для захисту даних при передачі між подами та зовнішніми сервісами. У Kubernetes можна налаштувати автоматичний SSL за допомогою інструментів на кшталт cert-manager, який інтегрується з Let's Encrypt.

### Управління доступом за допомогою AWS IAM, Azure AD або GCP IAM:

- AWS IAM (для EKS): AWS Identity and Access Management (IAM) дозволяє створювати ролі з точним контролем доступу до ресурсів. Ви можете використовувати IAM Roles for Service Accounts для надання ролей конкретним подам у кластері Kubernetes:

```
eksctl create iamserviceaccount \
  --name my-service-account \
  --namespace my-namespace \
  --cluster my-cluster \
  --attach-policy-arn arn:aws:iam::aws:policy/AmazonS3FullAccess \
  --approve
```

- Azure Active Directory (для AKS). Azure AD дозволяє інтегрувати Kubernetes з Azure AD для управління правами доступу та авторизації. Ви можете налаштувати Azure AD Pod Identity, щоб надавати подам доступ до конкретних ресурсів, таких як SQL Database або Blob Storage;

- GCP IAM (для GKE): Google Cloud IAM дозволяє налаштувати ролі для подів у Google Kubernetes Engine (GKE). Ви можете використовувати Workload Identity для безпечного доступу до ресурсів GCP без необхідності використовувати ключі API.

Role-Based Access Control (RBAC) у Kubernetes. RBAC дозволяє керувати доступом до ресурсів Kubernetes для користувачів і подів:

- створення ролі для подів: Можна налаштувати ролі, які контролюють доступ до ресурсів Kubernetes. Наприклад, роль, яка дозволяє лише читати конфігурацію подів:

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: my-namespace
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list"]
```

Прив'язка ролей. Прив'язка ролі до конкретного користувача або сервісного облікового запису:

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: read-pods
  namespace: my-namespace
subjects:
- kind: User
  name: jane
```

```

  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io

```

### Використання мережевої безпеки (Network Policies):

- мережеві політики (Network Policies) дозволяють контролювати потоки трафіку між подами та зовнішніми ресурсами в кластері Kubernetes. Ви можете використовувати Network Policies для обмеження доступу між подами, забезпечуючи кращу безпеку даних.

```

apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-specific-pod
  namespace: my-namespace
spec:
  podSelector:
    matchLabels:
      app: myapp
  policyTypes:
    - Ingress
  ingress:
    - from:
      - podSelector:
          matchLabels:
            app: allowed-app
  ports:
    - protocol: TCP
      port: 80

```

Це обмежує доступ до подів додатка лише з певних інших подів або ресурсів.

Інтеграція контейнеризованих додатків із хмарними сервісами, такими як AWS, Azure або GCP, забезпечує доступ до масштабованих і керованих ресурсів для обробки, зберігання та аналізу даних. Забезпечення безпеки додатків у Kubernetes вимагає використання шифрування, IAM для керування доступом і RBAC для контролю прав користувачів. Ці інструменти допоможуть вам побудувати надійну та захищену систему, здатну ефективно функціонувати в мультитенантному середовищі.

#### 4.8 Методика тестування мультитенантної системи на відповідність вимогам інтелектуальних безперервних виробництв

Тестування мультитенантної системи для інтелектуальних безперервних виробництв охоплювало кілька етапів для перевірки її продуктивності, надійності та масштабованості. Основна мета полягала в тому, щоб оцінити, як система впорається з одночасною обробкою запитів від різних клієнтів (тенантів), забезпечуючи при цьому стабільну роботу та необхідний рівень ізоляції даних.

Методика тестування включала наступні етапи:

- підготовка середовища тестування. Система була розгорнута у хмарному середовищі з використанням контейнеризації (Docker) і оркестрації (Kubernetes). Для кожного тестування була налаштована відповідна кількість клієнтів (тенантів) із різним рівнем навантаження на систему. Тести проводилися з інтеграцією до різних хмарних платформ, таких як AWS, Azure та GCP, для оцінки ефективності кожної платформи:

- моделювання навантаження. Для моделювання роботи кількох тенантів використовувались інструменти для генерації навантаження, зокрема Apache JMeter і Locust, які симулювали запити користувачів різного типу (запити на отримання даних, запис нових даних, виконання обчислень). Було

протестовано кілька сценаріїв: від низького рівня навантаження (10-20 тенантів) до високого (понад 200 одночасних клієнтів);

- моніторинг та збір даних. Система моніторингу, реалізована за допомогою Prometheus і Grafana, дозволяла збирати реальні дані щодо використання ресурсів, часу відгуку системи, продуктивності баз даних та пропускної здатності при змінних навантаженнях. Зібрані метрики включали:

- а) час відгуку на запити (response time);
- б) використання ресурсів (CPU, пам'ять);
- в) пропускну здатність (кількість оброблених запитів за секунду);
- г) надійність (кількість помилок і відновлень у разі збоїв);

- тестування надійності. Для перевірки надійності система піддавалася тестам на відмовостійкість, де навмисно створювалися збої (наприклад, відключення вузлів або перевантаження одного з компонентів). Це дозволило перевірити, як Kubernetes автоматично перерозподіляє навантаження, відновлює пошкоджені вузли і підтримує роботу тенантів;

- оцінка масштабованості. Було проведено серію тестів для оцінки можливості автоматичного масштабування системи в залежності від навантаження. Для цього використовувався горизонтальний автоскейлер (Horizontal Pod Autoscaler) Kubernetes, який автоматично збільшував або зменшував кількість подів, що обробляють запити, відповідно до поточного навантаження.

Тести показали, що система здатна ефективно обробляти запити навіть при значному навантаженні. При 200 одночасних тенантах середній час відгуку збільшився з 0,1 сек до 0,35 сек, що є прийнятним для більшості виробничих процесів. Пропускна здатність системи знизилася зі 1000 запитів/сек до 600 запитів/сек при максимальному навантаженні, але загальна продуктивність залишалася стабільною, без суттєвих затримок.

Система показала високу стійкість до збоїв. При моделюванні відключення одного з вузлів Kubernetes автоматично перерозподіляв

навантаження між іншими вузлами без втрати даних або зупинки роботи клієнтів. Середній час відновлення вузла склав 30 секунд, після чого система продовжувала працювати у штатному режимі.

Тести показали, що система здатна автоматично масштабуватися залежно від рівня навантаження. При перевищенні 70% використання CPU було автоматично додано нові поди, що дозволило збалансувати навантаження і зберегти стабільну продуктивність системи. Зменшення навантаження призводило до автоматичного скорочення кількості подів, що забезпечує ефективне використання ресурсів.

Отримані результати свідчать про те, що мультитенантна архітектура, побудована на основі контейнеризації та оркестрації, є ефективною для інтелектуальних безперервних виробництв. Продуктивність системи залишається стабільною навіть при значних навантаженнях, що робить її придатною для роботи з великими обсягами даних і великою кількістю клієнтів. Висока надійність системи підтверджується її здатністю швидко відновлюватися після збоїв і перерозподіляти навантаження між вузлами. Автоматичне масштабування дозволяє ефективно використовувати ресурси і підтримувати оптимальну продуктивність незалежно від змін у навантаженні.

Результати тестування демонструють, що запропонована архітектура відповідає вимогам інтелектуальних виробничих процесів і може бути рекомендована для впровадження в різних галузях, де необхідна висока гнучкість, безпека та надійність хмарних рішень.

#### 4.9 Порівняльний аналіз хмарних платформ для інтеграції інформаційної системи з підтримкою мультитенантності

На рис. 4. 1 наведено діаграму, яка візуалізує основні характеристики трьох провідних хмарних платформ – Amazon Web Services (AWS), Microsoft Azure та Google Cloud Platform (GCP). Порівняння охоплює такі аспекти, як:

- масштабованість. Всі три платформи пропонують високий рівень масштабованості, але AWS має найбільшу глобальну інфраструктуру;
- безпека. Всі платформи пропонують потужні засоби управління безпекою, зокрема, IAM (Identity and Access Management) у AWS, Azure Active Directory у Azure, та Google IAM у GCP;

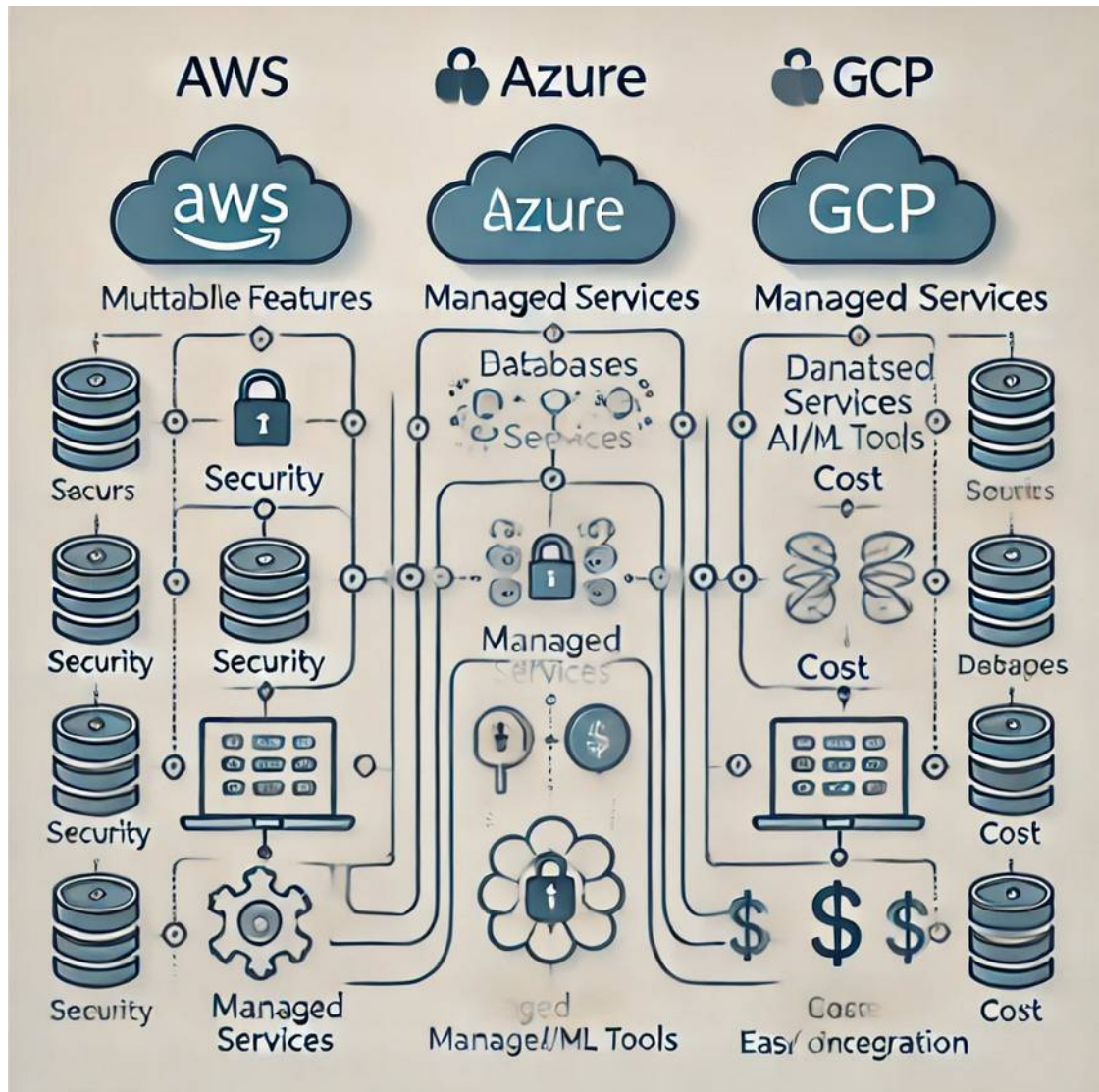


Рис. 4.1 – Основні характеристики обраних хмарних платформ

- вартість. Ціноутворення залежить від вимог до ресурсів, однак AWS і Azure можуть бути дорогими для довгострокового використання

великих обсягів даних, тоді як GCP пропонує конкурентні ціни, особливо для аналітики та машинного навчання;

- керовані сервіси. AWS та Azure надають широкий спектр керованих сервісів для корпоративних користувачів, тоді як GCP лідирує в галузі аналітики та машинного навчання.

У табл. 4.1 наведені сильні сторони та недоліки обраних хмарних платформ.

Таблиця 4.1 - Порівняльна таблиця обраних хмарних платформ.

| Хмарні платформи            | Сильні сторони   | Недоліки   |
|-----------------------------|--|--|
| Amazon Services (AWS)       | WebМасштабованість<br>Керовані сервіси<br>Безпека<br>Вартість  | Ціноутворення<br>Складність налаштування<br>Перевантаження інтерфейсу                          |
| Microsoft Azure             | Інтеграція з корпоративними рішеннями<br>Широкий набір інструментів для розробників<br>Машинне навчання<br>Безпека | Надійність і простота<br>Ціноутворення<br>Крива навчання                                       |
| Google Cloud Platform (GCP) | Аналітика та великі дані<br>Інтеграція з AI/ML<br>Масштабованість<br>Ціноутворення                                 | Обмежена кількість сервісів<br>Менше присутності на ринку<br>Ціноутворення на хостинг і мережу |

На основі діаграми та таблиці можна зробити наступні висновки. AWS – найкращий вибір для масштабних рішень з великим навантаженням, які потребують глобального охоплення та великої кількості керованих сервісів. AWS забезпечує найширший набір сервісів, але складність налаштування і високі витрати можуть стати суттєвим бар'єром для невеликих команд.

Azure ідеально підходить для інтеграції з існуючими корпоративними інструментами Microsoft та забезпечує високу безпеку та зручність для великих компаній. Azure добре інтегрується з корпоративними рішеннями Microsoft, але його надійність і складність управління можуть створювати проблеми для критично важливих додатків.

GCP – оптимальний варіант для аналітики даних та інтелектуальних проєктів, що вимагають високої автоматизації й можливостей машинного навчання. GCP спеціалізується на аналітиці й AI, але обмежений вибір сервісів і менш популярна підтримка на ринку може бути недоліком для великих проєктів.

## ВИСНОВКИ

У даній роботі було розглянуто процес розробки хмарної інформаційної системи з підтримкою мультитенантності, починаючи з аналізу основних компонентів системи і закінчуючи інтеграцією контейнеризованих додатків із хмарними сервісами та налаштуванням безпеки.

Проведене дослідження з розробки мультитенантної хмарної інформаційної системи для інтелектуальних виробництв підтвердило ефективність використання контейнеризації та оркестрації Docker та Kubernetes для забезпечення гнучкості, масштабованості та безпеки систем. Основним досягненням стало створення архітектури, що забезпечує ізоляцію даних клієнтів (тенантів) при спільному використанні інфраструктури, що суттєво знижує витрати на підтримку і управління системою. Інтеграція з хмарними сервісами AWS, Azure, GCP дала можливість застосовувати масштабовані обчислювальні потужності та забезпечити надійність роботи при різних рівнях навантаження.

Внесок даного дослідження полягає у розвитку підходів до побудови мультитенантних хмарних систем, зокрема для інтелектуальних виробництв. Запропонована архітектура вирішує проблеми ізоляції даних, автоматичного масштабування та відмовостійкості, що робить її придатною для застосування у виробничих процесах з високими вимогами до продуктивності та безпеки. Крім того, використання AI/ML сервісів дозволяє підвищити ефективність управління виробничими процесами за рахунок аналізу даних і автоматизації прийняття рішень.

Подальший розвиток цієї теми можна здійснити у кількох напрямках. Перш за все, слід зосередитися на оптимізації механізмів автоматичного масштабування для обчислювальних ресурсів, що дозволить підвищити ефективність роботи системи під час пікових навантажень. Додатково варто

дослідити питання оптимального розподілу ресурсів між тенантами для уникнення "шумних сусідів" у загальному середовищі, що може впливати на продуктивність системи. Також важливо поглибити інтеграцію з сервісами штучного інтелекту та машинного навчання для більш інтелектуальної автоматизації виробничих процесів.

Одним з напрямків покращення мультитенантних рішень у контексті інтелектуальних виробництв може стати розвиток інструментів моніторингу та аналізу стану системи в реальному часі. Це дозволить оперативно виявляти потенційні проблеми з продуктивністю або безпекою і автоматично адаптувати ресурси під змінні умови. Також варто розглянути можливість використання гібридних хмарних рішень, де частина критично важливих обчислень буде виконуватися на приватних серверах, що підвищить безпеку та забезпечить додаткову гнучкість. Важливим кроком є розвиток механізмів самовідновлення та автоматичного відновлення системи у разі збоїв або втрати частини даних, що мінімізує можливі простої у виробничих процесах.

Результати роботи буди представлені та апробовані на XVII Всеукраїнській науково-практичній WEB конференції аспірантів, студентів та молодих вчених «Комп'ютерні інтелектуальні системи та мережі», та на XIV Всеукраїнській конференції молодих вчених «Молоді вчені 2024 – від теорії до практики» та на V Всеукраїнській науково-технічній конференції молодих вчених, аспірантів і студентів «Комп'ютерні ігри та мультимедіа як інноваційний підхід до комунікації – 2024».

## ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Сердюк Н.М., Вольгуст М.С., Трипольєв О. Використання можливостей Cloud Computing при обробці інформації у безперервних виробництвах: тези доповідей XIV Всеукр. наук.-практ. конф., Дніпро, 21 березня 2024 р. – С. 177.
2. Trypoliev O., Volhust M., Serdiuk N. Features of cloud computing in big data processing: матеріали XVII Всеукр. наук.-практ. веб-конф., Кривий Ріг, 26–28 березня 2024 р. – 283 с. – С. 28.
3. Сердюк Н.М., Трипольєв О.В. Особливості розробки архітектури хмарної інформаційної системи з підтримкою мультитенантності. Комп'ютерні ігри і мультимедіа як інноваційний підхід до комунікації: тези доповідей IV Всеукр. наук.-практ. конф., Одеса, 2024.
4. Губенко А.П., Іванов І.О. Хмарні обчислення у сучасних виробничих процесах // Вісник ХНУРЕ. – 2023. – №5. – С. 45–52.
5. Brown P., Smith J. Multitenancy Architecture for Cloud Applications. – New York: Springer, 2021. – 250 p.
6. Коваленко С.М. Технології контейнеризації для автоматизації виробництва // Інформаційні технології в управлінні. – 2022. – Т. 12. – №3. – С. 85–92.
7. Алексєєв Ю.В., Бойко Л.В. Принципи безпеки даних у мультитенантних системах // Кібербезпека в сучасному світі. – 2022. – Т. 15. – №2. – С. 30–38.
8. Microsoft Azure Documentation Multitenancy and Data Isolation Guidelines for Cloud Computing [Електронний ресурс]. – Режим доступу: <https://azure.microsoft.com> (дата звернення: 05.01.2025).

9. Amazon Web Services AWS Well-Architected Framework [Электронный ресурс]. – Режим доступа: <https://aws.amazon.com> (дата звернения: 24.12.2024).

10. Google Cloud Best Practices for Multitenant Architectures [Электронный ресурс]. – Режим доступа: <https://cloud.google.com> (дата звернения: 05.01.2025).

11. Red Hat Introduction to Kubernetes for Multitenant Applications [Электронный ресурс]. – Режим доступа: <https://www.redhat.com> (дата звернения: 24.12.2024).

12. Docker Best Practices for Containerized Multitenancy [Электронный ресурс]. – Режим доступа: <https://www.docker.com> (дата звернения: 02.01.2025).

13. IBM Cloud Multitenancy in Cloud Computing: Solutions and Approaches [Электронный ресурс]. – Режим доступа: <https://www.ibm.com/cloud> (дата звернения: 02.01.2025).