МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Аналіз технологій контейнеризації та оптимізація розгортання масштабованого додатку на платформі Kubernetes»

Завідувач
випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Кузіков Б.О.**

Студентки групи ІН.Мз – 91 с **Бойко Г.О.**

Нормоконтроль **Проценко О.Б.**

СУМИ 2020

Сумський державний університет

(назва вузу)

Факультет *ЦЗДВФН* Кафедра *Комп'ютерних наук*

Спеціальність *«Інформатика»*

Затверджую:

зав.кафедрою _____

_____

"_____" _____ 20___р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Бойко Ганні Олександрівні

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Аналіз технологій контейнеризації та оптимізація розгортання масштабованого додатку на платформі Kubernetes*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи)
_____

3. Вхідні данні до проекту (роботи) _____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Формулювання проблеми, постановка задачі дослідження. 2) Аналіз сучасного стану хмарних технологій. 3) Огляд технологій віртуалізації та контейнеризації. 4) Огляд хмарних платформ для розгортання додатків. 5) Розробка*

оптимізованої схеми розгортання та масштабування тестового додатку на базі платформи Azure Kubernetes 6) Розроблення рекомендацій до вибору методу розгортання додатку

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання Видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____

(підпис)

Завдання прийняв до виконання _____

(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
	Формулювання проблеми, постановка задачі дослідження		
	Аналіз сучасного стану хмарних технологій, віртуалізації та контейнеризації		
	Огляд хмарних платформ для розгортання додатків		
	Розгортання та масштабування тестового додатку на базі платформи Azure Kubernetes		
	Оформлення пояснювальної записки до дипломної роботи		

Студент – дипломник _____

(підпис)

Керівник проекту _____

(підпис)

РЕФЕРАТ

Записка: 62 стор., 15 рис., 2 табл., 1 додаток, 22 джерел.

Об'єкт дослідження — технології кластеризації контейнерів Kubernetes;

Мета роботи – аналіз технологій кластеризації та розгортання контейнерного додатку, оптимізація схеми розгортання дадатку на базі платформи Kubernetes в серидовищі Azure та його автомасштабування.

Методи дослідження – аналіз, порівняння, функціональних випробувань.

Результати – Проаналізовано деякі можливості що забезпечуються хмарними технологіями; Порівняно підхід для реалізації однієї з таких систем Kubernetes – найбільшими постачальниками хмарних послуг; Розроблено оптимізовану схему роботи з контейнерними додатками, та розглянути механізм розгортання одатку на платформі оркестрації контейнерів.

Ключові слова: хмара, кластеризація, контейнеризація, масштабування, Docker, Kubernetes.

Зміст

ВСТУП

Основними методами загальнонаукового пізнання є аналіз і синтез. Ці методи також актуальні в сучасних інформаційних технологіях. Зокрема, можна виділити тенденцію наслідування основам цих принципів: величезні інформаційні структури розкладаються на окремі маленькі компоненти, такі як мікросервіси, що є типовим аналогом аналізу; а потім ці мікропроцеси за допомогою високофункціональних, автоматизованих систем оркетсровки синтезуються в повномасштабні глобальні програмні продукти.

Та можливо, це чи не єдина паралель, яку можна провести між античністю і сучасними технологіями. Обєми інформаційних даних зараз настільки значні, що впоратися з ними підсилу тільки таким же масштабним за функіональністю програмним рішнням для організації і зберігання.

Хмарні технології, віртуалізація, Docker – контейнеризація, сервіс-орієнтоване проектування покликані допомогти розробникам зосередитися на самому процсі проектуваняя системи, не відволікаючись на тісно повязані проблеми масштабування, встановлення залежностей і оновлення додатків. Проте є самі по собі досить складними і багатофункціональними.

Тому в даній роботі ми розглянемо концепції що дозволяють впроваджувати та підтримувати так актуальні сьогодні мікропроцеси та хмарні технології. Порівняємо механізми віртуалізації та контейнеризації. Розглянемо платформи, і їх можливості, що дозволяють підтримувати хмарні обчислення. Та здійснемо оптимізовану реалізацію процесу розгортання додатку на основі платформи Азур.

Актуальність роботи полягає у дослідженні сучасних технологій хмарних обчислень, методів віртуалізації і контейнеризації; розробці оптимізованого розгортання контейнерного додатку на платформі Azure Kubernetes та порівнянні з іншими методами розгортання додатку.

Обєкт дослідження – технології кластеризації контейнерів Kubernetes;

Предмет дослідження – платформа хмарних технологій – Azure; Kubernrtes кластер з набором контейнерів Docker;

Мета: аналіз технологій розгортання контейнерного додатку, оптимізація схеми розгортання дадатку на базі платформи Kubernetes в серидовищі Azure;

Для досягнення мети сформульовані наступні завдання роботи:

1. Проаналізувати основні переваги і тенденції методів хмарних обчислень
2. Дослідтит механізм роботи платформи Kubernetes всередовищі Азур
3. Розгорнути оптимізований кластер Kubernetes за допомогою системи конфігурування Cloud Shell
4. Порівняти розгортання додатків на платформі Kubernetes та за допомогою віртуального хостингу
5. Сформулювати рекомендації до вибору методу розгортання додатку

Наукова новизна отриманих результатів полягає у вирішенні науковопрактичної задачі аналізу механізму рогортання кластеру Kubernetes з запущеними в ньому Docker контейнерами, порівнянні механізмів запуску додатків

РОЗДІЛ 1. МЕТОДИ І ПРИНЦИПИ ХМАРНИХ ОБЧИСЛЕНЬ

1.1 Поняття хмарних обчислень

Хмарні обчислення перетворюють IT-інфраструктуру на утиліту, що дозволяє «підключатись» до інфраструктури через Інтернет та використовувати обчислювальні ресурси, не встановлюючи та не підтримуючи їх локально.

Хмарні обчислення - це доступ в мережі інтернет за запитом до обчислювальних ресурсів - додатків, серверів (фізичних серверів та віртуальних серверів), сховища даних, засобів розробки, можливостей мереж та іншого - розміщених у віддаленому центрі обробки даних, керованому хмарними службами (cloud services provider). CSP надає бізнес користувачу такі ресурси за абонентську плату або виставляє рахунки відповідно до використання.

Термін «хмарні обчислення» також позначає технологію, яка змушує хмарні роботи. Сюди входить певна форма віртуалізованої IT-інфраструктури - сервери, програмне забезпечення операційної системи, мережа та інша інфраструктура, яка абстрагована за допомогою спеціального програмного забезпечення, так що її можна об’єднати та розділити незалежно від фізичних меж обладнання. Наприклад, один апаратний сервер можна розділити на кілька віртуальних серверів.

Віртуалізація дозволяє хмарним провайдерам максимально використовувати ресурси своїх центрів обробки даних. Не дивно, що багато корпорацій застосували хмарну модель доставки для своєї локальної інфраструктури, щоб вони могли реалізувати максимальне використання та економію витрат порівняно з традиційною IT-інфраструктурою та пропонувати такі ж самообслуговування та спритність своїм кінцевим споживачам.

Хмарні обчислення (англ. Cloud Computing) – модель забезпечення повсюдного та зручного доступу до спільного пулу обчислювальних та інформаційних ресурсів через інтернет мережу на вимогу, що підлягають налаштуванню (наприклад, до серверів, комунікаційних мереж, сервісів та прикладних програм), і які можуть бути оперативно надані з мінімальним зверненням до провайдера та управлінськими затратами.

Програмне забезпечення надається кінцевому користувачеві як Інтернет-сервіс зі зверненням до хмарних обчислень. Так користувач має повний, захищений доступ до власних даних та операційних систем, але не має можливості управляти інфраструктурою і не повинен піклуватися про фізичні ресурсина з якими працює система. Так метафорично, «Хмарою» називають мережу, що приховує всі технічні деталі. Згідно з документом IEEE, від 2008 року, «Хмарні обчислення — це парадигма, в рамках якої інформація постійно зберігається на серверах у мережі інтернет і тимчасово кешується на клієнтській стороні, зокрема на персональних комп'ютерах, ноутбуках, ігрових приставках, смартфонах тощо».[21]

Хмарні технології надають в орендують ресурси: простір для зберігання даних або потужність процесора, на комп’ютерах іншої компанії. Компанію, що надає ці послуги, називають хмарним провайдером. Деякі приклади постачальників - це Microsoft, Amazon та Google.

Хмарний постачальник відповідає за фізичне обладнання, необхідне для виконання роботи, і за його оновлення. Пропоновані обчислювальні послуги, як правило, відрізняються залежно від постачальника хмарних послуг. Однак зазвичай вони включають: обчислювальну потужність, зберігання - файлів та баз даних, послуги

корпоративної мережі - безпечні зв’язки між хмарним провайдером та компанією, аналітику - візуалізація даних телеметрії та продуктивності

1.2. Основні послуги хмарних обчислень та їх моделі

Мета хмарних обчислень полягає в тому, щоб зробити бізнес простішим та ефективнішим, будь то малий стартап чи велике підприємство, оскільки кожен бізнес унікальний і має різні потреби. Для задоволення цих потреб постачальники хмарних обчислень пропонують широкий спектр послуг.

Найбільш вагомою послугою є надання обчислювальної потужності. Як споживач, ми всі залежамо від обчислювальних послуг, що надаються різними хмарними провайдерами, що складають інтернет, під час надсилання повідомлення, перегляду відео, оплати рахунків в Інтернеті, хмарні сервери, обробляють кожен запит і повертають відповідь.

1.2.1 Віртуалізація, контейнеризація та безсерверність

Під час створення програмного продукту за допомогою хмарних технологій, можна обрати спосіб виконання роботи на основі потреб та ресурсів. Наприклад, за необхідності мати більше контролю та відповідальності за обслуговування, можна створити віртуальну машину (VM - це емуляція комп’ютера, подібно до робочого столу чи ноутбука). Кожна віртуальна машина включає операційну систему та апаратне забезпечення, яке видається користувачеві як фізичний комп'ютер під управлінням Windows або Linux. Потім можна встановити будь-яке

необхідне для виконання завдань програмне забезпечення, що буде запускатися в хмарі.

Головна перевага хмари в тому, що користувачу непотрібно купувати будь-яке обладнання та встановлювати ОС. Хмарний провайдер запускає віртуальну машину на фізичному сервері в одному з центрів обробки даних - часто розділяючи цей сервер з іншими віртуальними машинами (ізольованими та захищеними). За допомогою хмари можна розгорнути віртуальну машину і мати її готовою до роботи за лічені хвилини з меншими витратами, ніж налаштування фізичного комп'ютера.

Віртуальні машини не єдиний спосіб надання обчислювальної потужності та дискового протору - є ще два популярних варіанти рішення: контейнери та безсерверні обчислення.

Контейнери забезпечують послідовне, ізольоване середовище для виконання програм. Вони схожі на віртуальні машини, за винятком того, що не потребують гостьової операційної системи. Натомість додаток та всі його залежності упаковуються у "контейнер", а потім для виконання програми використовується стандартне середовище виконання. Це дозволяє запустити контейнер всього за кілька секунд, оскільки немає ОС для завантаження та ініціалізації.

Однією з провідних платформ для управління контейнерами є Docker. Контейнери Docker забезпечують ефективний, легкий підхід до розгортання додатків, оскільки вони дозволяють незалежно розгортати різні компоненти програми в різних контейнерах. На одній машині можна запускати кілька контейнерів, а контейнери можна переміщати між машинами. [2] Переносимість контейнера полегшує розгортання додатків у кількох середовищах, як локально, так і у хмарі, часто без змін у програмі.

Безсерверні обчислення дозволяють запускати код програми без створення, налаштування або обслуговування сервера. Основна ідея полягає в тому, що додаток розбивається на окремі функції, які запускаються при активації якоїсь дії. Це ідеально підходить для автоматизованих завдань - наприклад, можна створити безсерверний процес, який автоматично надсилає підтвердження електронною поштою після того, як клієнт здійснить онлайн-покупку.

Безсерверна модель відрізняється від віртуальних машин та контейнерів тим, що для неї вагомим є лише час обробки, який використовується кожною функцією під час її виконання, а не вес час, як для віртуальної машини чи контейнера. Ця архітектура працює не для кожної програми, але коли логіку програми можна розділити на незалежні блоки, можна протестувати їх окремо, оновити окремо та запустити за мікросекунди, то цей підхід стає найшвидшим варіантом розгортання. Схематично, зображено порівняння трьох обчислювальних підходів, які ми розглянули.

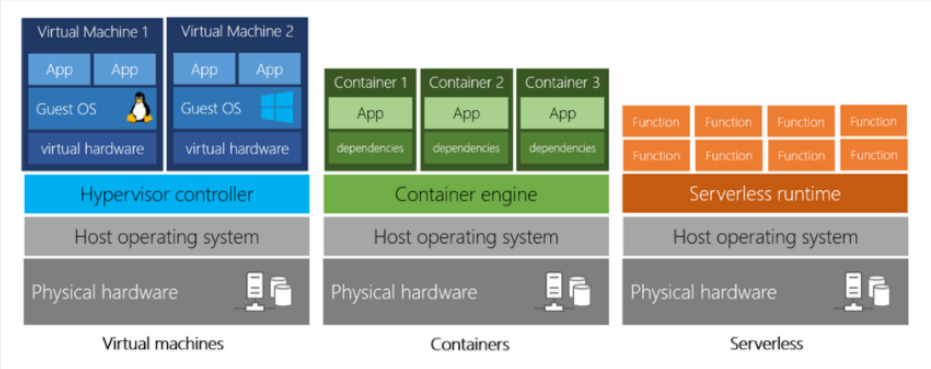

Рисунок 1.1 Порівняння моделей віртуалізації, контейнеризації та безсерверних технологій [16]

Більшість програм чи пристроїв читають та/ або записують дані. Тип даних та спосіб їх зберігання можуть бути різними. Хмарні

провайдери зазвичай пропонують послуги, які можуть обробляти дані незалежно від їх типів і зберігати такі дані в хмарі у вигляді файлу чи запису бази даних. Перевагою використання хмарного сховища даних є можливість масштабування дискового простору відповідно до потреб. У деяких випадках сховище може навіть розширюватися і скорочуватися автоматично - отже, користувач сплачує лише за те, що потрібно в будь-який момент часу.

1.2.2. Моделі розгортання хмари

Існує три різні моделі розгортання хмари. Модель хмарного розгортання визначає, де зберігаються дані та як клієнти взаємодіють з ними і де працюють додатки? Це також залежить від того, якою частиною інфраструктури потрібно керувати.

Публічна хмара - це найпоширеніша модель розгортання. У цьому випадку немає місцевого обладнання, яким можна керувати чи оновлювати його - все працює на апаратному забезпеченні хмарного постачальника. У деяких випадках можна заощадити додаткові витрати, поділившись обчислювальними ресурсами з іншими хмарними користувачами.

Підприємства можуть використовувати декілька публічних хмарних провайдерів різного масштабу. Microsoft Azure, Amazon, Google Cloud є прикладом публічного хмарного провайдера.

Переваги

Висока масштабованість - для масштабування не потрібно купувати новий сервер

Оплата в міру використання, без витрат CapEx

Користувачу немає необхідності слідкувати за технічним обслуговуванням або оновлення обладнання

Мінімальні технічні знання для налаштування та використання - можна використовувати навички та досвід хмарного провайдера, щоб забезпечити безпеку та доступність робочих навантажень

Поширеним сценарієм використання є розгортання веб-програми або веб-сайту блогу на апаратному забезпеченні та ресурсах, які належать хмарному постачальнику. Використання загальнодоступної хмари в цьому сценарії дозволяє користувачам хмари швидко організувати свій веб-сайт або блог, а потім зосередитись на підтримці сайту, не турбуючись про придбання, управління або обслуговування обладнання, на якому він працює.[5]

Не всі сценарії відповідають публічній хмарі. Ось кілька недоліків:

- Специфічні вимоги безпеки, які неможливо виконати за допомогою загальнодоступної хмари
- Державна політика, галузеві стандарти або законодавчі вимоги, яким публічні хмари не можуть відповідати
- Неможливість керування апапратними ресурсами якщо це потрібно
- Унікальним бізнес-вимогам, таким як необхідність підтримувати застарілу програму, може бути важко задовольнити

Приватна хмара

У приватній хмарі, середовищем є власний центр обробки даних, користувачі хмари одночасно є повністю відповідальними за придбання та обслуговування апаратних та програмних послуг.

Такий підхід має ряд переваг:

- Можна переконатися, що конфігурація підтримує будь-який сценарій або застарілу програму
- Контроль (і відповідальність) за безпеку
- Приватні хмари можуть відповідати суворим вимогам безпеки, відповідності чи законодавству

Та все ж є деякі причини відходу команд від приватної хмари:

Необхідні початкові витрати CapEx, на придбання обладнання, запуск та обслуговування

Володіння обладнанням обмежує гнучкість - для масштабування потрібно придбати, встановити та налаштувати нове обладнання

Приватні хмари вимагають спеціальних IT-навичок та досвіду

Прикладом використання приватної хмари є ситуація, коли організація має дані, які неможливо розмістити у загальнодоступній хмарі, зокрема з юридичних причин, наприклад, державна політика вимагає, щоб дані зберігалися в країні або в приватному порядку.

Приватна хмара може надати хмарну функціональність як зовнішнім клієнтам, так і певним внутрішнім підрозділам, таким як бухгалтерський облік або управління персоналом.

Гібридна хмара поєднує переваги загальнодоступної та приватної хмари, що дозволяє запускати програми в найбільш підходящому порядку. Наприклад, можна розмістити веб-сайт у публічній хмарі та зв’язати його із спеціально захищеній базі даних, розміщеною у приватній хмарі (або локальному центрі обробки даних).

Звичайно, основою превагою гібридної хмари є її гнучкість і використання всіх переваг одночасно приватного і публічного середовища, проте, може бути дорожче, ніж вибір однієї моделі розгортання, оскільки передбачає витрати на ресурси CapEx, та

інтеграцію між різними моделями хмар, а також передбачає технічні складнощі з налаштування та управління.

Хмарні обчислення є гнучкими і пропонують широкі можливості їх використання. Обрана и модель розгортання хмари залежить від бюджету, необхідного рівня безпеки, масштабованості та можливостей обслуговування.

1.2.3 Види хмарних сервісів

Виділяють основні три категорії хмарних обчислень: IaaS (Інфраструктура як послуга), PaaS (Платформа як послуга), SaaS (Програма як послуга)

Інфраструктура як послуга є найбільш гнучкою категорією хмарних послуг. Вона має на меті надати контроль над обладнанням, яке запускає програму (сервери IT-інфраструктури та віртуальні машини (ВМ), сховища та операційні системи). Тобто IaaS – це надання в оренду ресірсів обладнання. Це миттєва обчислювальна інфраструктура, що надається та управляється через Інтернет.

При використанні IaaS забезпечення роботи та запуск додатку є спільною відповідальністю: хмарний постачальник відповідає за забезпечення належної роботи інфраструктури; клієнт відповідає за те, щоб послуга, була правильно налаштована, актуальна та доступна. IaaS зазвичай використовується в таких сценаріях:

Міграція робочих навантажень. Зазвичай засоби управління IaaS управляються аналогічно локальній інфраструктурі та забезпечують простий шлях міграції для переміщення існуючих додатків до хмари.

Тест і розробка. Команди можуть налаштувати та демонтувати середовища тестування та розробки, динамічно масштабувати їх швидше виводячи нові програми на ринок.

Зберігання, резервне копіювання та відновлення. Організації уникають капітальних витрат та складності управління зберіганням, що, як правило, вимагає кваліфікованого персоналу для управління даними та дотримання вимог законодавства та відповідності. IaaS корисний для управління непередбачуваним попитом та постійно зростаючими потребами у сховищі. IaaS також може спростити планування та управління системами резервного копіювання та відновлення.

Платформа як послуга PaaS забезпечує середовище для створення, тестування та розгортання програмних додатків. Мета PaaS - допомогти створити додаток, не керуючи базовою інфраструктурою. Наприклад, при розгортанні веб-програми за допомогою PaaS не потрібно встановлювати операційну систему, веб-сервер або навіть оновлення системи.

PaaS - це повне середовище розробки та розгортання в хмарі, що має ресурси, що дозволяють організаціям доставляти все, від простих хмарних додатків до складних хмарних корпоративних програм. Доступ до ресурсів здійснюється через захищене інтернет з'єднання.

PaaS дозволяє розробникам створювати програми за допомогою вбудованих програмних компонентів. Включені такі хмарні функції, як масштабованість, висока доступність та можливості для декількох орендарів, що зменшує обєм кодування, яке повинні робити розробники. Аналітика або бізнес-аналітика. Інструменти, що надаються як послуга з PaaS, дозволяють організаціям аналізувати та створювати свої дані. Вони можуть знайти уявлення та закономірності, а також передбачити результати для вдосконалення ділових рішень, зокрема прогнозування, дизайн продукту та віддача інвестицій.

SaaS - це програмне забезпечення, яке розміщується та управляється централізовано для кінцевих споживачів. Зазвичай SaaS

базується на архітектурі, де одна версія програми використовується для всіх клієнтів та ліцензується через щомісячну або річну підписку. Microsoft 365, Skype та Dynamics CRM Online - приклади програмного забезпечення SaaS.

Категорії хмарних технологій є шарами один над одним. Наприклад, PaaS додає додаткові можливості над IaaS. В залежності від продукту що розробляється неоххіді ті чи інші послуги. На наступному рисунку наведено межу керування ресурсами, між постачальником хмарних послуг та користувачем, у кожній категорії хмарних послуг.

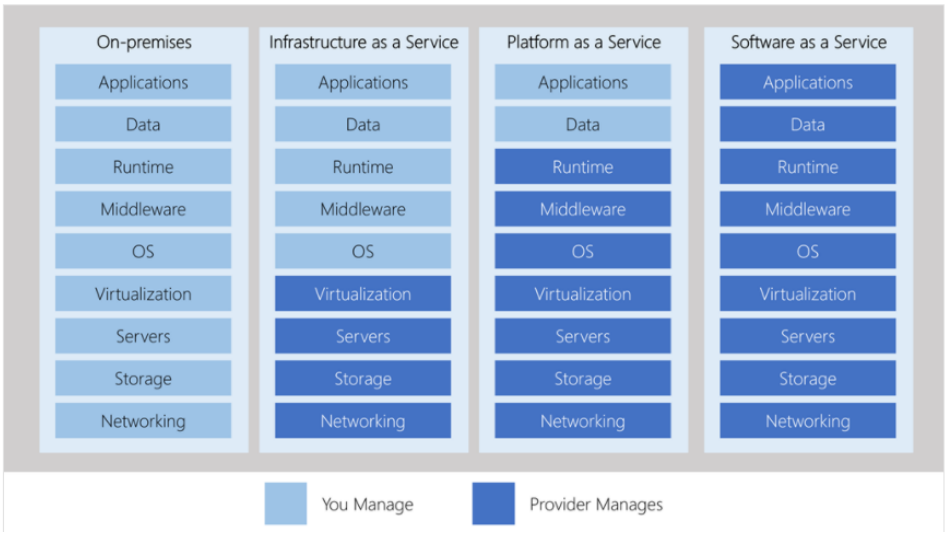

Рисунок 1.2 надання ресурсів залежно від типу хмарних послуг[16]

IaaS вимагає максимального управління користувачем серед усіх хмарних служб. Користувач відповідає за управління операційними системами, даними та програмами, провайдер забезпечує лише фізичним обладнанням та засобами віртуалізації.

PaaS вимагає менше управління користувачами. Хмарний провайдер управляє операційними системами, а користувач відповідає за програми та дані, які запускають і зберігають.

SaaS вимагає найменшої кількості управління. Хмарний провайдер відповідає за все управління, а кінцевий користувач просто використовує програмне забезпечення.

IaaS, PaaS та SaaS містять різні рівні керованих послуг. Можна легко використовувати комбінацію цих типів інфраструктури. Можна використовувати Microsoft 365 на комп'ютерах компанії (SaaS), а в Azure можна розміщувати свої віртуальні машини (IaaS) і використовувати Azure SQL Database (PaaS) для зберігання даних. Завдяки гнучкості хмари можна використовувати будь-яку комбінацію, що забезпечує максимальний результат.

1.3 Переваги і проблеми хмарних обчислень (постановка задачі дослідження)

Хмарні обчислення - це не сервісний підхід "все або нічого". Компанії можуть вибрати хмару для зберігання своїх даних і виконувати логіку стільки, скільки потрібно для задоволення їх бізнес-вимог. Існуючі компанії можуть вибрати поступовий рух, щоб заощадити витрати на інфраструктуру та адміністрування (іменовані як "підняття та зміна"), тоді як нова компанія може почати свою діяльність у хмарі.

Поперше, це рентабельно - хмарні обчислення надають модель ціноутворення на основі оплати за споживання, що приносить із собою багато переваг, серед яких:

Відсутність попередніх витрат на інфраструктуру;

Не потрібно купувати та керувати дорогою інфраструктурою, яка ще йбуде використовуватися не в повній мірі.

Можливість нарощувати ресурси лише тоді, коли вони потрібні, або зменшувати якщо непотрібні.

По друге – масштабовано. Можна будь-коли збільшити або зменшити використовувані ресурси та послуги, виходячи з попиту чи навантаження. Хмарні обчислення підтримують як вертикальне, так і горизонтальне масштабування залежно від потреб бізнесу.

Вертикальне масштабування, також відоме як "масштабування вгору", - це процес додавання ресурсів для збільшення потужності існуючого сервера. Приклади вертикального масштабування: додавання більшої кількості процесорів або збільшення пам'яті.

Горизонтальне масштабування, також відоме як "масштабування назовні" - це процес додавання більшої кількості серверів, які функціонують як одна одиниця. Наприклад, у є кілька серверів, які обробляють вхідні запити.

Масштабування може бути здійснено вручну або автоматично на основі певних тригерів, таких як статистика навантаженості центрального процесора або кількість запитів та ресурсів. Що можна здійснити за лічені хвилини.

По третє – пластично. Оскільки навантаження змінюється через стрибок або падіння попиту, система хмарних обчислень може компенсувати це шляхом автоматичного додавання або видалення ресурсів.

Наприклад, веб-сайт розміщений у статті новин, що призведе до збільшення обсягу відвідуваності протягом ночі. Оскільки хмара еластична, вона автоматично виділяє більше обчислювальних ресурсів

для обробки збільшеного трафіку. Коли трафік починає нормалізуватися, хмара автоматично розподіляє додаткові ресурси для мінімізації витрат.

Інший приклад: якщо запущена програма, яку використовують співробітники, хмара може автоматично додавати ресурси для пікових годин роботи, протягом яких більшість людей отримує доступ до програми, і видаляти ресурси в звичайний кінець дня.

Також, використовуючи хмару, можна зосередитись на тому, що має значення: побудові та розгортанні програм, виключаючи необхідність підтримки виправлень програмного забезпечення, налаштування обладнання, оновлення та інших завдань з управління IT. Все це здійснюється автоматично, щоб переконатися, у використанні найновіших та найкращіх інструментів для ведення бізнесу.

Крім того, комп'ютерне обладнання підтримується та модернізується постачальником. Наприклад, якщо диск виходить з ладу, його замінить постачальник хмарних послуг. Якщо стане доступним нове оновлення обладнання, процес заміни обладнання здійсниться знову ж таки постачальнивом, і буде забезпечено, щоб оновлення обладнання були доступні для автоматично.

По пяте - це надійно. Постачальники хмарних обчислень пропонують послуги резервного копіювання, аварійного відновлення та реплікації даних, щоб переконатися, що клієнтські дані завжди в безпеці. Крім того, надмірність часто вбудована в архітектуру хмарних служб, тому, якщо один компонент виходить з ладу, його місце займає резервний компонент. Це називається стійкістю до несправностей, і це гарантує, що клієнти не зазнають впливу.

Ще одна перевега – глобальність. Хмарні провайдери мають центри обробки даних, розташовані в різних регіонах по всьому світу. Це

дає місцеву присутність, близьку до клієнтів, щоб отримати найкращий час відгуку, незалежно від того, де в світі вони перебувають.

Можна скопіювати свої послуги в декілька регіонів для надмірності та місцевості, або вибрати конкретний регіон, щоб забезпечити відповідність законам про збереження даних та відповідність законам для своїх клієнтів.

І звичайно – безпечність. Захист власного центру обробки даних не завжди висока і окрім мережевого захисту варто організовувати ще й фізичний захист обладнання. В той час як хмарні провайдери пропонують широкий набір політик, технологій, засобів управління та експертних технічних навичок, які можуть забезпечити кращий захист, ніж більшість організацій. Результатом є посилена безпека, яка допомагає захистити дані, програми та інфраструктуру від потенційних загроз.

Що стосується фізичної безпеки - загрози хмарній інфраструктурі, провайдери інвестують значні кошти у стіни, камери, ворота, персонал служби безпеки тощо, щоб захистити фізичні активи. Вони також мають суворі процедури, щоб забезпечити працівникам доступ лише до тих ресурсів, якими вони уповноважені управляти.[18]

Щодо цифрової безпеки. Хмарні провайдери пропонують інструменти, які допомагають усунути загрози безпеці: організовуючи подвійну авторизацію, токени.

Економія від масштабу - це здатність робити речі ефективніше або з меншою вартістю за одиницю при роботі в більших масштабах. Ця вигідна вартість є важливою перевагою хмарних обчислень. Хмарні провайдери, такі як Microsoft, Google і Amazon, - це великі компанії, що використовують переваги економії від масштабу. Потім ці постачальники можуть передати заощадження своїм клієнтам.

Традиційно проблеми безпеки є основною перешкодою для організацій, які розглядають хмарні послуги, особливо державні хмарні служби. Однак у відповідь на попит безпека, яку пропонують постачальники хмарних послуг, стабільно перевершує локальні рішення безпеки.

Тим не менше, підтримка хмарної безпеки вимагає інших процедур та набору навичок співробітників, ніж у звичайних IT-середовищах. Деякі найкращі практики хмарної безпеки включають наступне:

Спільна відповідальність за безпеку. Як правило, хмарний постачальник відповідає за захист хмарної інфраструктури, а замовник за захист своїх даних у хмарі, але також важливо чітко визначити право власності на дані між приватними та державними третіми сторонами.

Шифрування даних: Дані повинні шифруватися, коли вони перебувають у стані спокоюзберігаються, переміщуються та під час використання. Клієнти повинні підтримувати повний контроль над ключами безпеки та апаратним модулем безпеки.

Ідентифікація користувача та управління доступом: Клієнтам та IT-командам потрібне повне розуміння та доступ до мережі, пристрою, програми, дани залежно від ролі в системі.

Спільне управління: належний зв’язок та чіткі, зрозумілі процеси між IT, операціями та командами безпеки забезпечать безперебійну інтеграцію хмарних технологій.

Моніторинг безпеки та відповідності: починається з розуміння всіх стандартів відповідності нормативним актам, що застосовуються у галузі, та налаштування активного моніторингу всіх підключених систем та хмарних служб для підтримки видимості обміну даними між загальнодоступним, приватним та гібридним хмарним середовищем.

Найголовнішим недоліком хмарних технологій вважають залежність від мережі інтернет, оскільки не маючи доступу, неможливо отримати свої дані та працювати з ними. Та цей недолік можно порівняти з залежністю від електричної енергії при роботі з локальними ресурсами: за її відсутності, всі інформаційні дані збережені на фізичному дисковому просторі стають недоступними через неможливість їх обробки і відображення.

Можна виділити ще одну проблема, а точніше наслідок від глобалізації хмари – експоненційне збільшення обємів даних, порівнюючи з економічною теорією – зростання попиту, яке вимагає росту пропозиції. Тому виникає потреба в розширенні додатків, використання ними все більших ресурсів, а отже їх ефективного масштабування і розширення, а в разі потреби – навпаки згортання. Рішенню саме цієї проблеми присвячені наступні розділи роботи.

Висновки до розділу

Отже, технологічний світ поступово переходить до хмарних технологій. І передумовами такого переходу стали зростаючі потреби в інформаційних та обчислювальних ресурсах. Запитів на хмарні технології стає більше. Ми бачимо, що індустрія прискорено розвивається, і зростає потреба в клауд продуктах на ринку.

Найперша проблема, яку вирішують хмарні технології – прив'язка до конкретного вендору обладнання, що вирішується за допомогою технологій віртуалізації. Віртуальні машини досить пластичні і дані можна легко переносити з одного сервера на інший разом з віртуальною машиною, не втрачаючи при цьому програмного налаштування операційної системи.

Також на зміну серверам та хостингам запропонували гігінтські дата центри, що окрім апаратних ресурсів надають в оренду широкий спектр програмних можливостей. Виділяють 3 види хмар: публічні (Amazone, Google, Microsoft, IBM...), приватні – побудонані на апаратних ресурсах дата центрів, але з використанням спеціальних політик і технологій, та гібридні – що обєднують в собі переваги і можливості двох попередніх.

Оскільки дані користувачів зберігаються у загальній мережі, важливим аспектом хмар є питання безпеки, що забезпечується на найвищому рівні. Ще одна з причин появи хмари - це Risk Management, оскільки в разі потреби досить легко відмовитися від орендованого сервісу або змінити його при цьому не втрачаючи даних.

Також за допомогою хмарних технологій вирішують потребу швидкого створення та налаштування нових сервісів, орієнтуючись на потреби клієнтів. Тому хмарні провайдери розробляють широкий набір послуг як Saas, Paas, Iaas.

Наступна проблема, яку вирішують хмарні технології – розростання самих додатків, і відповідно необхідність розділення їх на мікросервіси, тому з'явилася мікросервісная архітектура, що в свою чергу потребує процедур для їх обєднання в одну систему, яку вирішує технології контейнеризації та кластеризації.

Важливим і найбільш цікавим нововведення технологій є можливість легкого і автоматичного масштабування додатків, що є необхідним для активно змінного інформаційного простору. Саме це питання буде розглянуте в наступних розділах роботи.

РОЗДІЛ 2. ПЛАТФОРМИ ТА СЕРВІСИ ХМАРНИХ ОБЧИСЛЕНЬ

2.1 Docker – інструмент контейнризації

Docker - найпопулярніший інструмент для створення та запуску контейнерів. Хоча ранні форми контейнерів були представлені десятки років тому (з такими технологіями, як Jacks FreeBSD та розділи робочого навантаження AIX), контейнери набули свого головного розвитку в 2013 році, коли Docker представив їх широкому загалу завдяки новій реалізації, зручній для розробників та хмар.

Docker починався як проект з відкритим кодом, але сьогодні він також посилається на Docker Inc., компанію, яка виробляє Docker - комерційний набір інструментів для контейнерів, який базується на проекті з відкритим кодом. Docker був побудований на традиційній технології контейнерів Linux (LXC), але забезпечує більш детальну віртуалізацію процесів ядра Linux і додає функції, що дають можливості розробникам для створення, розгортання, управління та захисту.

Хоча сьогодні існують альтернативні контейнерні платформи (такі як Open Container Initiative (OCI), CoreOS та Canonical (Ubuntu) LXD), Docker настільки широко вподобаний, що він практично є синонімом контейнерів і іноді помилково вважається конкурентом додаткових технологій, таких як Kubernetes. З розвитком технології контейнеризації зростала необхідність розвробки інструментів для планування та автоматизації розгортання контейнерів, створення мереж, масштабованості та доступності, зрештою зявилися інструменти оркестровки контейнерів. Хоча інші варіанти організації контейнерів - зокрема напочатку розвитку технології Docker Swarm та Apache Mesos

займали переважне значення, проте Kubernetes швидко став більш швидкозростаючим.

Контейнери використовують перевагу однієї форми віртуалізації операційної системи (ОС), яка дозволяє кільком додаткам спільно використовувати ОС, ізолюючи процеси та контролюючи обсяг процесора, пам'яті та диска, до якого ці процеси можуть отримати доступ.

Можливо, простіше або корисніше пояснити контейнери як останню точку континууму автоматизації та абстракції IT-інфраструктури. У традиційній інфраструктурі програми працюють на фізичному сервері компанії і забирають усі ресурси, які вони можуть отримати. Все ж є можливість запуску декількох програм на одному сервері, проте є ризик, що одна з них збере ресурси за рахунок інших, або ж запуститься програма, що марно витрачає ресурси і не масштабується.[10]

Віртуальні машини (ВМ) - це сервери, абстраговані від власне апаратного забезпечення комп'ютера, що дозволяє запускати кілька ВМ на одному фізичному сервері або одній ВМ, що охоплює більше одного фізичного сервера. Віртуальна машина запускає власний екземпляр ОС, і можна ізолювати будь яку програму у власній віртуальній машині, зменшуючи ймовірність того, що програми, які працюють на одному базовому фізичному обладнанні, впливатимуть одна на одну. Віртуальні машини ефективніше використовують ресурси, набагато простіші та економічніші в масштабуванні, ніж традиційна інфраструктура.

Контейнери виводять цю абстракцію на більш високий рівень - зокрема, крім спільного використання базового віртуалізованого обладнання, також розділяють віртуалізоване ядро ОС. Контейнери пропонують однакову ізоляцію, масштабованість та одночасність використання віртуальних машин, але оскільки вони не зазнають

навантаження власного екземпляра ОС, то мають меншу вагу, порівняно з віртуальними машинами.[18] Вони ефективніші за ресурсами - дозволяють запускати більше додатків на меншій кількості машин (віртуальних та фізичних) із меншою кількістю екземплярів ОС.

2.2. Kubernetes

Платформа для організації контейнерів, яка автоматизує розгортання, управління та масштабування програм також відома як "k8s" або "kube".

Kubernetes вперше був розроблений інженерами Google, у 2014 році. Він є нащадком Borg, платформи для організації контейнерів, що використовується в Google внутрішньо. Kubernetes – в перекладі з грецької - керманич, тому зображене кермо в логотипі Kubernetes. Сьогодні Kubernetes та широка контейнерна система наближаються до загальної обчислювальної платформи, яка конкурує - якщо не перевершує - віртуальні машини (ВМ) як основні елементи сучасної хмарної інфраструктури та додатків. Така система дозволяє організаціям забезпечити високопродуктивну платформу як послугу (PaaS), що вирішує численні завдання, пов'язані з інфраструктурою та операціями, та проблеми, пов'язані з розробленням хмарних технологій, завдяки чому команди розробників можуть зосередитися виключно на кодуванні та інноваціях.

Розробники обирають Kubernetes за його широку функціональність, величезну та зростаючу систему інструментів підтримки, а також інтегрованість та портативність з провідними хмарними провайдерами (деякі з них пропонують повністю керовані послуги Kubernetes).

До основних функцій Kubernetes відносять:

Планування та автоматизація завданнь, пов’язаних з контейнерами:

Розгортання вказаної кількості контейнерів на вказаному хості та підтримка їх роботи у бажаному стані. Kubernetes дозволяє ініціювати, призупиняти, відновлювати або відміняти збірки.

Автоматична активізація контейнера в Інтернет, використовуючи ім'я DNS або IP-адресу.

Балансування навантаження та масштабування: коли трафік контейнера зростає, Kubernetes може застосувати балансування навантаження та масштабування, щоб розподілити його по мережі для підтримки стабільності.

Самовідновлення для високої доступності: коли контейнер виходить з ладу, Kubernetes може перезапустити або замінити його автоматично; він також може знести контейнери, які не відповідають вашим вимогам до перевірки працездатності.

Насправді, користуючись Docker і створюючи масштабні розгортання контейнерів на основі Docker, оркестрація Kubernetes є логічним наступним кроком для управління цими робочими навантаженнями.

Основні компоненти архітектури Kubernetes включають наступне:

Кластери є основними елементами архітектури Kubernetes. Кластери складаються з вузлів, кожен з яких представляє один обчислювальний хост (віртуальна або фізична машина). Кожен кластер складається з декількох робочих вузлів, які розгортають, запускають та керують контейнерними програмами, та одного головного вузла, який контролює та контролює робочі вузли. Головний вузол запускає службу

планування, яка автоматизує час та місце розгортання контейнерів на основі вимог до розгортання, встановлених розробником, та доступних обчислювальних можливостей. Кожен робочий вузол включає в себе інструмент, який використовується для управління контейнерами - наприклад, Docker, - і програмний агент під назвою Kubelet, який отримує та виконує замовлення від головного вузла.[3]

Структури - це групи контейнерів, які використовують однакові обчислювальні ресурси та одну мережу. Вони також є одиницею масштабованості в Kubernetes: якщо контейнер у підсистемі отримує більше трафіку, ніж він може обробити, Kubernetes реплікує підсистеми на інші вузли кластера, тому гарною практикою є компактні структури, щоб вони містили лише контейнери, які повинні спільно використовувати ресурси.

Сервісна сітка Istio - Kubernetes може розгортати та масштабувати модулі, але він не може керувати або автоматизувати маршрутизацію між ними та не надає жодних інструментів для моніторингу, захисту або налагодження цих з'єднань. Зі збільшенням кількості контейнерів у кластері кількість можливих шляхів з'єднання між ними зростає експоненціально (наприклад, два контейнери мають два потенційні з'єднання, але 10 підсистем мають 90), створюючи потенційні проблеми конфігурації та управління. До кожного кластера Kubernetes Istio додає контейнер бічної підвіски - по суті невидимий для програміста та адміністратора - який налаштовує, контролює та керує взаємодією між іншими контейнерами.[1]

За допомогою Istio можна встановлити єдину політику, яка налаштовує з'єднання між контейнерами, щоб непотрібно було налаштовувати кожне з'єднання окремо. Це полегшує налагодження зв’язків між контейнерами.

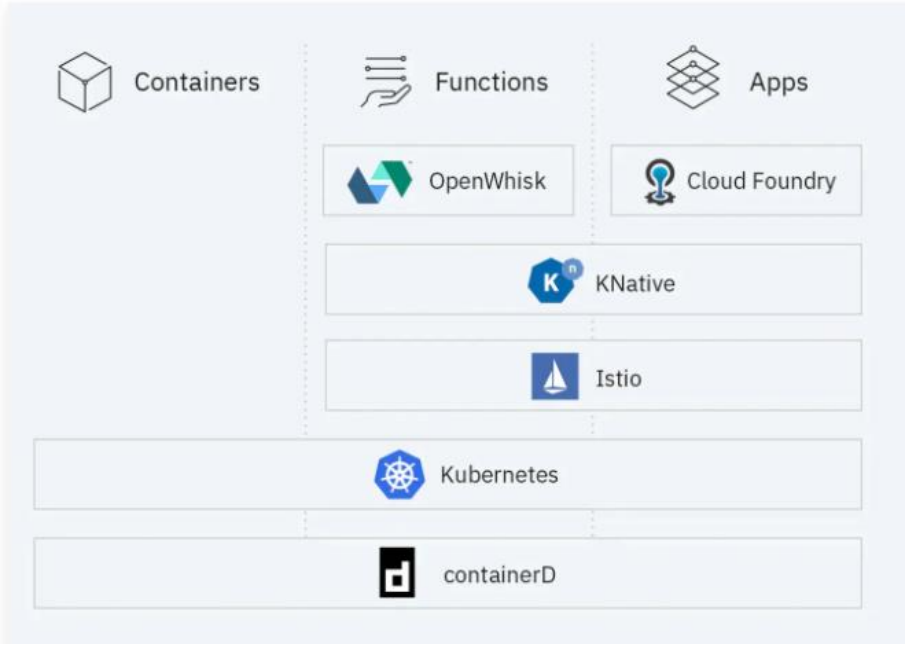

Рисунок 2.1 Місце Kubernetes в системі технологій конткйнеризації [12]

Istio також пропонує інформаційну панель, яку команди та адміністратори DevOps можуть використовувати для моніторингу затримок, помилок часу роботи та інших характеристик з'єднань між контейнерами. Він також створює безпеку - зокрема, управління ідентифікацією, яка утримує несанкціонованих користувачів від підробки службового дзвінка між контейнерами - та можливості автентифікації, авторизації та аудиту (AAA), які можуть використовувати фахівці з безпеки для моніторингу кластера.

2.3. Порівняння Kubernetes в Amazon, Google, Microsoft

Як було розглянуто вище, великі програмні рішення що являють собою декільна мікросервісів, зручно розміщати в окремих контейнерах, а потім керувати ними за допомогою спеціальних програм – оркстраторів. Так одним із найільш популярних рішень такого управління в хмарі є платформа Kubernetes. Звичайно, найбільші постачальники хмарних послуг, зокрема Amazon Web Services (AWS), Microsoft Azure та Google Cloud Platform (GCP), відіграють ключову роль у наданні пслуг для управління архітектурою Kubernetes.[7]

Проаналізуємо та порівняємо різні функції та послуги, які пропонують публічні хмари, і виділемо одну з них для реалізації розгортання контейнерного додатку та його автомасштабування.

Amazon має реалізацію Kubernetes сервіса – Elastic Container Service (EKS) - це керована послуга, яка стала доступною в 2018 році. EKS повністю сумісна з програмами, які працюють на будь-якій стандартній архітектурі Kubernetes. Специфічним у Amazon EKS є те, що для кожного кластера запускається площина управління Kubernetes з одним орендарем, де площина управління не розділена між кластерами.

Служба Microsoft - Azure Kubernetes (AKS) - це рішення також доступне з 2018 року та раніше Microsoft вже анонсував службу Azure Container Service для Apache Mesos та Docker Swarm., що свідчить про певний досвід в організації контейнерів.

Kubernetes вперше був представлений компанією Google в 2015 році. Google Kubernetes Engine (GKE) - це керована виробнича архітектура для розгортання контейнерних програм, яка є одним з найсучасніших рішень, що дозволяє швидко встановити контейнерні програми, без вимоги встановлювати та керувати кластерами Kubernetes.

Порівняємо деякі параметри цих продуктів і їх особливості у різних постачальників.

Версія продукту: Google зазвичай має найновішу збірку сервіса, за ним Microsoft Azure та AWS. Також вирішення проблем в роботі швидте у версії Google, ніж у Microsoft Azure та AWS, що зрозумілю, оскільки саме Google є основним розробником сервісу. Також

Автоматичне оновлення:

GKE забезпечує повністю автоматизоване оновлення для кластера без втрчання користувача; AKS дозволяє оновити кластер за допомогою простої команди. А от в Amazon EKS, користувачеві потрібно надіслати деякі інструкції командного рядка, що ускладнює роботу порівняно з попередніми програмами.

Моніторинг ресурсів:

Для моніторингу Google Cloud надає Stackdriver, що контролює весь сервіс, вузли, та усі компоненти Kubernetes всередині платформи, а також інтегрує реєстрацію. Microsoft Azure пропонує дві послуги: Azure Monitor для оцінки стану контейнера та Application Insights для моніторингу компонентів Kubernetes, також користувачеві потрібно налаштувати Istio (рішення службової мережі) для моніторингу компонентів Kubernetes. AWS не має інтегрованого рішення для моніторингу, але замість цього доступні безліч сторонніх рішеннь.

Пули вузлів: для різних типів робочих навантажень ефективно використовувати різні машини. Наприклад, для систем баз даних, потрібно більше оперативної пам'яті та кращих дисків, тоді як алгоритми машинного навчання, потребують кращого процесора. З допомогою пулів вузлів можна забезпечити найкращу доступність ресурсів.[15]

Google та AWS мають перевагу, надаючи широку підтримку пулу вузлів, тоді як в Microsoft Azure така можливість недоступна.

Автомасштабування: Kubernetes має можливість автоматично масштабувати вузли, щоб використовувати ресурси на вимогу, і це

найцікавіша його функція. Таким чином, послуги доступні постійно, при чому досягається економічно ефективна інфраструктура.

В області автоматичного масштабування Google Cloud є лідером як найбільш зріле рішення, доступне на інтерфейсі. Користувачеві потрібно лише вказати бажаний розмір віртуальної машини та діапазон вузлів у пулі вузлів. А рештою кроками керує Google Cloud. AWS посідає друге місце за автоматичним масштабуванням, оскільки йому потрібні незначні ручні конфігурації. Microsoft Azure представила автомасштаб, який частково охоплюється налаштуванням.

Висока доступність: означає, доступність кластера навіть якщо певний центр обробки даних, на якому розміщений кластер, не працює. Для цього, головні вузли розподілені на декілька зон доступності у кожної з трьох служб.

Рольовий контроль доступу (RBAC): контроль доступу на основі ролей (RBAC) через API Kubernetes, реалізовано у коженого з трьох постачальників послуг.

Робота через віртуальну машину або ж на фізичному обладнанні: ВМ - це емульована машина, що працює над реальним обладнанням і має ряд переваг для хмарного провайдера – можливість розділення великої машину на декілька менших для спільного використання кількома клієнтами, легке перенесення з однієї фізичної машини на іншу. Проте, рівень віртуалізації додає деякі складності дещо знижує продуктивність порівняно з фізичному апаратним забезпеченням. На даний момент лише у AWS доступна можливість відмови від віртуалізвції.

Ціноутворення: у GKE та AKS рахунов включають лише послуги, що використовуються, управління кластерами, головними вузлами та машини надається безкоштовно. Тоді як в Amazon EKS відбувається

тарифікація обслуговування за кожен розгорнутий кластер, разом з оплатою послуг.

Наступна таблиця узагальнює переваги і недоліки послуг різних провайдерів:

Таблиця 2.1: Переваги і недоліки основних сервіса Kubernetes найбільших хмарних провайдерів

	Переваги	**Недоліки**
AWS	Безпека, надійність та масштабованість	Потенційно дорожчий за інші варіанти
	Варіанти полегшених послуг, щоб випробувати Kubernetes та контейнери, перш ніж перенести локальне середовище у хмару	Складність у використанні, необхідні спеціальні знання
	Fargate надає спосіб розгортати контейнери, без втручання у серверну інфраструктуру	
Azure	Інтуїтивно зрозумілий для розробників інтерфейс	Розгортання дещо повільніше
	Підтримує контейнери Windows і Linux	Не підтримує гібридні контейнери
	AKS безкоштовний.	
GCP	Оригінальний творець Kubernetes, тому впровадження нових функцій відбувається швидше	Не інтегрується з хмарними послугами IaaS
	Ідеально підходить для роботи з великими даними та штучним інтелектом	
	Добре працює з командами DevOps	

Отже, який найкращий сервіс використання з Kubernetes? Однозначної відповіді немає, оскільки зрештою всі платформи пропонують відмінні сервіси та можливості. Та якщо є необхідність працювати зі штучним інтелектом – найкращі можливості надає Google, для значного інтегрування з хмарою – найліпше використовувати

Amazon, щоб отримати якомога більше додаткових програм як послуг – варто розглянути можливості Azure.

Для подальшої реалізації розгортання та масштабування додатку розглянемо платформу Azure, оскільки згідно з порівняльними даними поперше інтерфейс є найбільш зрозумілим саме в цій платформі, подруге, масштабування потребує деяких налаштувань, що і будуть розглянуті в наступному розділі.

2.4 Microsoft Azure

Azure - це платформа хмарних обчислень Microsoft. Azure надає понад 100 служб, які дозволяють робити все: від запуску існуючих програм на віртуальних машинах до розроблення нових програмних парадигм, таких як інтелектуальні боти та віртуальна реальність.

Наприклад: віртуалізація та контейнеризація хмарних обчислень, послуги баз даних, як реляційних, так і NoSQL, аутентифікація та захист користувачів, мережеві служби підключення центру обробки даних до хмари, рішення для зберігання структурованих, так і неструктурованих даних, служби штучного інтелекту та машинного навчання.

Розглянемо більш детально місце Azure серез хмарних технологій.

Сучасні компанії все частіше переносять свої програми на віртуальні машини та хмара - це більше, ніж просто "інше місце для запуску сервера". Перенесення існуючих програм на віртуальні машини є гарним початком, не лише переходу до більших потужностей та економічних вигод, а й для розивтку проекту з залученням можливостей що пропонуються постачальниками хмарних технологій

Наприклад, Azure пропонує послуги на базі штучного інтелекту та машинного навчання, які можуть природним чином спілкуватися з користувачами за допомогою зображення, слуху та мови. Сервіс також пропонує можливості для зберігання даних, які динамічно зростають для розміщення великих обсягів даних, та глобальну інфраструктуру, яка завжди доступна для створення додатків.

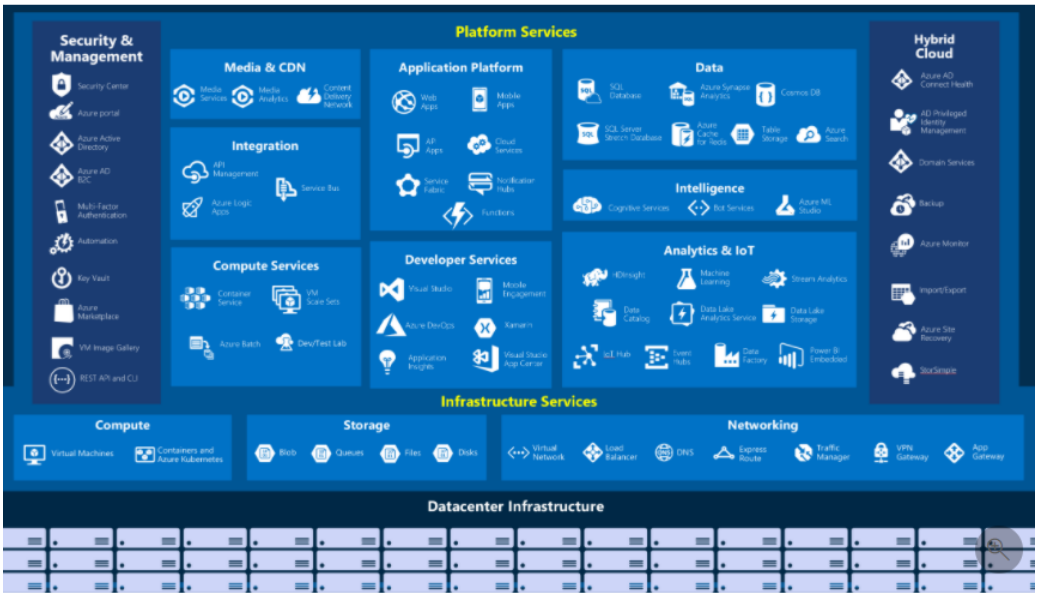

Рисунок 2.2 Набір послуг представлений платформою Azure [20]

Відновлення наслідків катастрофи та безперервність бізнесу завжди були природними явищами для хмари, оскільки хмара забезпечує економічно ефективне резервування для захисту даних від збоїв системи та фізичної відстані, необхідної для відновлення даних та програм у разі локального відключення або катастрофи. Всі основні державні хмарні провайдери пропонують службу Disaster-Recovery-as-a-Service (DRaaS).

Все, що передбачає зберігання та обробку величезних обсягів даних на високих швидкостях - і вимагає більшої ємності для зберігання та обчислень, ніж більшість організацій може або хоче придбати та

розгорнути локально - є ціллю для хмарних обчислень. Приклади включають:

- Аналітику великих даних
- Інтернет речей (IoT)
- Штучний інтелект - зокрема, машинне навчання та програми глибокого навчання

Для команд розробників, які застосовують Agile або DevOps (або DevSecOps) для впорядкування розробки, хмара пропонує самообслуговування кінцевих користувачів на вимогу, яке не дозволяє операційним завданням, таким як обертання серверів розробки та тестування, стати вузькими місцями для розвитку.

Висновки до розділу

Kubernetes зарекомендував себе як найпопулярніша послуга організатора контейнерів і став важливим рішенням для управління кластерами. AWS, Microsoft Azure та Google Cloud Platform - найпопулярніші хмарні провайдери, що забезпечують широкі можливості для роботи і мають значну кількість переваг.

Kubernetes - один з найбільш швидкозростаючих проектів з відкритим кодом в історії, що продовжує стрімко розвиватися серед розробників та компаній, які його використовують. Попит на фахівців що працюють з цією платформою стрімко зростає, зростає також кількість компаній що використовують цей продукт і його можливості.

РОЗДІЛ 3: ОПТИМІЗАЦІЯ ПРОЦЕСУ РОЗГОРТАННЯ КЛАСТЕРНОГО ДОДАТКУ В СИСТЕМІ KUBERNETES

3.1. Процес розгортання веб додатку в мережі

Кожен веб додаток зрештою має бути опублікованим в мережі інтернет, бути доступним за запитом користувачів, оновнюватись новими налаштунками, зберігати дані і т.ін. Існує безліч способів розгорнути додаток в мережі, але основними принципами є – існування фізичного простору для зберігання даних (фізичний сервер, віртуальний хостинг, віртальний виділений сервер, VPS-хостинг або ж хмара), унікальнаий домен або адреса сайту.

Отже, щоб розгорнути деякий додаток в мережі необхідно, перевірити доступність доменного імені, зарезервувати його. Обрати надійний хостинг, оформити підписку та завантажити файли додатку через менеджер сайтів (FTP –клієнт) на хостинг.

Як бачимо, послідовність кроків досить нескладна, але основна проблема у терміні «надійний». Так, зокрема звичайнй фізчний сервер або віртуальний хостинг – як машина для зберігання даних з мінімальним набором операційних послуг є досить нестабільною у використанні. І навіть, якщо не брати до уваги можливі перебої у його роботі, що відбувається доволі часто, основною проблемою є складність розширення ресурсів, оновлення додатку та відказостійкість.

З ростом додатку, зростає потреба в обчислювальних ресурсах, так, їх можна додати, проте збільшення потужності сервера зазвичай не пропорційне збільшенню його вартості. Також, виникає ризик простою

ресурсів, тоді як зворотнє масштабування знову викличе труднощі і додаткові витрати. Ще одним важливам фактором є неможливість перенесення даних у випадку відсутності доступу, оскільки звичайний хостинг не передбачає резервного копіювання чи бекапів даних.

Все ж для багатьох, зокрема, невеликих проектів віртуальний хостинг чи фізичний сервер є ідеальним рішенням, оскіль надає користівачу певний рівень безпеки, персональний дисковий простір, операційну память, потужності процесора та повний контроль над виділеними ресурсами.

Та для масштабованих, багатокомпонентних, постійно розвиваючих проектів хостинг технології є складними у використанні. Тому розглянемо процес розгортання додатку, що потребуватиме активного розширення в майбутньому з використанням хмарних технологій та контейнеризації.

Наведемо таблицю порівняння послуг що пропонуються віртуальним хостингом та хмарним провайдерами для розгортання додатків:

Таблиця 2. Порівняння можливостей розгортання додатків на хостингах та хмарах

	Хостинг	Хмара
Віртуалізація	Віртуальні машини на одному сервері	Контейнери, розподілені між декількома фізичними серверами
Масштабованість	Можлива, але складно організована	Автоматична
Виділення додаткових ресурсів	Тривалий процес	Миттєво
Ресурси памяті та процесора	Розділювані	Ізольовані
Вибір ОС	Пропонується хостом	Будь-яка
Вартість	Низька	Середня
Надійність	Непередбачувана	Висока

Отже, нехай ми маємо деякий додаток з мікросервісною архітектурою, кожен мікросервіс може бути розроблений будь якою мовою програмування, та для взаємодії з іншими мікросервісами має бути сформований декий інтерфейс. Далі ці мікросервіси треба запакувати в окремі контейнери, де вони вже будуть працювати а не просто являтимуть собою набір файлів. При цьому буде створено образ сервісу і його конфігурація занотована в Dockerfile. А вже наступним кроком потрібно встановити звязки між такими контейнерами, забезпечити їм необхідну конфігурацію та запустити в мережі, для цих операцій потрібні сервіси оркестрації контейнерів такі як Kubernetes, OpenShift, Docker Swarm та ін. Також ці сервіси дозволяють проводити моніторинг роботи як окремого контейнера так і їх звязків, налаштовувати автоматичне налаштування для контейнерів та керувати їх роботою.

3.2 Створення кластера Azure Kubernetes

Перш ніж розгорнути будь-яку програму, потрібно створити кластер. Розглянемо кілька концепцій, для розгортання нового кластера, зокрема в середовищі Azure Kubernetes Service (AKS).

Kubernetes базується на кластерах, замість того, щоб мати єдину віртуальну машину (ВМ), вона використовує кілька машин, що працюють як одна. Ці ВМ називаються вузлами. Kubernetes - це кластерний оркестратор, що надає переваги доступності, моніторингу, масштабування та оновлення.

Кластер базується на вузлах - нодах. У кластері Kubernetes є два типи вузлів, які забезпечують функціональність: Вузли управління -

розміщують параметри управління кластера і зарезервовані для служб, які керують кластером. Вони відповідають за надання API, який всі інші вузли використовує для комунікації. На цих вузлах не розгортається навантаження; Ноди – вузли, що відповідають за виконання нестандартних робочих навантажень та додатків, таких як компоненти хмарної служби.

Існує дві архітектури кластерів для розгортання на основі Kubernetes:

Єдина система управління та кілька вузлів - найпоширеніший архітектурний шаблон. Цей шаблон найпростіший у розгортанні, але він не забезпечує високої доступності основних служб управління кластером. Якщо вузол управління стає недоступним з будь-якої причини, ніякої іншої взаємодії з кластером не може відбутися. Незважаючи на, можливу недоступність, цієї архітектури зазвичай достатньо. Менше ймовірно, що основні служби управління стануть недоступними порівняно з вузлом, який переходить у офлайн-режим. Вузли управління піддаються меншій модифікації, ніж ноди, і більш еластичні.

Єдина система управління та єдиний вузол. Ця архітектура забезпечує лише один вузол, який розміщує управління і робочий вузол. Це корисно під час тестування або експериментування з різними концепціями Kubernetes. Єдина площина управління та архітектура одного вузла обмежує масштабування кластера і робить цю архітектуру непридатною для виробничого та інтерактивного використання.

Для групування вузлів створюється Node pools, де вказується розмір віртуальної машини кожного вузла. Пули вузлів використовують набори масштабів віртуальних машин як базову інфраструктуру, щоб дозволити кластеру масштабувати кількість вузлів у пулі вузлів. Нові

вузли, створені у пулі, завжди матимуть розмір, вказаний під час створення пулу вузлів.[13]

Кластер Kubernetes за замовчуванням блокує всі зовнішні зв'язки. Наприклад, розгортаючи веб-сайт, доступний для всіх клієнтів, необхідно вручну створити доступ або ж спеціально нелаштувати. Конфігурація вимагає змін, пов'язаних з мережею, які переадресовують запити від клієнта на внутрішній IP-адрес кластера а потім, до додатка. AKS дозволяє подолати складність, увімкнувши так звану маршрутизацію додатків HTTP. Ця надбудова полегшує доступ до програм на кластері за допомогою автоматичного розгортання контролера входу.[20]

Контролери Ingress надають можливість розгортати та виставляти програми у всьому світі без необхідності налаштування мережевих служб.

Схема маршрутизації додатків HTTP, яка показує, як контролер входу сканує вхідні ресурси та створює правила DNS, щоб зробити програми доступними для зовнішніх клієнтів.

Контролери Ingress створюють сервер зворотного проксі-сервера, який автоматично дозволяє обслуговувати всі запити з одного виходу DNS. Не потрібно створювати запис DNS кожного разу, коли розгортається нова служба. Про це потурбується контролер входу. Коли новий вхід розгортається в кластері, контролер входу створює новий запис у керованій Azure зоні DNS і пов'язує його з існуючим балансером навантаження. Ця функціо нальність забезпечує легкий доступ до ресурсу через Інтернет без необхідності додаткової конфігурації.

Для створення нового кластеру для розгортання серверу в систем Azure Kubernetes конфігурація буде налаштовуватись у Cloud Shell. Та найперше, для користування сервісами Azure необхідно зареєструватися

у системі та створити підписку, з якою власне і буде звязано розгорнутий додаток.

Отже, найперше – створимо групу, що включатиме кластери за допомогою конанди az group create, вказавши обовязкові параметри:

```
az group create --name rg-MicroExample --location eastus
```

Далі, в цій групі ствоимо сам кластер, вказавши основні параметри:

```
az aks create \
    --resource-group rg-MicroExample \
    --name aks-MicroExample \
    --node-count 2 \
    --enable-addons http_application_routing \
    --dns-name-prefix example-kubernetes-$RANDOM \
    --generate-ssh-keys \
        --node-vm-size Standard_B2s
```

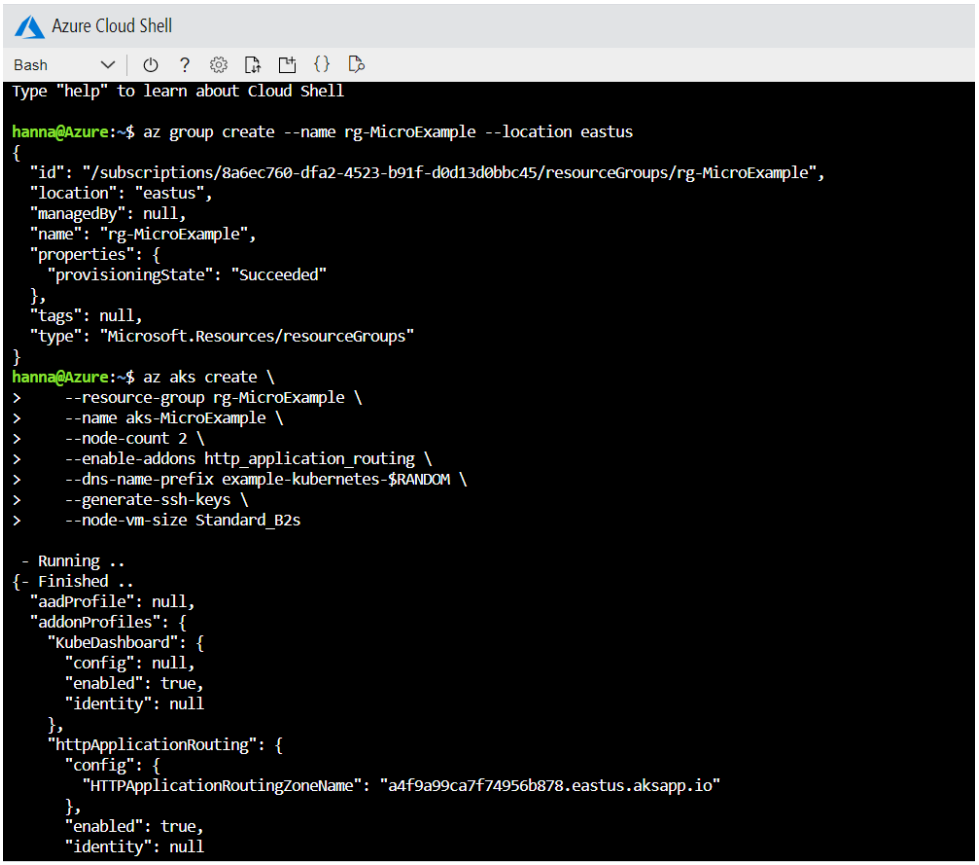

Рисунок 3.1 встановлення конфігурації в Cloud Shell

Команда `az aks create` створить кластер з назвою `aks-MicroExample` та у групі rg-MicroExample, що буде складатися з 2х нод, маршрутизація до кластера буде здійснюватись через `http_application_routing` до ДНС кластера `example-kubernetes`, а розмір віртуальної машини задано `Standard_B2s`.

Наступним кроком необхідно встановити зв'язок кластера і групи:

```
az aks get-credentials --name aks-MicroExample --resource-group rg-MicroExample
```

команда додасть конфігуацію в файл управління, який містить всю інформацію для доступу до кластерів. Kubectl дозволяє управляти кількома кластерами з одного інтерфейсу командного рядка.

Для перевірки створеної конфігурації існує команда `kubectl get nodes`, що поверне інформацію по обом створеним нодам: NAME, STATUS, ROLES, AGE, VERSION

```
hanna@Azure:~$ az aks get-credentials --name aks-MicroExample --resource-group rg-MicroExample
Merged "aks-MicroExample" as current context in /home/hanna/.kube/config
hanna@Azure:~$ kubectl get nodes
NAME                                STATUS   ROLES   AGE     VERSION
aks-nodepool1-25081665-vmss000000   Ready    agent   5m54s   v1.17.13
aks-nodepool1-25081665-vmss000001   Ready    agent   5m54s   v1.17.13
hanna@Azure:~$
```

Зисунок 3.2 статус створеної конфігурації

Важливим для налаштування контейнеру є файл маніфест Kubernetes, що дозволяє декларативно описувати навантаження у форматі YAML та спрощувати управління об'єктами Kubernetes. І містить всю інформацію, необхідну для створення та управління робочим навантаженням.

Структура файлів маніфесту відрізняється залежно від типу ресурсу. Однак файли маніфесту мають спільні інструкції, яку

визначають різні аспекти, наприклад API, потрібні для використання, та тип робочого навантаження для створення.

Зокрема, файл маніфесту обовязково має включати основні елементи:

```
apiVersion: apps/v1
kind: Deployment
metadata:
        name: micro-website # the name of the deployment
```

Параметр apiVersion визначає API ендпоінт сервера, що управляє об'єктом, який розгорнете.

Параметр kind визначає навантаження, яке створить це розгортання.

Інші загальні параметри - це метадані та ключі імен. Усі ресурси Kubernetes повинні мати назву, що знаходиться всередині параметра метаданих.

Щоб застосувати метафайл для встановлення системи використовують команду `kubectl`:

```
kubectl apply -f ./deploy.yaml
```

Наступним кроком є створення образу контейнера, і розміщення його в реєстрі контейнера. Azure забезпечує Azure Container Registry для зберігання зображень (образів) контейнерів, а команда `az acr create` дозволяє створити такий образ

```
az acr create \
    --resource-group rg-MicroExample \
    --name ACRMicroExample\
        --sku Basic
```

Але щоб пов’язати кластер ресурсної групи та образ контейнера, необхідно обновити кластер командою `az aks update` зв’язавши всі ці три параметри:

```
az aks update \
    --name aks-MicroExample \
    --resource-group rg-MicroExample \
        --attach-acr ACRMicroExample
```

Розгорнемо для прикладу деяку програму збережену в гіт хабі що вже має конфігурацію докера і відповідно, являє собою контейнер. Для цього склонуємо файли додатку в створений образ контейнера

```
git clone https://github.com/hannagm2016/MicroExample.git
```

а потім в Cloud Sell перейдемо в створену дерикторію за допомогою команди `cd`:

```
cd MicroExample
```

Щоб згенерувати апустити контейнер, необхідно виконати команду az acr build з наступними параметрами:

```
az acr build \
    --image Example-website \
    --registry ACRMicroExample \
        --file Dockerfile .
```

Параметр --image додає до образу тег веб-сайту `Example-website`. Образ зображення автоматично надсилається до реєстру після успішного завершення збірки.

Після створення всіх вищеописаних елементів, необхідно додати їм параметри в конфігурацію – уже існуючий маніфест файл `deploy.yaml`

Потрібно описати под, що є шаболоном специфікації. Назва пода буде згенерована автоматично. Щоб деплоймент згрупував поди, необхідний тег `labels` Под обєднує в собі контейнери, кожен з яких містить інформацію під тегом `app:` щодо контейнера. Отже треба оновити файл даними про под:

```
spec:
template: # Шаблон пода деплоймента
  metadata: # Метадані pod
  labels:
        app: Example-website
```

Також в файл конфігурації, після описання поду варто додати інформацію про контейнер, зокрема конфігурацію посилання на розгорнутий сайт:

```
spec:
containers:
- image: <acr_name>.azurecr.io/Example-website
name: contoso-website
```

Обмеження щодо використовуваних ресурсів, зокрема мінімальне та максимальне навантаження процесора, виділену память:

```
resources:
      requests:
        cpu: 50m
        memory: 128Mi
      limits:
        cpu: 200m
              memory: 256Mi
```

А також дані порта, на якому буде сконфігуровано сайт:

```
ports:
      - containerPort: 80
```

```
                name: http
```

Далі необхідно зберегти змінений файл конфігурації, і засосувати зміни за допомогою команди `kubectl apply`

```
kubectl apply -f ./deploy.yaml
```

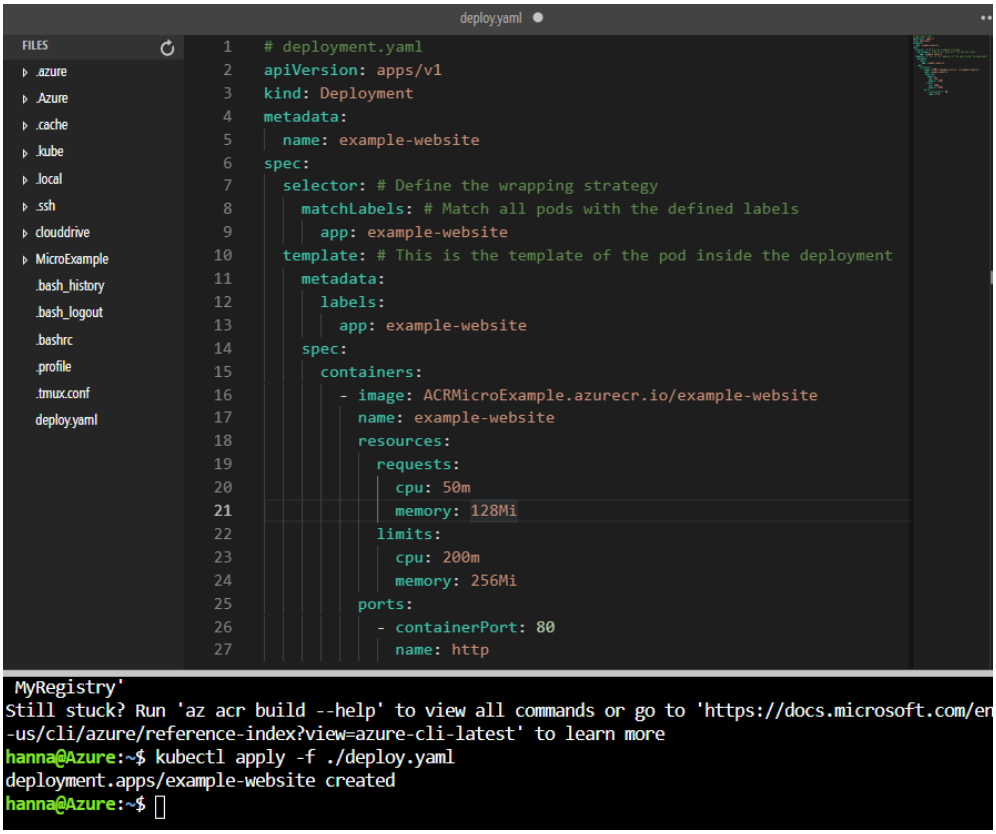

Рисунок 3.3 Застосувнання файлу конфігурації

Щоб перевірити успішність деплойменту, запуску пода і переглянути його назву, треба виконати команду: `kubectl get pods`

Отриманий результат матиме вигляд:

```
NAME                                READY   STATUS             RESTARTS   AGE
example-website-75bfc74bc-cx764     0/1     ImagePullBackOff   0          2m29s
```

3.3 Забезпечення доступу до додатку через мережу

Як і всі ресурси, служби також мають файли маніфестів, що описують їх поведінку.

Створимо у Cloud Shell файл маніфесту для служби Kubernetes під назвою service.yaml, встановимо знову команду `apiVersion`, але вже з інакшим параметром: `v1`, тип встановимо – `Service`, назва – знову таки назва контейнера. Окрім вже відомих команд, необхідно вказати тип сервісу:

```
spec:
  type: ClusterIP
```

Визначити поди, які служба буде групувати, додавши цей атрибут до специфікації:

```
  selector:
    app: example-website
```

Також необхідно вказати дані порту та протоколу маршрутизації

```
ports:
    - port: 80
      name: http
     targetPort: http
     protocol: TCP
```

Далі, обновивши конфігурацію кластера з урахуванням даних сервісного файлу:

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

отримаємо зконфігуровану IP адресу для кластера:

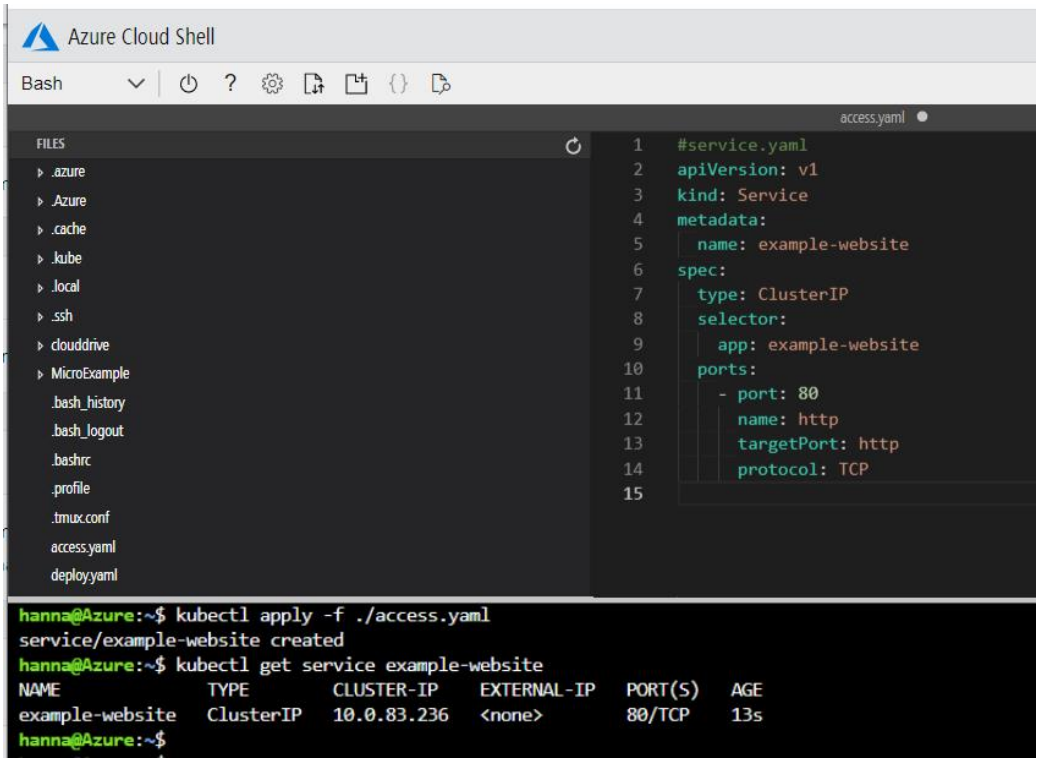

Рисунок 3.4 Встановлення налаштувань мережі

Настіпні налаштування необхідні, щоб дати доступ додатку у інтернет мережу через DNS - потрібно створити контролер входу. Найперше створимо файл маніфесту з правами для входу в мережу, почнемо з основних атрибутів:

```
#ingressExample.yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: Example-website
```

Додамо дані для використання надбудови маршрутизації додатків HTTP для входу, за допомогою атрибута `annotations` всередині розділу метаданих файлу маніфесту і встановимо ключ для kubernetes.io/ingress.class зі значенням addon-http-application-routing.

```
annotations:
    kubernetes.io/ingress.class: addon-http-application-routing
```

Наступним кроком – необхідно визначити і встановити параметри хоста під тегом специфікації, і щоб хост був унікальним і повязаним з ресурсною групою необхідно згенерувати унікальну назву зони для конкретного кластера за допомогою команди `az network dns zone list --output table` в Cloud Shell. Отриманий результат буде містити ZoneName

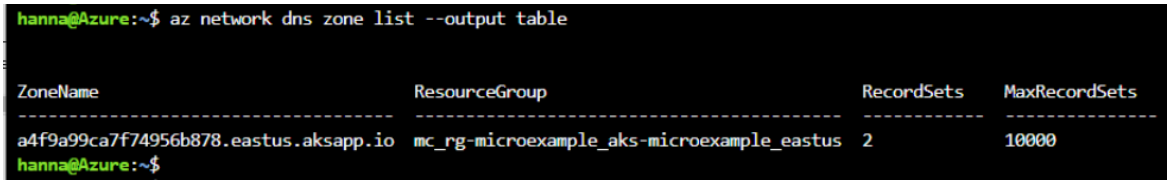

Рисунок 3.5 встановлення параметрів кластера

Скопіювавши значення, додамо його о назви хоста в файл маніфеста:

```
spec:
  rules:
    - host: Example.<zonename>
```

І залишились лише деталі внутрішньої конфігурації та правила входу. Для ключа http додамо данні paths.

```
      http:
        paths:
          - backend: # rules to handle requets
              serviceName: Example-website
              servicePort: http
            path: /
```

Після розгортання файлу маніфесту в кластері можна перевірити успішність деплойменту: `kubectl get ingress Example-website` і отриманий результат мітитиме повну назву хоста і призначену `IP` адресу:

```
NAME              HOSTS                                              ADDRESS   PORTS   AGE
example-website   example.a4f9a99ca7f74956b878.eastus.aksapp.io                80      24s
```

Далі за допомогою команди `az network dns record-set` з параметрами відповідно `resource group та zonename`, отримаємо доступні хости для кожного кластеру. І ці посилання вже будуть доступними з мережі інтернет

`az network dns record-set list -g mc_rg-microexample_aks-microexample_eastus -z a4f9a99ca7f74956b878.eastus.aksapp.io --output table`

```
hanna@Azure:~$ az network dns zone list --output table
ZoneName                            ResourceGroup                                 RecordSets    MaxRecordSets
----------------------------------  --------------------------------------------  ------------  --------------
a4f9a99ca7f74956b878.eastus.aksapp.io  mc_rg-microexample_aks-microexample_eastus  4             10000
hanna@Azure:~$ az network dns record-set list -g mc_rg-microexample_aks-microexample_eastus -z a4f9a99ca7f74956b878.eastus.aksapp.io --output table
Fqdn                                          Name     ProvisioningState    ResourceGroup                                 Ttl
--------------------------------------------  -------  -------------------  --------------------------------------------  ------
a4f9a99ca7f74956b878.eastus.aksapp.io.         @        Succeeded            mc_rg-microexample_aks-microexample_eastus  172800
a4f9a99ca7f74956b878.eastus.aksapp.io.         @        Succeeded            mc_rg-microexample_aks-microexample_eastus  3600
example.a4f9a99ca7f74956b878.eastus.aksapp.io.  example  Succeeded            mc_rg-microexample_aks-microexample_eastus  300
example.a4f9a99ca7f74956b878.eastus.aksapp.io.  example  Succeeded            mc_rg-microexample_aks-microexample_eastus  300
hanna@Azure:~$
```

Перейдемо за посиланням сформованим програмою: example.a4f9a99ca7f74956b878.eastus.aksapp.io.

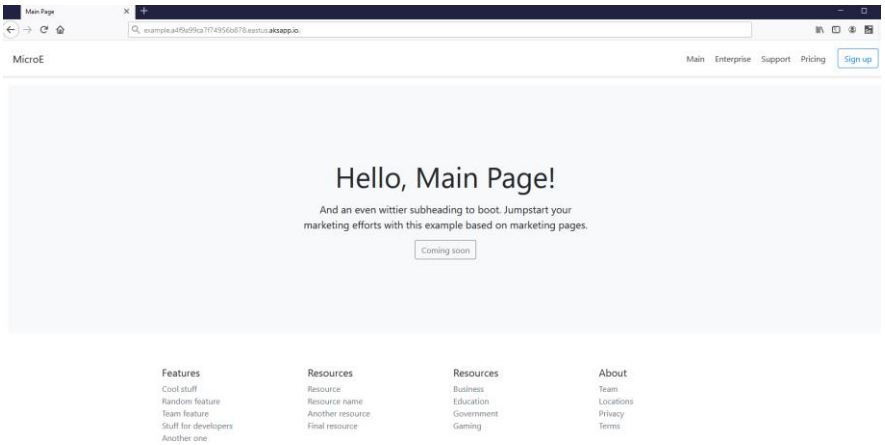

Рисунок 3.6 Запущений додаток в мережі

3.4 Налаштування автомасштабування

Kubernetes підтримує горизонтальне автомасштабування ресурсів, змінюючи число pod в розгортанні в залежності від загрузки ЦП або інших обраних параметрів. Сервер метрик використовується для надання відомостей про використання ресурсів в Kubernetes.

Щоб використовувати інструмент автомасштабування, для всіх контейнерів в групах pod, необхідно визначити ліміти ресурсів. У розгортанні azure-vote-front контейнер зовнішнього застосування вже запрошено 0,20 ресурсів ЦП з лімітом в 0,5 загрузки ЦП. Ці запити і обмеження ресурсів визначені, як показано в фрагменті коду файла маніфесту:

```
resources:
  requests:
     cpu: 200m /min percentage requested
  limits:
     cpu: 500m /limit to scale
```

Також можна встановити автомасштабування за допомогою команди в `Cloud Shell: kubectl autoscale`, для якої потрібно вказати процент завантаження ЦПУ, або ж інший параметр, при досягненні якого відбуватиметься автомасштабування. При чому вказано, яка мінімальна та максимальна кількість подів може бути розгорнута:

```
kubectl autoscale deployment aks-MicroExample --cpu-percent=75 --min=4
--max=9
```

Ще один, більш розширений варіант – створення окремого файлу маніфесту, де можна вказати назву контейнера до якого буде застосовано автомасштабування, та звичайно ж вказати розширені правила, за якими система буде ініціювати розширення чи звуження використання ресурсів:

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: aks-MicroExample
spec:
  maxReplicas: 9
  minReplicas: 4
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: aks-MicroExample
  targetCPUUtilizationPercentage: 75
```

Звичайно, як і будь який інший файл маніфесту, його необхідно застосувати до системи, командою `kubectl apply -f azure- scale - hpa.yaml`, а щоб переконатися в успішності застосування автомасштабування, можна викликати команду: `kubectl get hpa`, ця ж команда також показує журнал масштабування і ті модулі що були додані або видалені.

Отже, як бачимо, процес розгортання контейнерного додатку в хмарному середовищі доволі складний і зовсім неефективний для малих

проектів. Проте має ряд суттєвих переваг зокрема: налаштувавши конфігурацію для розгортання одного кластеру, за такою ж схемою можна працювати з іншими контейнерами, причому немає жодних апаратних обмежень на їх кількість і обєм.

Основною перевагою деплойменту з використанням кластерів є можливість налаштування автомасштабовування, як показано в попередньому підрозділі – це доволі проста задача. Також служба оркестрації дозволяє відслідковувати роботу контейнерів, вести журнал обліку наванажень, керувати безпекою та має багато інших важливих функцій для моніторингу і керування додатком.

Зрозуміло, що неможливо відслідкувати жодних переваг при розгортанні таким методом малого тестового додатку, та для значних сервісів такий метод є оптимальним, оскільки для розгортанням кожного наступного конейнера, потрібно все менше і менше зусиль,

3.5 Рекомендації щодо розгортання контейнерного додатку

Одним з завдань роботи є розроблення рекомендацій для розгортання додатку з використанням служби Kubernetes. З огляду на всі вищеописані методи і технології, найперше – з огляду на складність процесу кластеризаціїї, потрібно визначити межі проекту, і встановити чи потрібна взагалі така методології для розгортання веб сервісу, можливо послуги веб хостингу або просто хмарного хостингу цілком задовольнять потреби проекту. Також, хмарні провайдери пропонують широкий спектр послуг Paas та навіть Saas які можуть значно спростити розробку додатку і прискорити процес отримання готового рішення.

Та з огляду на масштабність проекту, звичайно, деплоймент мікросервісного додатку що вже розміщений в контейнері в системі оркестрації контеєнерів є цілком обгрунтованим та слушним. Загальна схема має складатися з таких етапів:

- Розгортання контейнерів за допомогою системи оркестрації типу Kubernetes, та налаштування необхідної їм конфігурації:
    - Створення кластеру в групі ресурсів;
    - Розміщення одного чи декількох контейнерів в кластері;
    - Створення конфігурації для деплойменту, встановлення параметрів мережі та прав доступу.
    - Деплоймент такого кластеру з використанням вищезазначених конфігурацій.
- Моніторинг і управління роботи кластерів в системі оркестрації

Такий підхід надає проекту високу мобільність, відмовостійкість, продуктивність і швидкодію, а головне, повну автоматичну масштабованість і ефективне використання ресурсів.

Висновки до розділу

В даному розділі було поріняно два підходи до деплойменту веб додатку: з використанням хостингу та хмарних технологій. Виділено головну відмінність що не може бути вирішена в рамках віртуальних хостингів. З використаням офіційної документації служби Azure Kubernetes розроблено оптимізоване розгортання контейнеру та налаштування його автомасштабування. Розроблено рекомендації щодо розгортання кластероного додатку за допомогою системи оркестрації.

ВИСНОВКИ

Проаналізовано деякі можливості що забезпечуються хмарними технологіями; Порівняно підхід для реалізації однієї з таких систем Kubernetes – найбільшими постачальниками хмарних послуг; Розроблено оптимізовану схему роботи з контейнерними додатками, та розглянути механізм розгортання одатку на платформі оркестрації контейнерів.

У першому розділі проаналізовано основні вигоди що забезпечують хмарні обчислення в сучасному інформаційному світі: уніфікація мережі інтернет в хмарну ситему, за рахунок стандартизації звязків між різнотипними елементами; розглянуто основні послуги що надаються хмарними провайдерами: підтримка механізмів віртуалізації та контейнеризації, надання користувачам фізичних ресурсів (приватні, публічні, гібридні хмари), програмних ресурсів (Iaas, Paas, SaaS) та підтримка безпеки даних. Виділено проблему стрімкого масштабуваняя додатків через високу доступність даних.

У другому розділі розглнуті існуючі рішення для швидкого і ефективного розгортання багатокомпонентних сервісів що потенційно можуть розширбватись. Проведено порівняльний аналіз таких рішень від найбільших провайдерів хмарних послуг.

У третьому розділі здійснено оптимізоване розгортання додатку в системі Kubernetes, порівняно цей процес з розгортанням на звичайному веб хостингу. Розроблено конфігурації автомасштабування додатку та сормовано рекомендації щодо вибору методу розгортання додатку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Арундел Д., Домингус Д. - Kubernetes для DevOps: развертывание, запуск и масштабирование в облаке, - Бестселлеры O'Reilly, 2020, 384 с.
2. Загороднюк А.О. Архитектура Docker - Молодой исследователь: вызовы и перспективы: материалы LXIX междунар. науч.-практ. конф. – № 16(69). – М., «Интернаука», 2018. – 482 с.
3. Марко Лукша: Kubernetes в действии, Переводчик: Логунов А. В.: ДМК-Пресс, 2019 г.
4. Сайфан Д. Осваиваем Kubernetes. Оркестрация контейнерных архитектур, Питер. – 2019, 400с.
5. Gens, Frank (2009-10-05). IDC’s New IT Cloud Services Forecast: 2009-2013
6. Nikhil Buduma Fundamentals of Deep Learning. Designing next-generation machine intelligence algorithms. O’Reilly Media, 2017. 277 с
7. Amazon, Microsoft Or Google: Which Is The Best Play On Surging Cloud Infrastructure Demand? [Електронний ресурс] – Режим доступу до ресурсу: https://www.forbes.com/sites/greatspeculations/2020/05/14/amazon-microsoft-or-google-which-is-the-best-play-on-surging-cloud-infrastructure-demand/
8. Comparing Kubernetes Services on AWS vs. Azure vs. GCP [Електронний ресурс] – Режим доступу до ресурсу: https://www.sumologic.com/blog/kubernetes-aws-azure-gcp/
9. Deploy a containerized application on Azure Kubernetes Service [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/learn/modules/aks-deploy-container-app/
10. Developers bring their ideas to life with Docker [Електронний ресурс] – Режим доступу до ресурсу: https://www.docker.com/why-docker

11. DevOps [Електронний ресурс] – Режим доступу до ресурсу: **https://www.ibm.com/cloud/learn/devops-a-complete-guide**

12. Kubernetes blog [Електронний ресурс] – Режим доступу до ресурсу: https://cloudacademy.com/blog/category/kubernetes/

13. Kubernetes documentation [Електронний ресурс] – Режим доступу до ресурсу: https://kubernetes.io/docs/

14. Kubernetes on AWS [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/ru/kubernetes/

15. Kubernetes vs. Docker: A Primer [Електронний ресурс] – Режим доступу до ресурсу: https://containerjournal.com/topics/container-ecosystems/kubernetes-vs-docker-a-primer/

16. Microsoft learn [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/learn/browse/

17. Why (and when) you should use Kubernetes [Електронний ресурс] – Режим доступу до ресурсу: https://hackernoon.com/why-and-when-you-should-use-kubernetes8b50915d97d8

18. Why Google Cloud [Електронний ресурс] – Режим доступу до ресурсу: https://cloud.google.com/why-google-cloud

19. Мікросервісна архітектура [Електронний ресурс] – Режим доступу до ресурсу: - https://medium.com/@IvanZmerzlyi/microservices-architecture-461687045b3d.

20. Начало работы с Azure Kubernetes [Електронний ресурс] – Режим доступу до ресурсу: https://azure.microsoft.com/ru-ru/services/kubernetes-service/#getting-started

21. Основы Кубернетис [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/258443/

22. Хмарні обчислення [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Cloud_computing

Додаток А

Файли Маніфесту для налаштування контейнера

deploy.yaml

```yaml
# deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: example-website
spec:
  selector: # Define the wrapping strategy
    matchLabels: # Match all pods with the defined labels
      app: example-website
  template: # This is the template of the pod inside the deployment
    metadata:
      labels:
        app: example-website
    spec:
      containers:
        - image: ACRMicroExample.azurecr.io/example-website
          name: example-website
          resources:
            requests:
              cpu: 100m
              memory: 128Mi
            limits:
              cpu: 250m
              memory: 256Mi
          ports:
            - containerPort: 80
              name: http
```

ingress.yaml

```yaml
#ingress.yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: example-website
  annotations:
    kubernetes.io/ingress.class: addon-http-application-routing
spec:
  rules:
    - host: example.a4f9a99ca7f74956b878.eastus.aksapp.io
      http:
        paths:
          - backend: # How the ingress will handle the requests
              serviceName: example-website # Which service the request will be forwarded to
              servicePort: http # Which port in that service
            path: / # Which path is this rule referring to
```