

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

Пояснювальна записка

до випускної кваліфікаційної роботи магістра
на тему:

«Дослідження методів підвищення відмовостійкості інформаційної системи на базі кластеру Kubernetes на прикладі Linux кластеру Azure Kubernetes Service»

Виконав

студент II-го курсу, групи КІм-22-1
спеціальності 123 «Комп'ютерна інженерія»



В.В. Миколайчук

Керівник

Завідувач кафедри комп'ютерної інженерії
та кібербезпеки,
кандидат технічних наук,

доцент



А.А. Єфіменко

Консультант

Професор кафедри комп'ютерної інженерії
та кібербезпеки,
доктор технічних наук,
доцент



В.В. Воротніков

Рецензент

Доцент кафедри комп'ютерної інженерії
та кібербезпеки, кандидат технічних наук,
доцент



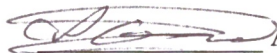
Ю.М. Россінський

м. Житомир – 2023 рік

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА КІБЕРБЕЗПЕКИ
СПЕЦІАЛЬНІСТЬ 123 «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ»

ЗАТВЕРДЖУЮ

Завідувач кафедри
комп'ютерної інженерії
та кібербезпеки

 А.А. Єфіменко

«13» жовтня 2023 року

ЗАВДАННЯ

на випускню кваліфікаційну роботу

Студента **Миколайчука Вадима Володимировича**

Тема роботи: **Дослідження методів підвищення відмовостійкості інформаційної системи на базі кластеру Kubernetes на прикладі Linux кластеру Azure Kubernetes Service**

Затверджена Наказом по університету від «13» жовтня 2023 р. №572/с

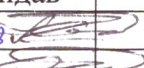
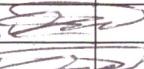
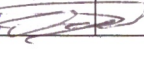
Термін здачі студентом закінченої роботи «11» грудня 2023 р.

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

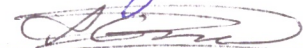
Предметом даного дослідження є методи та технології, спрямовані на підвищення відмовостійкості інформаційних систем, що використовують кластери Kubernetes

Об'єктом дослідження є процес модернізації та оптимізації інформаційної системи на базі Linux кластера в Azure Kubernetes Service з метою підвищення її відмовостійкості. Цей процес включає в себе розробку та імплементацію стратегій, що базуються на кращих практиках із забезпечення надійності і відмовостійкості.

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

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Єфіменко А.А.	13.10.2023 	13.10.2023 
2	Єфіменко А.А.	13.10.2023 	13.10.2023 
3	Єфіменко А.А.	13.10.2023 	13.10.2023 

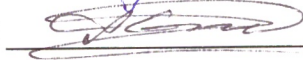
Студент  В.В. Миколайчук

Керівник  А.А. Єфіменко

Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка завдання	13 жовтня 2023 – 13 жовтня 2023	Виконано
2	Пошук, огляд та аналіз літературних джерел щодо контент-маркетингу ринку та Kibernetos	14 жовтня 2023 – 23 жовтня 2023	Виконано
3	Аналіз можливості автоматизації передачі контенту між віртуальними медіа-каналами, середовищу виконання контент-маркетинг-підприємств	24 жовтня 2023 – 26 жовтня 2023	Виконано
4	Аналіз функцій ринку, зокрема моделі та змінних ринку	27 жовтня 2023 – 30 жовтня 2023	Виконано
5	Аналіз Kibernetos, його та Kibernetos Master, компонентів Azure Kibernetos Service	31 жовтня 2023 – 2 листопада 2023	Виконано
6	Аналіз методів підвищення відносності інформаційної системи для Kibernetos	3 листопада 2023 – 4 листопада 2023	Виконано
7	Аналіз методів підвищення відносності інформаційної системи для Kibernetos	5 листопада 2023 – 9 листопада 2023	Виконано
8	Дослідження запропонованих практик	10 листопада 2023 – 13 листопада 2023	Виконано
9	Дослідження практик покращення відносності в Kibernetos	14 листопада 2023 – 20 листопада 2023	Виконано
10	Дослідження практик покращення відносності кластеру AKS	23 листопада 2023 – 25 листопада 2023	Виконано
11	Написання загальної записки	26 листопада 2023 – 4 грудня 2023	Виконано
12	Підготовка презентації та виступу	5 грудня 2023 – 6 грудня 2023	Виконано
13	Отримання вогнища керівника та рецензії на кваліфікаційну роботу	11 грудня 2023	Виконано
15	Захист	Згідно розкладу	Виконано

Студент  В.В. Миколайчук

Керівник  А.А. Єфіменко

АНОТАЦІЯ

Кваліфікаційна робота містить __ сторінок, __ ілюстрації, __ таблиць, список використаних джерел із __ найменувань.

Метою атестаційної магістерської роботи є аналіз та розробка методів підвищення відмовостійкості інформаційних систем з використанням Kubernetes. Це буде демонструватися на прикладі Linux кластеру в Azure Kubernetes Service (AKS) з інфраструктурою, наближеною до продю, що буде відображатись на виборі типу розміщення інфраструктури та технологій, що використовуються.

В ході дослідження проведено аналіз будови кластеру Kubernetes, а також Azure Kubernetes service та принципів розміщення конйнеризованих додатків в даних ситемах.

В роботі запропонований підхід для покращення рівня відмовостійкості інформаційної системи в даних середовищах.

Ключові слова: KUBERNETES, AZURE KUBERNETES SERVICE, КОНТЕЙНЕР, ВІДМОВОСТІЙКІСТЬ

ANNOTATION

The qualification work contains __ pages, __ illustrations,, __ tables, a list of __ references.

The purpose of the work is the analysis and development of methods of increasing the fault tolerance of information systems using Kubernetes. This will be demonstrated using the example of a Linux cluster in Azure Kubernetes Service (AKS) with near-production infrastructure, which will be reflected in the type of infrastructure deployment choices and technologies used.

During the study, the structure of the Kubernetes cluster, as well as the Azure Kubernetes service and the principles of placing containerized applications in the data on the sites were analyzed.

The paper proposes an approach to improve the failure rate of the information system in these environments.

Keywords: KUBERNETES, AZURE KUBERNETES SERVICE, CONTAINER, FAILOVER

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	4
1. ТЕОРЕТИЧНІ ОСНОВИ ЗАСТОСУВАННЯ ХМАРНИХ КОНТЕЙНЕРИЗОВАНИХ РІШЕНЬ	7
1.1. АНАЛІЗ ТЕХНОЛОГІЇ КОНТЕЙНЕРИЗАЦІЇ.....	7
1.1.1. ПЕРЕВАГИ КОНТЕЙНЕРІВ ПЕРЕД ВІРТУАЛЬНИМИ МАШИНАМИ	7
1.1.2. СЕРЕДОВИЩЕ ВИКОНАННЯ КОНТЕЙНЕРІВ	9
1.1.3. ЖИТТЄВИЙ ЦИКЛ КОНТЕЙНЕРУ	12
1.2. ОГЛЯД ХМАРНИХ РІШЕНЬ.....	14
1.2.1. ХМАРНІ МОДЕЛІ	14
1.2.2. ХМАРНІ ПРОВАЙДЕРИ	16
1.3. ОГЛЯД ПЛАТФОРМИ KUBERNETES.....	23
1.3.1. ПРИЗНАЧЕННЯ KUBERNETES	23
1.3.2. СТРУКТУРА KUBERNETES	24
1.3.3. WINDOWS ТА LINUX КЛАСТЕРИ KUBERNETES	28
1.3.4. БАЗОВІ КОМПОНЕНТИ AZURE KUBERNETES SERVICE	29
ВИСНОВКИ ДО ПЕРШОГО РОЗДІЛУ	37
2. АНАЛІЗ МЕТОДІВ ПІДВИЩЕННЯ ВІДМОВОСТІЙКОСТІ ІНФОРМАЦІЙНОЇ СИСТЕМИ	39
2.1. АНАЛІЗ МЕТОДІВ ПІДВИЩЕННЯ ВІДМОВОСТІЙКОСТІ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ KUBERNETES	39
2.2. АНАЛІЗ МЕТОДІВ ПІДВИЩЕННЯ ВІДМОВОСТІЙКОСТІ ІНФОРМАЦІЙНОЇ СИСТЕМИ LINUX КЛАСТЕРУ AKS.....	48
ВИСНОВКИ ДО ДРУГОГО РОЗДІЛУ	54
3. ДОСЛІДЖЕННЯ ЗАПРОПОНОВАНИХ ПРАКТИК	55
3.1. ДОСЛІДЖЕННЯ ПРАКТИК ПОКРАЩЕННЯ ВІДМОВОСТІЙКОСТІ В KUBERNETES	55
3.2. ДОСЛІДЖЕННЯ ПРАКТИК ПОКРАЩЕННЯ ВІДМОВОСТІЙКОСТІ КЛАСТЕРУ AZURE KUBERNETES SERVICE	59
ВИСНОВКИ ДО ТРЕТЬОГО РОЗДІЛУ	63
ВИСНОВКИ.....	64
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ	65

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ІС – інформаційна система
AKS – Azure Kubernetes Service
ВМ – віртуальна машина
ОС – операційна система
CLI – інтерфейс командного рядка
CNCF – Фонд хмарних обчислень
OCI – Ініціатива відкритих контейнерів
DNS – Domain Name System
SaaS – Software as a Service
PaaS – Platform as a Service
IaaS – Infrastructure as a Service
AWS – Amazon Web Services
GCP – Google Cloud Platform
ІІ – штучний інтелект
CSP – Cloud Solution Provider
API – application programming interface
RaMP – Rapid Migration Program
VMSS – virtual machine scale set
CA – cluster autoscaler
PDB – Pod disruption budget
SLA – Service Level Agreements
PVC – PersistentVolumeClaim
UDR – User-defined routes
AGIC – Application Gateway Ingress Controller

ВСТУП

Сучасні бізнес-вимоги стимулюють розвиток інформаційних технологій, зокрема в аспекті оптимізації витрат на інфраструктуру. Ключовим елементом у цьому контексті стали контейнери, які дозволяють забезпечувати високу надійність та безперебійність систем при зменшенні оперативних витрат. Використання оркестрації контейнерів через платформу Kubernetes значно спростило управління складними інфраструктурами, зробивши цю технологію стандартом для багатьох компаній. Спеціалісти, що працюють з інформаційними системами, повинні володіти глибокими знаннями Kubernetes для побудови надійних систем, що задовольняють високі стандарти відмовостійкості.

Мета цього дослідження полягає в аналізі та розробці методів підвищення відмовостійкості інформаційних систем з використанням Kubernetes. Це буде демонструватися на прикладі Linux кластеру в Azure Kubernetes Service (AKS) з інфраструктурою, наближеної до продю, що буде відображатись на виборі типу розміщення інфраструктури та технологій, що використовуються.

Об'єктом дослідження є процес модернізації та оптимізації інформаційної системи на базі Linux кластера в Azure Kubernetes Service з метою підвищення її відмовостійкості. Цей процес включає в себе розробку та імплементацію стратегій, що базуються на кращих практиках із забезпечення надійності і відмовостійкості.

Предметом даного дослідження є методи та технології, спрямовані на підвищення відмовостійкості інформаційних систем, що використовують кластери Kubernetes.

Методологічний підхід базується на аналізі найкращих практик та використанні системного аналізу для створення ефективної інфраструктури Kubernetes. Це включає моделювання інфраструктури, оцінку поточного стану систем, і планування удосконалень із зосередженням на метриках відмовостійкості та доступності. Також будуть застосовані емпіричні методи, такі як тестування на продуктивність та стрес-тестування, для валідації теоретичних моделей та підходів.

Наукова новизна полягає у:

аналізі існуючих методів покращенні відмовостійкості інформаційних систем та їх покращенні;

виявленні та аналізі нових патернів відмов інформаційних систем в Kubernetes;

проведенні експериментального дослідження з використанням Azure Kubernetes Service (AKS);

розробці логічного продовженні існуючих методів покращення відмовостійкості ІС на базі Kubernetes з подальшим розвитком даної роботи;

розробці конфігураційних файлів або частин конфігурацій, що допоможуть досягнути поставленої у роботі, мети.

Практичне значення полягає у побудові фреймворку для покращення відмовостійкості інформаційної системи на базі Kubernetes на основі проведеного дослідження. Використовуючи дані результати інженери, що можуть бути залучені до аналогічного завдання, будуть мати змогу використати дані практики.

1.ТЕОРЕТИЧНІ ОСНОВИ ЗАСТОСУВАННЯ ХМАРНИХ КОНТЕЙНЕРИЗОВАНИХ РІШЕНЬ

1.1. АНАЛІЗ ТЕХНОЛОГІЇ КОНТЕЙНЕРИЗАЦІЇ

1.1.1.ПЕРЕВАГИ КОНТЕЙНЕРІВ ПЕРЕД ВІРТУАЛЬНИМИ МАШИНАМИ

На сьогодні досі досить часто для розміщення веб-додатків використовуються віртуальні машини. Розглянемо що ж таке віртуальні машини та що таке контейнери, та які переваги та особливості застосування має кожна з технологій. Віртуальна машина — модель обчислювальної машини, створеної шляхом віртуалізації обчислювальних ресурсів: процесора, оперативної пам'яті, пристроїв зберігання та вводу і виводу інформації [1]. Контейнер — це ізольоване, легковагове сховище для запуску додатку на операційній системі хоста. Контейнери побудовані на базі ядра операційної системи хоста (яке можна уявити як приховану систему операційної системи) і містять лише додатки та деякі легковагові API та сервіси операційної системи, які працюють у режимі користувача. На сьогодні є велика кількість хмарних (і не тільки хмарних) обчислювальних сервісів, що дозволяють побудувати інфраструктуру для майже будь-якої інформаційної системи [2].

Контейнери значно ефективніше використовують ресурси порівняно з віртуальними машинами. Це досягається за рахунок спільного використання ядра ОС хоста, що знижує загальні вимоги до системних ресурсів. Контейнеризація дозволяє мінімізувати витрати на обслуговування численних віртуальних машин, кожна з яких потребує власної операційної системи. В результаті, контейнери займають значно менше дискового простору та оперативної пам'яті, що підвищує щільність розгортання додатків і знижує вартість використання обладнання.

Швидкість розгортання та запуску контейнерів є однією з їх ключових переваг. Контейнерам не потрібно завантажувати власну операційну систему, тому вони можуть запускатися майже миттєво. Це робить контейнери ідеальними для середовищ, де необхідні швидкі оновлення та масштабування, як

наприклад, у DevOps практиках, де швидкість випуску та оновлення програмного забезпечення є критично важливою.

Контейнери забезпечують високу ступінь консистентності в різних середовищах. Це досягається завдяки тому, що контейнер включає в себе усе необхідне для роботи додатку: код, бібліотеки, системні інструменти та налаштування середовища. В результаті, додаток, який працює в контейнері, буде поводитися однаково незалежно від того, де він запущений: на локальному комп'ютері розробника, тестовому середовищі, чи у продакшені на хмарному сервері.

Портабельність контейнерів значно спрощує процес міграції та розгортання додатків в різних середовищах. Оскільки контейнери ізолюють додаток від середовища, вони можуть легко переміщатися між різними хмарними провайдерами або між хмарою та локальним середовищем, не вимагаючи будь-яких змін у коді або конфігурації.

Оркестраційні інструменти, такі як Kubernetes, дозволяють автоматизувати розгортання, масштабування та управління контейнерами. Це підвищує продуктивність DevOps команд, забезпечуючи швидке реагування на зміни в потребах бізнесу та автоматизацію рутинних процесів. Застосування оркестрації контейнерів також сприяє більш ефективному використанню ресурсів, оскільки система автоматично розподіляє навантаження та оптимізує використання інфраструктури.

Хоча контейнери не забезпечують такої повної ізоляції, як віртуальні машини, вони все одно пропонують достатній рівень безпеки для багатьох застосунків. Сучасні технології контейнерів включають різні засоби безпеки, такі як простори імен та контроль доступу на основі ролей, які допомагають ізолювати додатки один від одного.

Ізоляція в контейнерах важлива для запобігання взаємодії між додатками та обмеження можливості поширення вразливостей. Це особливо важливо в багатокористувацьких середовищах, де різні додатки можуть мати різні рівні довіри та вимоги до безпеки.

Контейнери пропонують значні переваги у вигляді ефективності ресурсів, швидкості, консистентності, портабельності, легкості управління, а також певного рівня безпеки та ізоляції. Ці переваги роблять їх ідеальним вибором для сучасних розробників та інфраструктурних архітекторів, особливо в середовищах, де швидкість, гнучкість та масштабованість є ключовими факторами.

1.1.2.СЕРЕДОВИЩЕ ВИКОНАННЯ КОНТЕЙНЕРІВ

Середовище виконання контейнерів — це програмний компонент, який може запускати контейнери в операційній системі хоста. У контейнеризованій архітектурі середовище виконання контейнера відповідає за завантаження образів контейнера зі репозиторіїв, моніторинг локальних системних ресурсів, ізоляцію системних ресурсів для використання контейнеру та керування його життєвим циклом.

Більшість середовищ виконання контейнерів зазвичай працюють разом із оркестраторами контейнерів. Оркестратор відповідає за керування кластерами контейнерів, відповідає за масштабування, мережу та безпека. Механізм контейнерів бере на себе відповідальність за керування окремими контейнерами, що працюють на кожному обчислювальному вузлі в кластері.

Типовими прикладами середовищ виконання контейнерів є runC, containerd, Docker і контейнери Windows. Існує три основні типи середовищ виконання контейнерів — середовища виконання низького рівня, середовища виконання високого рівня та середовища виконання із ізольованим програмним середовищем або віртуалізовані середовища виконання.

Open Container Interface (OCI) — це проект Linux Foundation, започаткований компанією Docker, мета якого — забезпечити відкриті стандарти для контейнерів Linux. Основним проектом із відкритим кодом, розробленим OCI, є runC, випущений у 2015 році. runC — це низькорівневе середовище виконання контейнерів, яке реалізує специфікацію OCI. Він є основою для багатьох інших механізмів виконання контейнерів.

OCI надає специфікації середовища виконання контейнерів. Середовища виконання, реалізовані відповідно до специфікацій OCI, називаються низькорівневими середовищами виконання, тому що основна увага приділяється управлінню життєвим циклом контейнера.

Нативні низькорівневі середовища виконання контейнерів відповідають за створення та запуск контейнерів. Після запуску контейнерного процесу середовище виконання контейнерів не потрібна для виконання інших завдань. Це тому, що низькорівневі середовища виконання абстрагують примітиви Linux і не призначені для виконання додаткових завдань.

Найпопулярніші низькорівневі середовища виконання включають:

`runC` — створений Docker і OCI. Тепер це де-факто стандарт в класі низькорівневих середовищ виконання контейнерів. `runC` написано мовою Go. Він підтримується `mobu` — проектом Docker з відкритим кодом.

`crun` — реалізація OCI під керівництвом Red Hat. `crun` написаний мовою C. Він розроблений, щоб бути легким і продуктивним, і був одним із перших середовищ виконання, які підтримували `cgroups v2`.

`containerd` — демон з відкритим вихідним кодом, який підтримується Linux і Windows, який полегшує керування життєвими циклами контейнера через запити API. API контейнера додає рівень абстракції та покращує портативність контейнера.

Приклади високорівневих середовищ виконання контейнерів:

Docker (`containerd`) — провідна контейнерна система, яка пропонує повний набір функцій, як безкоштовних, так і платних. Це середовище виконання контейнерів за замовчуванням (мається на увазі `containerd`) в Kubernetes, що надає специфікації образів, інтерфейс командного рядку (CLI) і службу створення образу контейнеру.

CRI-O — реалізація середовища виконання контейнерів Kubernetes (CRI) з відкритим вихідним кодом, що пропонує легку альтернативу `rkt` та Docker. Дозволяє запускати контейнери за допомогою OCI-сумісних середовищ

виконання, забезпечуючи підтримку в першу чергу runC і Kata (хоча можливо підключити будь-яке OCI-сумісне середовище виконання).

Контейнери Windows і контейнери Hyper-V — дві легкі альтернативи віртуальним машинам Windows, доступні у Windows Server. Контейнери Windows пропонують абстракцію (подібно до Docker), а Hyper-V — віртуалізацію. Контейнери Hyper-V легко переносяться, оскільки кожен з них має власне ядро, тож є можливість запускати несумісні програми у вашій хост-системі [3].

Важливо також додатково зазначити та пояснити зв'язок containerd (який являється середовищем виконання контейнерів за замовчуванням в Kubernetes) та Docker (середовище виконання контейнерів, що часто використовується розробниками). Розглянемо ще раз, що вміє containerd та його історію.

containerd — це середовище виконання контейнерів, яке керує життєвим циклом контейнера на фізичній або віртуальній машині (хості). Це процес-демон, який створює, запускає, зупиняє та знищує контейнери. Він також може завантажувати образи контейнерів з реєстрів контейнерів, монтувати сховище і вмикати мережу для контейнера.

containerd був створений компанією Docker і переданий Cloud Native Computing Foundation (CNCF). Це підмножина оригінального Docker Engine, яка має більшість функціональних можливостей Docker для запуску контейнерів, роботи зі сховищем та керування образами, але не має багато функцій, орієнтованих на розробників, що робить її придатною для широкомасштабного використання у складі контейнерних оркестраторів, таких як Kubernetes. Docker Engine використовує containerd у своїй роботі.

Docker Engine є прикладом високорівневого контейнерного движка, що використовується переважно розробниками, в той час як containerd є прикладом низькорівневого контейнерного движка, що має лише базову функціональність, придатну для використання автоматизованими механізмами.

containerd повністю підтримує стандарти, встановлені OCI, організацією з відкритим управлінням, зосередженою на найкращих практиках у сфері

контейнерів. Він також має підтримку container runtime interface (CRI) – специфікацію Kubernetes, розроблену для того, щоб дозволити декільком середовищам виконання контейнерів працювати у складі кластера. Kubernetes може використовувати containerd та інші низькорівневі середовища виконання контейнерів, що підтримують OCI, для запуску контейнерів на вузлах Kubernetes [4].

1.1.3.ЖИТТЄВИЙ ЦИКЛ КОНТЕЙНЕРУ

Розглянемо життєвий цикл контейнеру на прикладі Docker-контейнерів. Для ефективної роботи з контейнерами (в тому числі і Docker-контейнерами) важливо розуміти різні етапи існування контейнера, які називаються його життєвим циклом. Життєвий цикл контейнера Docker складається з декількох станів, таких як created (створений), running (запущений), paused (зупинений), exited (завершений) і dead/deleted (видалений).

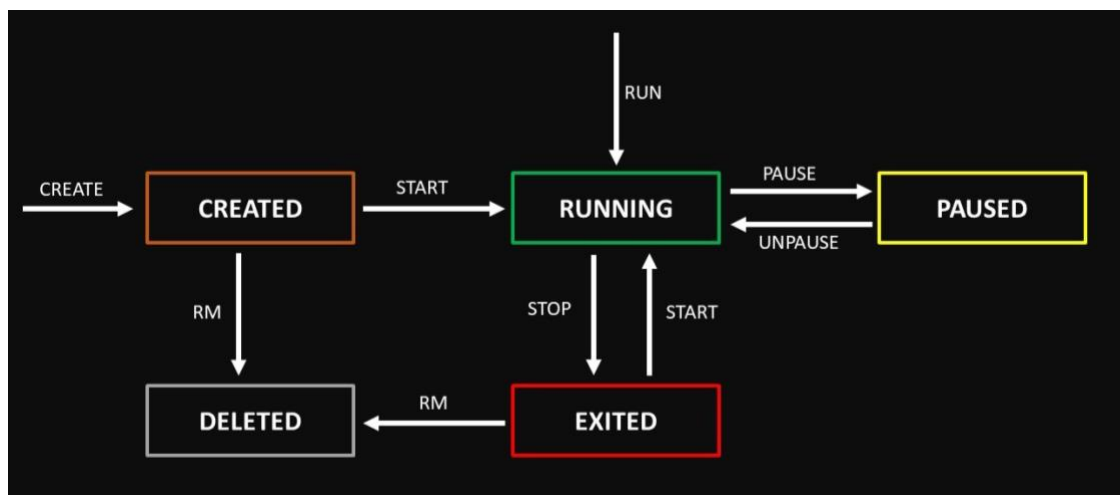


Рисунок 1.1.3 — Ілюстрація життєвого циклу контейнеру

Коротко опишемо кожен зі станів:

Created — це початковий стан Docker-контейнеру після його створення, але до того, як його було запущено.

Running — стан, у якому створений контейнер запущено. У цьому стані контейнер власне виконує свої процеси.

Paused — якщо контейнер призупинено, виконання його процесів тимчасово зупиняється, але він не завершується.

Exited — якщо основний процес контейнера завершується, контейнер зупиняється і переходить у стан exited.

Dead — якщо контейнер не вдається запустити, він переходить у стан "мертвий". Контейнери у цьому стані не можуть бути перезапущені і повинні бути створені заново.

Created — це перший стан у життєвому циклі контейнера. Цей стан означає, що контейнер створено, але не запущено. Створити контейнер можливо за допомогою команди `docker create`.

Коли ми створюємо контейнер Docker, до шарів тільки для читання (R/O) образу, що використовується для побудови свого, додається шар читання і запису(R/W). Це готує контейнер до запуску, наприклад, на цьому етапі завантажуються образ, налаштовуються змінні оточення, встановлюється точка входу тощо.

Важливо зазначити, що створення контейнера не призводить до його автоматичного запуску. Однак, ми можемо встановити всі необхідні параметри конфігурації під час процесу створення, такі як ліміти процесора і пам'яті, образ контейнера і т.д.. Коли контейнер знаходиться у цьому стані, конфігурацію контейнера також можна оновити за допомогою команди `docker update`.

Це означає, що є можливість створити контейнер один раз з усіма необхідними параметрами і запустити його пізніше без необхідності вказувати їх знову. Також слід зазначити, що у цьому стані ресурси не виділяються.

Running — стан, при якому створений контейнер запущено. Команди, описані в маніфесті контейнеру виконуються одна за одною.

При запуску контейнеру Docker налаштовує виділення ресурсів, які необхідні контейнеру (мережеві ресурси, ресурси процесору, оперативну пам'ять). Потім Docker створює середовище для роботи контейнеру. Як тільки контейнер готовий до роботи, він починає виконувати інструкції, вказані в маніфесті.

Killed/exited — це стан, в якому контейнер не виконується. Є ряд причин, через які контейнер переходить у стан exited. Наприклад, якщо процес, що виконувався в контейнері завершився або коли процес завершився з помилкою.

1.2.ОГЛЯД ХМАРНИХ РІШЕНЬ

1.2.1.ХМАРНІ МОДЕЛІ

Інфраструктуру для Kubernetes можливо розмістити різними способами. Розглянемо хмарні моделі, які можна використати для цього. Існують три хмарні моделі: private, public та hybrid.

Приватна хмара (private cloud) надає хмарні послуги, доступ до яких надається які пропонуються через Інтернет або приватну внутрішню мережу з обмеженим доступом (на сьогодні є велика кількість механізмів, що дозволяє це зробити). Приватні хмарні обчислення, які також називають внутрішньою або корпоративною хмарию, надають організаціям багато переваг загальнодоступної хмари, включаючи самообслуговування, масштабованість і еластичність. Крім того, приватні хмари забезпечують вищий рівень безпеки та конфіденційності при використанні брандмауєру та ізольований хостинг, що гарантує, що та конфіденційні дані недоступні для сторонніх. Одним із недоліків є те, що ІТ-відділ компанії несе відповідальність за обслуговування та витрати на підтримку приватної хмари. Таким чином, приватні хмари вимагають тих самих витрат на персонал, управління та технічне обслуговування, що й традиційний центр обробки даних [5].

Публічна хмара (public cloud) є найпопулярнішим типом послуги, що надають провайдери хмарних послуг. Хмарні ресурси (наприклад, сервери та сховища даних) належать і управляються власне постачальником хмарних послуг і надаються у користування через Інтернет. У публічній хмарі все обладнання, програмне забезпечення та інша допоміжна інфраструктура належить і управляється постачальником хмари. Microsoft Azure є прикладом публічної хмари.

У загальнодоступній хмарі клієнти користуються серверами, сховищем і мережним обладнанням, що може використовуватись одночасно для надання

послуг одночасно кільком «орендарям», а доступ користувачі отримують через облікові записи з відповідного рівня доступом

Переваги публічних хмар:

- менші витрати — не потрібно купувати обладнання чи програмне забезпечення, і потрібно платити лише за послугу, якою користуєтеся;
- не потрібно виконувати оновлення апаратного забезпечення самостійно — постачальник послуг забезпечує обслуговування;
- майже необмежена масштабованість — доступні ресурси на вимогу для задоволення потреб;
- висока надійність — внутрішні механізми, що використовують хмарні провайдери для своєї інфраструктури забезпечують високу надійність.

Гібридна хмара (hybrid cloud) — це обчислювальне середовище, яке поєднує локальний дата-центр (типу приватної хмари) з публічною хмарию, дозволяючи розміщувати дані або робочі навантаження в будь-якій або в публічній, або в приватній площині. Багато організацій обирають гібридну хмару через вимоги, що диктує бізнес або для дотримання регуляторних вимог [6].

Розглядаючи та оцінюючи публічні хмару, дуже важливо розуміти модель розподіленої відповідальності, а також те, які завдання вирішує хмарний провайдер, а які користувач послуг. Відповідальність за робоче навантаження варіюється залежно від того, продуктом якого типу доводиться користуватись Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS) або ж локальному центрі обробки даних (ЦОД).

Власник дата-центру повністю несе відповідальність за його інфраструктуру та підтримку (як програмну, так і апаратну). В той же час при використанні публічної хмари, провайдер хмарних послуг бере частину зобов'язань з підтримки на себе.

Зони відповідальності у моделі розподіленої відповідальності

Відповідальність	SaaS	PaaS	IaaS	On-premise
Інфраструктура (фізичні сервери, мережа)	П	П	П	К
Фізична безпека дата-центрів	П	П	П	К
Управління мережею	П	П	С	К
Віртуалізація	П	П	С	К
Операційна система	П	П	К	К
Мережева безпека	П	С	К	К
Додатки	П	К	К	К
Ідентифікація та доступ	С	С	К	К
Захист даних	С	С	К	К

П – провайдер, К – клієнт (споживач послуг), С – спільна відповідальність

1.2.2.ХМАРНІ ПРОВАЙДЕРИ

Звіти “Магічний квадрант” є підсумком ретельних, заснованих на фактах досліджень в межах конкретних ринків. Звіти дають широке уявлення про позиції провайдерів на тих ринках, які мають високі темпи зростання і чітку диференціацію провайдерів. Так у звіті «Magic Quadrant for Strategic Cloud Platform Services» Gartner визначає стратегічні послуги хмарних платформ як такі, що є стандартизовані, автоматизовані, загальнодоступні хмарні пропозиції, що надають змогу інтегрувати інфраструктурні сервіси (IaaS) (наприклад, обчислювальні, мережеві та сховища зберігання даних), платформні сервіси (PaaS) (наприклад, керовані сервіси додатків і даних) та послуги трансформації (програми/ресурси, які допомагають клієнтам впроваджувати хмаро-орієнтовані моделі надання ІТ-послуг). Інфраструктурні та платформні сервіси можуть розміщуватися як на стороні постачальника хмарних послуг, так і в дата-центрах клієнтів; однак вони належать провайдеру послуг. Сервіси повинні бути еластично масштабованими, їх використання має бути прозорим і управління сервісами має бути доступним через веб-інтерфейси та програмовані API. Програми трансформації можуть реалізовуватися за допомогою автоматизованих інтерфейсів самообслуговування та керованої взаємодії, яку забезпечують команди/партнери по роботі з клієнтами.



Рисунок 1.3.2.1 – Позиції провайдерів у «Magic Quadrant for Strategic Cloud Platform Services»

Розглянемо компанії трьох очевидних лідерів даного рейтингу та їх продуктів – найпопулярніших хмарних провайдерів для подальшого аналізу у нашій роботі, а саме Amazon Web Services (AWS) з їх однойменним хмарним провайдером, Microsoft та їх Microsoft Azure та Google з їх Google Cloud Platform.

Amazon Web Services (AWS), дочірня компанія Amazon.com, є лідером у цьому магічному квадранті. AWS є комплексним постачальником хмарної інфраструктури як послуги (IaaS) та платформи як послуги (PaaS) для найрізноманітніших потреб ІТ. Для корпоративних клієнтів AWS підтримує цифрову трансформацію в першу чергу за допомогою програм міграції та модернізації робочих навантажень. AWS не зосереджується на корпоративному SaaS, вважаючи за краще вирішувати більшість випадків використання бізнес-додатків через свою партнерську екосистему.

Зберігаючи провідну частку на світовому ринку IaaS+PaaS, що перевищує 40%, AWS продовжує розширюватися у стабільно прибутковий спосіб. Ключовими драйверами зростання є сервіси баз даних та аналітики, а також штучний інтелект/машинне навчання. Нещодавно AWS оголосила про партнерство з Anthropic для просування своєї стратегії генеративного ШІ.

Сильні сторони:

Ширина та глибина послуг IaaS та PaaS: AWS пропонує широкий спектр послуг хмарної інфраструктури та платформ, які часто є більш досконалішими, ніж у конкурентів. Основні послуги AWS зазвичай безпосередньо впливають на ринок, встановлюючи стандарти та ціни, які приймають інші провайдери.

Інновації в області апаратного забезпечення: AWS був першим хмарним провайдером, який впровадив прискорення продуктивності та безпеку за рахунок апаратного розвантаження за допомогою системи Nitro (гіпервізор), а також першим, хто розробив власні чіпсети на базі ARM. Сьогодні ці технології широко використовуються в інфраструктурі AWS, що дає AWS потенційну перевагу у вартості та ефективності порівняно з провайдерами, які покладаються виключно на сторонні компоненти сторонніх постачальників.

Велика, потужна партнерська екосистема: AWS має розвинені партнерські канали системних інтеграторів та постачальників керованих послуг (MSP – managed service provider), а партнерські програми AWS приділяють значну увагу технічній підтримці партнерів. AWS Marketplace – найбільший і найбільш повнофункціональний цифровий ринок хмарного програмного забезпечення сторонніх розробників у галузі.

Застереження:

Брак інструментів та інструкцій для мультихмарних сценаріїв: серед найбільших гіперскейлерів AWS вирізняється відсутністю інструментів і сервісів, які безпосередньо стосуються мультихмарних сценаріїв використання. Результати опитування Gartner показують, що 76% корпоративних клієнтів IaaS і PaaS сьогодні покладаються на більш, ніж одного постачальника послуг публічної хмари.

Надмірний акцент на "lift and shift": у відділах продажів AWS та Програмі прискорення міграції AWS (MAP – Migration Acceleration Program) часто наголошується на підході "переходьте зараз, модернізуйте пізніше", який зосереджується на тому, щоб клієнти якомога швидше перейшли на AWS. Клієнти, які не модернізують свої робочі навантаження після міграції, можуть

опинитися в ситуації, коли їм доведеться підтримувати застарілі системи в AWS, не отримавши очікуваної гнучкості та економічних переваг.

Потенційна конкуренція з партнерами: AWS має історію впровадження послуг, які конкурують з партнерами по екосистемі, що може викликати у клієнтів сумніви щодо того, у кого купувати. AWS пропонує багато керованих сервісів для популярних пакетів з відкритим кодом, інструменти міграції та стимули для переходу клієнтів з партнерських продуктів баз даних, а також конкурує з власним каналом обслуговування за допомогою таких пропозицій, як AWS Managed Services. Microsoft є лідером у магічному квадранті. Microsoft Azure ефективна у всіх випадках використання, але особливо добре підходить для організацій, орієнтованих на використання продуктів Microsoft, з гібридною хмарною стратегією. Широка глобальна партнерська мережа також робить Microsoft логічним вибором для клієнтів малого та середнього бізнесу з усього світу.

Корпорація Microsoft інвестує в усі сфери IaaS, PaaS і SaaS. Azure лежить в основі чотирьох додаткових хмарних пропозицій: Microsoft 365, Dynamics 365, GitHub і Power Platform. Крім того, корпорація Microsoft інвестує значні кошти в інноваційну компанію OpenAI, яка займається розробкою штучного інтелекту. Такий комплексний та крос-інтегрований підхід може бути привабливим для клієнтів, які прагнуть працювати з одним постачальником, але складним для клієнтів, які сподіваються використовувати окремі служби Azure незалежно від повного портфоліо Microsoft.

Сильні сторони:

Гібридна та мультихмарна трансформація: корпорація Microsoft систематично розширює можливості Azure Resource Manager для керування широким спектром хмарних, мультихмарних і локальних ресурсів, використовуючи Azure Arc як міст для з'єднання. Завдяки рішенням з підтримкою Arc для VMware vSphere, а також для апаратних серверів традиційних OEM-виробників, клієнти Microsoft можуть використовувати Azure для адміністрування та захисту як приватних хмарних, так і нехмарних ресурсів.

Можливості для розробників: не всі служби Microsoft для розробників є "найкращими у своєму класі", однак, якщо взяти їх разом, портфоліо Azure для розробників добре зарекомендувало себе в широкому спектрі випадків використання, включаючи розробку хмарних додатків, модернізацію застарілих програм, обробку даних та штучний інтелект. Крім того, хоча інші сервіси Microsoft, орієнтовані на розробників (Power Platform та GitHub), знаходяться поза межами цього магічного квадранту, інтеграцію Azure з цими ресурсами не можна ігнорувати.

Стратегічні партнерства: Корпорація Microsoft виявила бажання і змогла домовитися про стратегічне партнерство Azure з широким колом лідерів галузі, включаючи IT-лідерів, таких як SAP, Oracle, Dell і VMware; вертикальних лідерів галузі, таких як Eric і AT&T; а також нових компаній-лідерів, таких як OpenAI. Така гнучкість важлива для підприємств, які прагнуть здійснити цифрову трансформацію, використовуючи технології від декількох провідних постачальників.

Застереження:

Постійні проблеми зі стійкістю та безпекою: незважаючи на постійні зусилля корпорації Microsoft, збій у роботі Azure в Сінгапурі в січні 2023 року продемонстрував, що непослідовні операційні практики та архітектури відмовостійкості залишаються проблемою в деяких регіонах. Крім того, за останні два роки дослідники оприлюднили серйозні вразливості безпеки в Azure. Наразі корпорація Microsoft працює над покращенням своєї системи безпеки та розвиває культуру, в якій безпека є першочерговим пріоритетом.

Міжсервісні залежності та технологічна прив'язка: багато клієнтів Gartner повідомляють про розчарування тим, що служби Azure спеціально створені для "найкращої роботи" з іншими службами Azure, що ускладнює контроль над витратами. Часто фінансові наслідки використання нової служби Azure лише для задоволення потреб іншої не є зрозумілими для клієнтів, які потім бачать, що їх витрати зростають у непередбачуваний спосіб.

Нерівномірна якість партнерів та уніфікованої підтримки: корпорація Microsoft має розгалужену партнерську мережу та велику службу підтримки, які повинні охоплювати всі продукти Microsoft, а не лише Azure. Корпорація Microsoft відстає від деяких інших лідерів хмарних технологій у забезпеченні технічної якості своєї партнерської мережі постачальників хмарних рішень Azure CSP (CSP – Cloud Solution Provider) та співробітників служби уніфікованої підтримки.

Компанія Google, дочірня компанія Alphabet Inc. є лідером у цьому магічному квадранті. Google Cloud Platform (GCP) охоплює майже всі випадки використання IaaS та PaaS. GCP приваблює компанії, які мають на меті замінити застарілу інфраструктуру на контейнерні або безсерверні альтернативи. Організації отримують найкращі результати від GCP, коли вони приймають індивідуальний підхід Google до хмарної архітектури, щоб отримати доступ до інновацій Google у сфері великих даних та штучного інтелекту.

GCP спочатку зростала навколо клієнтів, які створювали хмарні додатки з нуля. Її програма швидкої міграції (Rapid Migration Program, RaMP) приділяє більше уваги, ніж інші програми міграції в хмару, модернізації додатків та гнучким методам. У 2022 році частка ринку GCP зросла більш ніж на 40% і зараз становить понад 10%.

Сильні сторони:

Елегантність дизайну: створюючи широкий спектр сервісів IaaS і PaaS, Google застосовує свої основні принципи дизайну більш послідовно у своєму портфоліо, ніж його конкуренти. Хмарні інженери повідомили Gartner, що вважають API GCP більш послідовними, легшими у вивченні та використанні.

Сервіси IaaS і PaaS з впровадженням штучним інтелектом: Google не тільки підтримує розробку AI/ML та інтеграцію сторонніх моделей за допомогою свого набору інструментів Vertex AI, але й є провідним розробником власних моделей, включаючи популярні великі мовні моделі, такі як PaLM 2. Google розпочав процес інтеграції своїх моделей у свої основні IaaS та PaaS-сервіси на глибокому рівні. Екологічна стійкість: GCP має найвищий рейтинг стійкості серед

провайдерів у цьому магічному квадранті. Енергоефективність центрів обробки даних особливо важлива для клієнтів у регіонах з нестабільним енергопостачанням або цінами на енергоносії, а також для клієнтів, які займаються хмарною діяльністю з високим енергоспоживанням, наприклад, навчанням моделей штучного інтелекту. *Застереження:*

Каденція випуску загальної доступності (GA): Google має практику запуску сервісів у режимі попереднього перегляду та агресивного просування їх на ринку без надання зобов'язань перед GA. Клієнти повинні бути обережними при включенні "найновіших і найкращих" інновацій GCP-сервісів перед їх використанням у проді.

Нерівномірна зрілість галузі та партнерів: Google підтримує повнофункціональну програму міграції клієнтів (RaMP), яка постійно розширюється і налічує понад 70 сертифікованих партнерів з міграції. Проте Gartner досить часто чує від деяких клієнтів, що їхній проект міграції на Google Cloud був ускладнений через відсутність команди підтримки обліку Google або недостатньо глибокого досвіду роботи партнерів з GCP.

Мінімальна інтеграція між Google Cloud та іншими продуктами Alphabet: GCP не має тісної інтеграції з іншими хмарними сервісами Google, зокрема Workspace, а також з іншими продуктами материнської компанії Alphabet. Клієнти, які сподіваються використовувати GCP як основу для створення цифрових рішень з усього портфолію Google, наразі не знайдуть достатньої підтримки, і їм, можливо, доведеться домовлятися про комерційні умови з кількома бізнес-підрозділами Google [7].

1.3.ОГЛЯД ПЛАТФОРМИ KUBERNETES

1.3.1.ПРИЗНАЧЕННЯ KUBERNETES

Kubernetes (скорочено – K8s, де 8 означає кількість літер між "K" та "s") – це система з відкритим вихідним кодом для розгортання, масштабування та управління контейнерними додатками будь-де, тобто її можна розгорнути на власному комп'ютері, локальному дата-центрі або ж у хмарі [8].

Власне самі контейнери, які використовуються в Kubernetes – це прекрасний спосіб для того, щоб упакувати та запустити необхідні для роботи системи, додатки. У виробничих умовах команді інженерів потрібно керувати контейнерами та відслідковувати їх життєвий цикл. Наприклад, якщо один контейнер припиняє свою роботу, інший має бути запущений замість нього. Якщо таких контейнерів в системі багато, то раціонально було б автоматизувати цей процес та дозволити системі робити це. Власне Kubernetes і дозволяє вирішити цю проблему. Kubernetes надає його користувачам каркас для еластичного запуску розподілених систем. Він має здатність вирішувати питання масштабування та аварійного відновлення додатків, що використовуються в системах, має доступні шаблони для розгортання, тощо. Як приклад, Kubernetes також дозволяє реалізувати популярний підхід розгортання за стратегією canary. Розгортання за стратегією canary дозволяє розгорнути нову версію додатку, поступово замінюючи стару. Цей метод дозволяє зменшити ризик впровадження нової версії додатку, забезпечуючи плавний перехід для користувачів.

Kubernetes дозволяє вирішувати наступні задачі з:

Виявлення сервісів та балансування навантаження. Kubernetes може надавати доступ до контейнеру, використовуючи DNS-ім'я або IP-адресу (використання IP-адреси для зв'язку з контейнером не є поширеною практикою, оскільки контейнери є ефемерними за своєю природою і при перестворенні контейнеру йому присвоюється нова адреса). Якщо контейнер зазнає високого навантаження, Kubernetes здатний збалансувати та розподілити його таким чином, щоб якість обслуговування залишалась стабільною.

Оркестрація сховища інформації Kubernetes дозволяє автоматично монтувати системи збереження інформації, залежно від задачі. Це може бути як локальне сховище або ж рішення від хмарних провайдерів.

Автоматичне розгортання та відкатування. Kubernetes надає змогу описати бажаний стан контейнерів, що розгортаються і він регулярно стежить за виконання цієї умови. Наприклад, в Kubernetes можливо автоматизувати процес створення нових контейнерів, видалення існуючих і передачу їх ресурсів на новостворені контейнери.

Оптимізація розподілення ресурсів. Kubernetes має ряд механізмів для ефективного використання ресурсів. За допомогою їх резервування Kubernetes гарантує, що задана кількість ресурсів буде гарантована додатку, навіть якщо інші ресурси будуть потребувати їх. Kubernetes також «розуміє» як оптимально розмістити контейнери між вузлами кластеру, а за нестачі існуючих ресурсів робочих вузлів кластеру має здатність горизонтально або вертикально масштабуватись.

Самозцілення. Kubernetes перезапускає контейнери, що вийшли з ладу, перетворює їх за потреби, перевіряючи задані умови для контейнеру з додатком.

Управління чутливою інформацією. Kubernetes дозволяє зберігати різного роду токени, ключі або іншу інформацію. Їх можна оновлювати без перестворення образів і вони будуть оновлені «на льоту».

1.3.2. СТРУКТУРА KUBERNETES

Кластер Kubernetes складається з набору робочих машин, які називаються вузлами, на яких виконуються контейнеризовані програми. Кожен кластер має принаймні один робочий вузол.

На робочому вузлі (вузлах) розміщуються поди. Всередині одного поду можливо розмістити декілька контейнерів за необхідності. Площина управління керує робочими вузлами та подами кластеру. У проді площина керування зазвичай працює на декількох хостах, а кластер зазвичай складається з більш як одного хоста в цілому, що забезпечує відмовостійкість і високу доступність.

Компоненти площини управління приймають глобальні рішення щодо кластера (наприклад, планування), а також виявляють події кластера та реагують на них (наприклад, запуск нового поду, коли поле *replicas* в маніфесті deployment не відповідає реальності).

Компоненти площини управління можуть бути запущені на будь-якій машині в кластері. Однак, для простоти, інсталяційні скрипти зазвичай запускають усі компоненти площини керування на одній ноді та не запускають контейнери користувача на цьому комп'ютері.

API-сервер – це компонент площини керування Kubernetes, який надає доступ до Kubernetes API. API-сервер є зовнішнім інтерфейсом для площини управління Kubernetes.

Основною реалізацією API-серверу в Kubernetes є kube-apiserver. kube-apiserver призначений для горизонтального масштабування – тобто, він масштабується шляхом розгортання більшої кількості інстансів. Можна запустити декілька екземплярів kube-apiserver і збалансувати трафік між ними.

etcd – консистентне та високодоступне сховище даних у вигляді key-value, що використовується як резервне сховище для всіх даних кластера Kubernetes. Якщо кластер Kubernetes використовує etcd як резервне сховище, потрібно переконатись, у наявності плану резервного копіювання даних.

Компонент площини управління у своєму складі використовує контролери. Логічно, кожен контролер є окремим процесом, але для зменшення складності всі вони компілюються в один бінарний файл і запускаються в одному процесі. Існує багато різних типів контролерів. Ось деякі з них:

- Вузловий контролер відповідає за виявлення та реагування на падіння вузлів.
- Контролер Job відстежує об'єкти Job, які представляють собою одноразові завдання, а потім створює поди для виконання цих Job.
- Контролер EndpointSlice створює об'єкти EndpointSlice для забезпечення зв'язку між сервісами та подами.

- Контролер ServiceAccount створює облікові записи ServiceAccounts за замовчуванням для нових Namespace.

- Компоненти вузлів виконуються на кожному наявному вузлі кластеру, підтримуючи запущені Pods та забезпечуючи середовище виконання Kubernetes.

- Kubelet – це агент, який працює на кожному вузлі кластера. Він стежить за тим, щоб контейнери були запущені у Pods.

- Kubelet отримує набір специфікацій, вказаних у PodSpec, які надаються за допомогою різних механізмів, і гарантує, що контейнери, описані в цих специфікаціях PodSpec, працюють і є здоровими. Kubelet не керує контейнерами, які не були створені Kubernetes.

- kube-proxy – це мережевий проксі-сервер, який працює на кожному вузлі вашого кластера, реалізуючи частину концепції Kubernetes Service. kube-proxy підтримує мережеві правила на вузлах. Ці мережеві правила дозволяють мережевий зв'язок з вузлами з мережевих сесій всередині або поза кластером. kube-proxy використовує рівень фільтрації пакетів на рівні операційної системи, якщо він є і він доступний. В іншому випадку kube-proxy пересилає трафік самотійно.

Середовище виконання контейнерів – фундаментальний компонент, який дозволяє Kubernetes ефективно запускати контейнери. Він відповідає за керування виконанням та життєвим циклом контейнерів у середовищі Kubernetes.

Kubernetes підтримує такі середовища виконання контейнерів, як containerd, CRI-O та будь-які інші реалізації Kubernetes CRI (Container Runtime Interface).

Також даний список є далеко не повним без ряду важливих доповнень, які також доступні в Kubernetes. Доповнення використовують такі об'єкти Kubernetes, як DaemonSet, Deployment і т.д., для їх встановлення в кластері. Оскільки вони надають можливості кластерного рівня, зазвичай вони встановлюються в Namespace kube-system.

Хоча інші доповнення не є обов'язковими, усі кластери Kubernetes повинні мати службу DNS всередині кластеру. Кластерний DNS – це DNS-сервер, додатковий до інших DNS-серверів у вашому середовищі, який обслуговує DNS-записи для служб Kubernetes. Контейнери, запущені Kubernetes, автоматично включають цей DNS-сервер до свого DNS-пошуку.

Dashboard – це веб-інтерфейс загального призначення для кластерів Kubernetes. Він дозволяє користувачам керувати та діагностувати додатки, що працюють у кластері, як і сам кластер.

Мережеві плагіни – це програмні компоненти, які реалізують специфікацію мережевого інтерфейсу контейнера (CNI). Вони відповідають за розподіл IP-адрес між контейнерами і дозволяють їм взаємодіяти один з одним у кластері [9].

Щоб масштабувати додатки та забезпечити їх надійність, потрібно розуміти, як система поводить себе після того, як її було розгорнуто. Можна дослідити системи у кластері Kubernetes, вивчивши її контейнери, Pods, Services та характеристики кластера в цілому. Kubernetes надає детальну інформацію про використання ресурсів додатку на кожному з цих рівнів. Ця інформація дозволяє оцінити продуктивність вашого додатку і визначити, де можна усунути вузькі місця (bottlenecks) для їх усунення та подальшого підвищення загальної продуктивності [10].

У Kubernetes для моніторингу іноді може знадобитись більше однієї системи. Є механізми базового збору метрик та більш просунуті. Так, наприклад, для роботи автоматичного горизонтального або вертикального масштабування, що працюють на основі постійного моніторингу використання ресурсів процесору та оперативної пам'яті достатньо використати проект metrics-server. Для більш складних сценаріїв можна скористатись більш просунутим рішенням, як Prometheus.

Логи додатків можуть допомогти зрозуміти, що відбувається всередині програми. Логи особливо корисні при пошуку потенційних проблем та помилок всередині системи і моніторингу активності кластера. Більшість сучасних

додатків мають певний механізм ведення логів. Середовища виконання контейнерів також розроблені з підтримкою логування контейнеризованих додатків. Найпростішим і найпоширенішим методом ведення логів для контейнерних додатків є запис у стандартний вивід (stdout) і стандартні потоки помилок (stderr).

Однак, стандартного функціоналу, що надається середовищем виконання контейнерів або самим контейнером, зазвичай недостатньо для повноцінного логування, яке було б корисним.

Доступ до логів додатку може знадобитись у випадку наступних подій: container crash, Pod eviction, node death .

У кластері логи повинні мати окреме сховище і життєвий цикл, незалежний від вузлів, контейнерів або Pods. Ця концепція називається логуванням на рівні кластеру.

Архітектура логування на рівні кластеру вимагає окремого бекенду для зберігання та аналізу логів. Kubernetes не надає власного рішення для зберігання даних логів. Натомість існує багато рішень для ведення журналів, які інтегруються з Kubernetes [11].

1.3.3.WINDOWS TA LINUX КЛАСТЕРИ KUBERNETES

Linux та Windows Kubernetes кластери представляють дві різні екосистеми, що використовуються для оркестрації контейнерів, кожна з яких має свої особливості та переваги. Розглянемо їх основні відмінності та поширення.

Linux Kubernetes кластери є найбільш традиційним і широко вживаним варіантом, адже Kubernetes було спочатку розроблено для операційних систем на базі Linux. Linux має велику спільноту, багато дистрибутивів і є відкритим рішенням з великою підтримкою незалежних розробників та великих компаній, типу Google, Microsoft, Red Hat, тощо. Велика гнучкість дозволяє операційних систем на базі Linux дозволяє користувачам легко модифікувати та налаштовувати систему під свої потреби. Він забезпечує велику гнучкість і контроль, широкі можливості для автоматизації та інтеграції з іншими інструментами DevOps. Також Linux підтримує широкий спектр мережевих

плагінів, що робить його ідеальним для комплексних кластерних конфігурацій. В якості підтвердження цього є наявна велика кількість немалої кількості мережевих планів, як плагінів з відкритим кодом, типу kubernetes та Calico, так і пропрієтарних, типу Azure CNI.

Windows Kubernetes кластери, з іншого боку, стали доступними пізніше і орієнтовані на організації, які використовують Windows-орієнтовані системи. Вони дозволяють запускати контейнери Windows в екосистемі Kubernetes, надаючи таким чином можливість інтеграції з існуючими Windows-орієнтованими програмами та сервісами. Це важливо для багатьох корпоративних середовищ, де Windows має сильні позиції. Однак, Windows кластери можуть мати певні обмеження в плані мережевої конфігурації та підтримки додаткового програмного забезпечення порівняно з Linux.

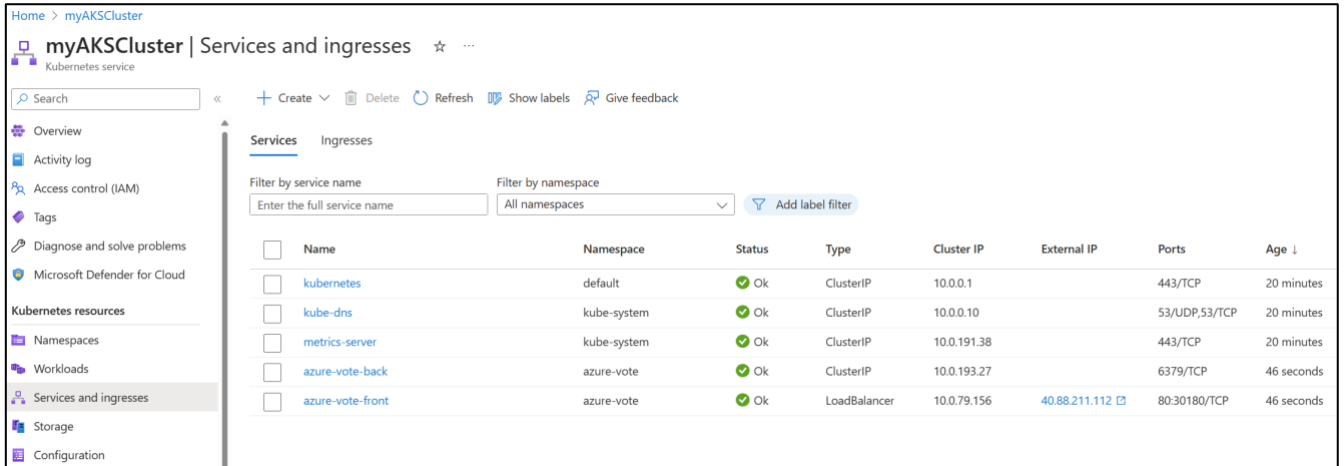
Що стосується популярності, Linux кластери мають більш широке використання в світі Kubernetes, особливо у великих та технологічно орієнтованих компаніях, які цінують його відкритість, гнучкість та масштабованість. Багато інструментів та розробок спільноти спочатку з'являються саме для Linux. Тим не менш, зі зростанням потреби у інтеграції з Windows-специфічними системами, Windows кластери також здобувають популярність, особливо в середовищах, де вже присутня значна інвестиція в технології Microsoft.

У виборі між Linux та Windows для Kubernetes кластерів часто визначальними є специфічні потреби проєкту, вже існуюча інфраструктура та вподобання команди. Linux залишається переважним вибором для більшості нових розгортань Kubernetes через свою зрілість, підтримку спільноти та гнучкість, але Windows кластери продовжують займати свою нішу, особливо в корпоративних середовищах, де потрібна інтеграція з іншими продуктами Microsoft.

1.3.4.БАЗОВІ КОМПОНЕНТИ AZURE KUBERNETES SERVICE

Azure Kubernetes Service (AKS) спрощує розгортання керованого кластера Kubernetes в Azure, беручи частину операційної відповідальності на себе [12].

Розгорнутий в Azure кластер Kubernetes ряд має ряд рішень, які автоматично імплементовано для усіх користувачів, що спрощує управління кластером і також має рішення, які можна також імплементувати з Azure Portal (див. рис.1.3.4.1) або інших рішень, що використовують Azure Resource Manager у своїй роботі (наприклад, рішення для логування та збору метрик).



Name	Namespace	Status	Type	Cluster IP	External IP	Ports	Age
kubernetes	default	Ok	ClusterIP	10.0.0.1		443/TCP	20 minutes
kube-dns	kube-system	Ok	ClusterIP	10.0.0.10		53/UDP,53/TCP	20 minutes
metrics-server	kube-system	Ok	ClusterIP	10.0.191.38		443/TCP	20 minutes
azure-vote-back	azure-vote	Ok	ClusterIP	10.0.193.27		6379/TCP	46 seconds
azure-vote-front	azure-vote	Ok	LoadBalancer	10.0.79.156	40.88.211.112	80-30180/TCP	46 seconds

Рис. 1.3.4.1 Один з пунктів мені налаштування AKS в Azure Portal

Коли створюється новий кластер AKS, автоматично створюється та налаштовується також і площина управління (архітектура кластеру з площиною управління зображена на рис. 1.3.4.2). Площина керування є безкоштовною та обслуговується самим Azure та є абстрагованою від користувача, також вона буде розгорнута в тому ж географічному регіоні, що і весь кластер. Користувачі платять та обслуговують лише вузли, які підключені до кластеру.

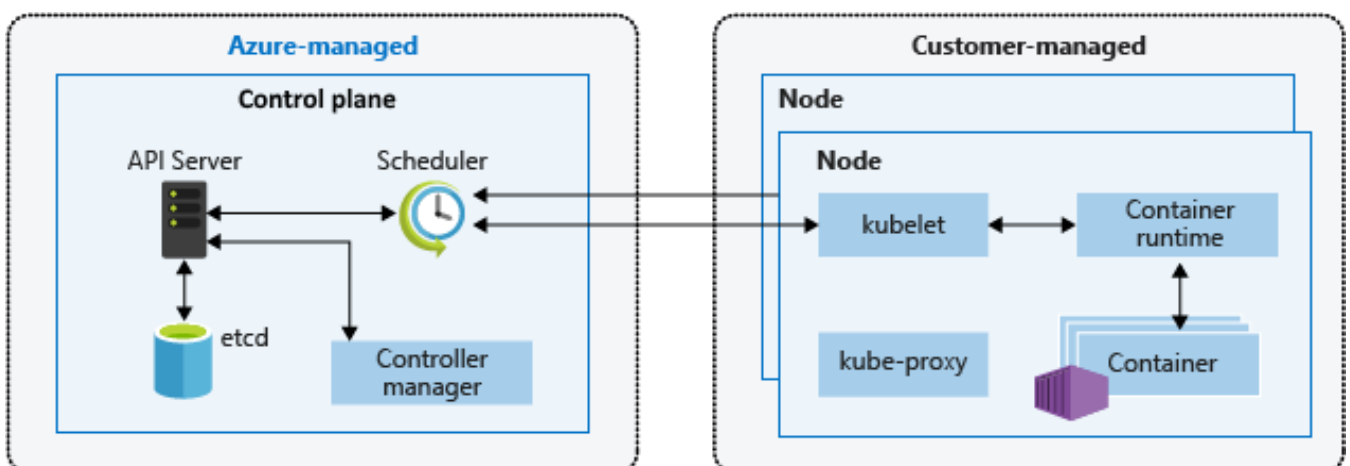


Рис. 1.3.4.2 Узагальнена архітектура кластеру AKS

Площина управління включає наступні основні компоненти Kubernetes: kube-apiserver, etcd, kube-scheduler, kube-controller-manager (див. табл. 1.3.4.1).

Таблиця 1.3.4.1

Складові площини управління

Компонент	Опис
kube-apiserver	API-сервер – це спосіб доступу до базових API Kubernetes. Цей компонент забезпечує взаємодію для інструментів керування, таких як kubectl (див. рис.1.3.4.3) або Kubernetes dashboard (див. рис.1.3.4.4).
etcd	Для збереження стану кластера та конфігурацій Kubernetes використовується високодоступне сховище etcd, що зберігає дані у форматі key-value.
kube-scheduler	Коли додатки створюються або масштабуються, kube-scheduler визначає, які вузли можуть розмістити необхідне робоче навантаження, і запускає їх.
kube-controller-manager	Диспетчер контролерів контролює низку менших контролерів, які виконують такі дії, як реплікація Pods та обробка операцій з вузлами.

```
kube-master:~$ kubectl top nodes
```

NAME	CPU(cores)	CPU%	MEMORY(bytes)	MEMORY%
kube-master	303m	3%	6001Mi	18%
kube-worker1	1613m	20%	11522Mi	35%
kube-worker2	451m	5%	8751Mi	27%
kube-worker4	550m	6%	9780Mi	30%
kube-worker5	237m	2%	10607Mi	33%
kube-gateway	849m	10%	9895Mi	30%
kube-service	171m	4%	3711Mi	23%

Рис. 1.3.4.3 Використання kubectl для отримання інформації про вузли кластеру

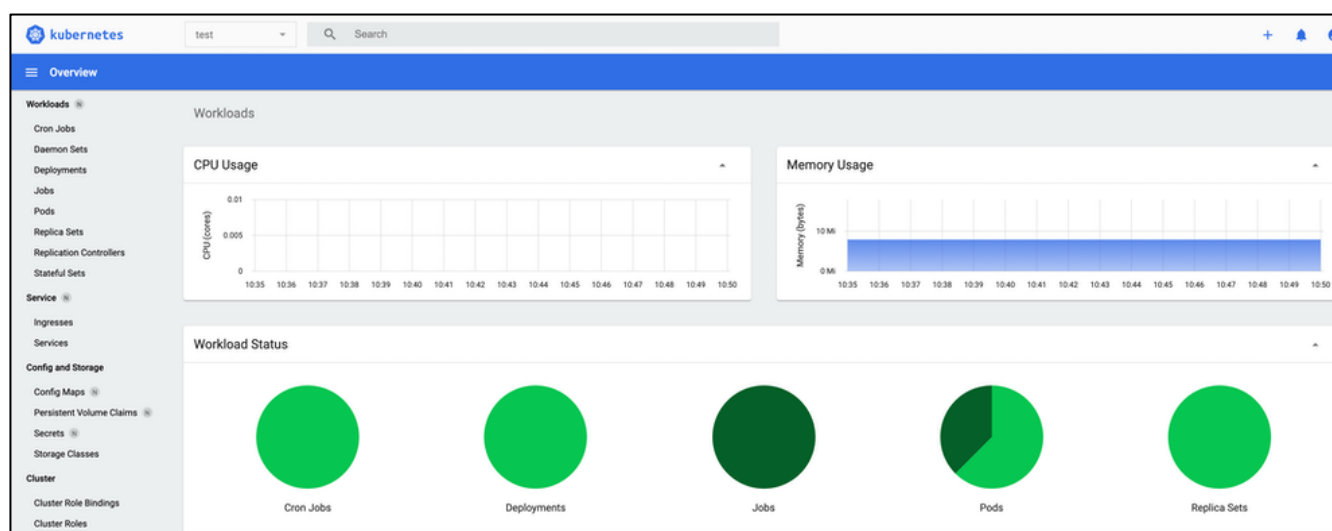
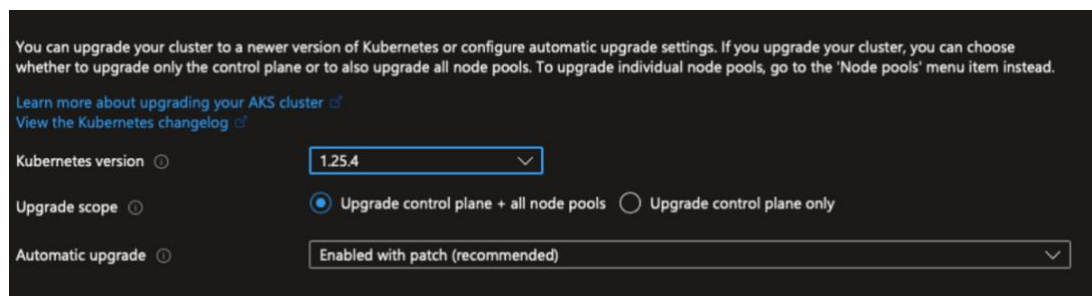


Рис. 1.3.4.4 Kubernetes dashboard з метриками компонентів кластеру

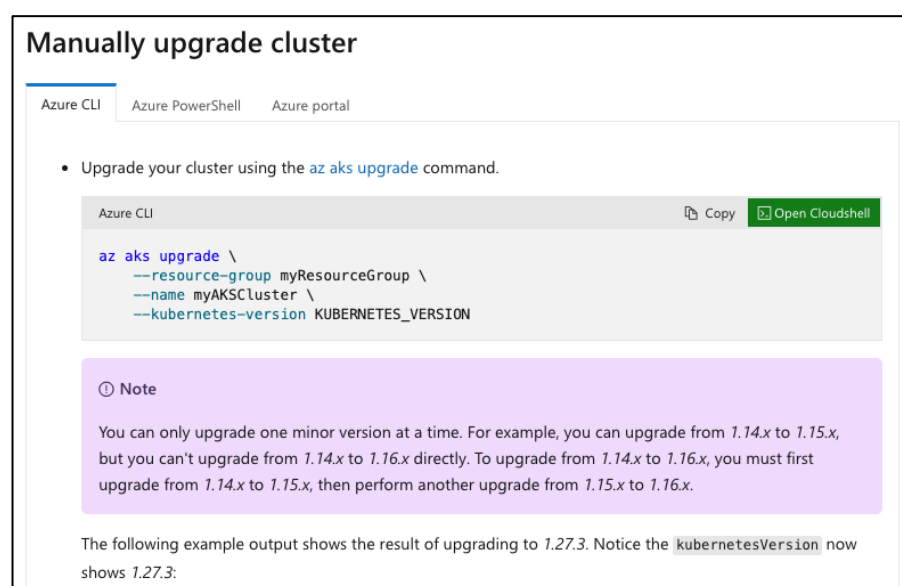
AKS надає в користування площину управління типу single-tenant, з виділеним API-сервером, планувальником тощо. Користувачі самостійно визначають кількість і розмір вузлів, а Azure забезпечує безпечний канал зв'язку між площиною управління та вузлами. Взаємодія з площиною управління відбувається через Kubernetes API, такі як kubectl або через Kubernetes dashboard, як це було згадано в табл.1.3.4.1.

Крім того, що користувачеві кластеру не потрібно самостійно налаштовувати компоненти площини управління (наприклад, високодоступне сховище etcd), отримати доступ напряду до площини управління також не є можливим. Оновлення площини керування Kubernetes та вузлів виконується за допомогою Azure CLI або порталу Azure (див. рис.1.3.4.5-1.3.4.6). Щоб усунути можливі проблеми, можете переглянути журнали площини керування за допомогою Azure Monitor [13].



The screenshot shows the 'Upgrade' section of the Azure portal for an AKS cluster. It includes a text block explaining that users can upgrade to a newer version of Kubernetes or configure automatic upgrade settings. Below this, there are three settings: 'Kubernetes version' set to '1.25.4', 'Upgrade scope' with 'Upgrade control plane + all node pools' selected, and 'Automatic upgrade' set to 'Enabled with patch (recommended)'. There are also links to learn more about upgrading and view the Kubernetes changelog.

Рис. 1.3.4.5 Опція оновлення площини управління та вузлів AKS кластеру з порталу Azure



The screenshot shows the 'Manually upgrade cluster' page in the Azure CLI documentation. It has tabs for 'Azure CLI', 'Azure PowerShell', and 'Azure portal'. The 'Azure CLI' tab is active, showing a list of commands to upgrade the cluster using the 'az aks upgrade' command. A 'Note' box explains that only one minor version can be upgraded at a time, and a direct upgrade from 1.14.x to 1.16.x is not possible. An example output shows the result of upgrading to 1.27.3.

Рис. 1.3.4.5 Опція оновлення площини управління та вузлів AKS кластеру з використанням Azure CLI

Для розгортання власних додатків в кластері Kubernetes має бути як мінімум один вузол. Це правило справедливе і для кластеру AKS, де один вузол тотожний одній віртуальній машині Azure, на якій встановлені необхідні компоненти Kubernetes та середовище виконання контейнерів. Розглянемо детальніше компоненти вузла AKS в табл. 1.3.4.2.

Таблиця 1.3.4.2

Компоненти вузла AKS

Компонент	Опис
kubelet	Агент Kubernetes, який обробляє запити від площини управління та планувальника та запускає відповідні контейнери.
kube-proxy	Керує віртуальною мережею на кожному вузлі. Проксі маршрутизує мережевий трафік і керує IP-адресацією для Services та Pods.
container runtime	Дозволяє запускати контейнеризовані додатки та взаємодіяти з додатковими ресурсами, такими як віртуальна мережа або сховище. Кластери AKS, що використовують Kubernetes версії 1.19+ для пулів вузлів Linux, використовують containerd як середовище виконання контейнерів. Починаючи з версії 1.20 Kubernetes для пулів вузлів Windows, containerd доступний в прев'ю, але Docker все ще залишається середовищем виконання контейнерів за замовчуванням для таких кластерів.

Типова структура вузла кластеру AKS зображена на рис. 1.3.4.7.

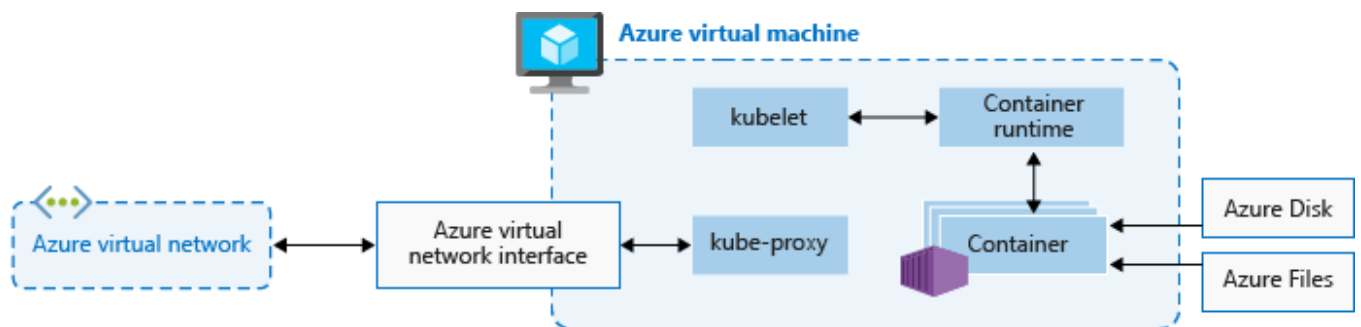


Рис. 1.3.4.7 Типова архітектура вузла AKS

Розмір віртуальної машини Azure для вузлів визначає кількість ядер процесору, оперативної пам'яті, обсяг і доступний тип сховища (наприклад, високопродуктивний SSD або звичайний жорсткий диск). Варто планувати розмір вузла, виходячи з того, чи можуть програми потребувати великих

обчислювальних ресурсів або високопродуктивного сховища. Також AKS підтримує горизонтальне масштабування вузлів відповідно до вимог системи.

В AKS доступні наступні образи для віртуальних машин: Ubuntu Linux, Azure Linux або Windows Server. При створенні кластеру AKS або змінює кількість робочих вузлів в кластері, платформа Azure автоматично створює та налаштовує необхідну кількість віртуальних машин, які вже будуть містити всіх необхідні компоненти для роботи у кластері. Оплата за вузли кластеру стягується як і за використання звичайної VM, тому будь-які знижки на розмір VM (включно з резерваціями) застосовуються автоматично.

Для керованих дисків розмір і продуктивність за замовчуванням буде призначено відповідно до обраного розміру VM і кількості ядер [14].

AKS використовує ресурси вузла кластеру, для розміщення його компонентів, які наведені у табл.1.3.4.2, для того, щоб вузол міг повноцінно функціонувати у складі кластеру. Для деяких користувачів це може бути неочевидним, адже виникає ситуація, коли, в описі розміру віртуальної машини буде вказано, що доступно для використання 4 ядра процесору, та 8 гігабайт оперативної пам'яті, а в той же час для створення нових ресурсів в AKS буде менше, за рахунок цих виділення ресурсів для цих компонентів. Для того, щоб дізнатись кількість ресурсів, яка доступна для виділення, можна виконати наступну команду: `kubectl describe node [NODE_NAME]`, де `[NODE_NAME]` – ім'я вузла в кластері. Потрібно також враховувати, що при встановленні доповнень, в кластер AKS, типу Container Insights, також можуть розгортатись нові об'єкти, що будуть використовувати його ресурси.

В AKS є визначена формула для резервації ресурсів, для компонентів Kubernetes (`kube-reserved`). `kube-reserved` — це параметр, який використовується в Kubernetes для вказівки ресурсів (таких як CPU, пам'ять, і т.д.), які необхідно зарезервувати для системних процесів Kubernetes на кожній машині (вузлі). Це дозволяє забезпечити стабільну роботу самого Kubernetes, адже він відокремлює певну частину ресурсів від тих, які можуть використовуватись контейнерами, які запускаються в кластері.

Власне правила резервація для CPU наведено в табл. 1.3.4.4.

Таблиця 1.3.4.4

Резервації ресурсів процесору для компонентів Kubernetes в AKS

Кількість ядер процесору на хості	1	2	4	8	16	32	64
Зарезервовані кількість ресурсів процесору(millicores)	60	100	140	180	260	420	740

Резервація оперативної пам'яті:

1) Компонент kubelet daemon завжди має `memory.available < 750MB`, що означає, що він завжди має як мінімум 750Mi, що будуть гарантовано йому виділені. Якщо на вузлі не буде достатньо ресурсів, щоб виділити 750MB для kubelet daemon, тоді kubelet завершить один з Pods, для того, щоб виділити ці ресурси.

2) Регресія резервування пам'яті для належного функціонування kubelet daemon (kube-reserved) виглядає наступним чином:

- 25% перших 4GB пам'яті
- 20% наступних 4GB пам'яті (до 8GB)
- 10% наступних 8GB пам'яті (до 16GB)
- 6% наступних 112GB пам'яті (до 128GB)
- 2% якщо використовується більше 128GB

Розглянемо приклад в формулі 1.3.4.1, скільки буде зарезервовано memory для kube-reserved у віртуальній машині з 7GB пам'яті:

$$0.75 + (0.25*4) + (0.2*3) = 0.75 + 1 + 0.6 = 2.35GB$$

(1.3.4.1)

Правила розподілу CPU та memory для kube-reserved незмінні користувачем і в загальному розроблені для наступного цілей:

- Підтримувати працездатність агентів вузлів, включаючи деякі системні Pods, які є критичні для працездатності кластера.
- Змусити вузол повідомляти про менший обсяг CPU та memory, ніж він повідомляв би, якби не був частиною кластера Kubernetes.

Також не варто забувати, що кожен вузол використовує частину ресурсів для потреб встановленої на ньому операційної системи [15].

В архітектурі AKS також є такий важлива сутність, як пул вузлів (node pool) – це набір вузлів, які мають єдину конфігурацію. Кожен кластер AKS має містити мінімум один пул вузлів, мінімум з одним вузлом. Початкова кількість вузлів та їх розмір встановлюється користувачем під час створення кластеру [16].

Також при створенні AKS кластеру важливо розуміти які ресурси створюються в Azure. При створенні кластеру, як і більшість ресурсів в Azure в цілому, обов'язково вказати ім'я ресурсу, групу ресурсів та регіон, в якому кластер буде розміщено. Але є один важливий нюанс, котрий варто враховувати – це власне те, що не всі ресурси кластеру будуть розташовані в одній групі ресурсів. Тобто при створенні кластеру AKS створюється ще одна група ресурсів, яка буде мати ім'я схоже на «MC_myResourceGroup_myAKSCluster_eastus». В цій групі ресурсів будуть знаходитись наступні ресурси:

- Virtual machine scale set (VMSS), кожен VMSS відповідає одному створеному пулу вузлів;
- віртуальна мережа кластеру;
- диск для збереження даних (створюються лише після створення persistent volume) [17].

Також в даних групах ресурсів будуть розміщуватись й інші ресурси, які можуть бути необхідні для роботи кластеру, типу Load Balancer, Public IP і т.д.

ВИСНОВКИ ДО ПЕРШОГО РОЗДІЛУ

Після проведеного аналізу в межах виконання даної наукової практики були зроблені наступні дослідження, що для встановлення даних висновків:

- Проведено порівняння віртуальних машин та контейнерів. Були розглянуті особливості кожної з технологій.
- Проаналізовано які бувають середовища виконання контейнерів, які з них можуть використовуватись в Kubernetes та визначено, що containerd (низькорівневе середовище виконання контейнерів) наразі використовується в Kubernetes за замовчуванням. Не дивлячись на це, Docker є популярним рішенням, що використовується розробника для створення інформаційних систем, а на основі образів, що створені за допомогою Docker можливо запустити контейнери в Kubernetes, не дивлячись на те, що це середовище виконання контейнерів може бути відсутнє, завдяки тому, що Docker Engine використовує containerd, що підтримує стандарти OCI.
- Встановлено етапи життєвого циклу контейнерів на прикладі Docker, що є необхідним кроком для подальшого дослідження можливостей щодо покращення відмовостійкості контейнеризованих систем, розгорнутих в Kubernetes.
- Здійснено аналіз хмарних провайдерів на основі «Magic Quadrant for Strategic Cloud Platform Services» та було вирішено, що через наявні переваги, порівняно з іншими хмарними провайдерами, варто проводити аналіз методів підвищення відмовостійкості на прикладі провайдера хмарних рішень Microsoft Azure.
- Було проаналізовано структуру кластеру Kubernetes, з яких компонентів він складається, та, власне, які компоненти є обов'язковими, а які – ні.
- Встановлено різницю між Linux та Windows кластерами Kubernetes, та встановлено, що перші є більш популярними за рахунок відкритості та підтримки.

- Проаналізовано архітектурні особливості кластеру AKS, встановлено які компоненти Kubernetes встановлюються в кластері AKS, які з них видимі користувачеві та як AKS визначає резервації для системних компонентів.
- Таким чином було здійснений комплексний аналіз технологій, що використовуються в контейнеризованих рішеннях, проаналізовано як вони пов'язані між собою, як вони функціонують, що дозволить продовжити дослідження методів підвищення відмовостійкості даних систем.

2.АНАЛІЗ МЕТОДІВ ПІДВИЩЕННЯ ВІДМОВСТІЙКОСТІ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1.АНАЛІЗ МЕТОДІВ ПІДВИЩЕННЯ ВІДМОВСТІЙКОСТІ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ KUBERNETES

У цьому підрозділі розглянемо методи та концепції підвищення відмовостійкості інформаційної системи, які будуть працювати в будь-якому кластерів Kubernetes та за коректної реалізації дозволять добитись якомога вищого часу безперервної роботи системи.

Kubernetes має ряд особливостей у своїй архітектурі, що дозволяються розробникам розгорнути системи з високим рівнем відмовостійкості. Але при недостатньому рівні експертизи та знань цих особливостей Kubernetes можливо витратити велику кількість ресурсів на розробку ІС, яка в кінцевому результаті буде ненадійною, тобто такою, яка не буде здатна виконувати свою основну функцію або такою, що буде погано масштабуватись чи не масштабуватись взагалі, що призведе до її повільності, що може не задовільнити кінцевого користувача.

Тому один з перших кроків, який потрібно зробити – це правильно розробити архітектуру самої системи, яка буде враховувати особливості Kubernetes. Тобто якщо команда розробки чи будь-яка інша відповідальна команда компанії проаналізувавши вимоги до інформаційної системи вирішили, що варто використати кластер Kubernetes для розміщення їх наявної чи майбутньої ІС, то одним з наступних важливих кроків буде вибір архітектури, що зможе використати основні переваги Kubernetes. Власне такою архітектурою може бути мікросервісна архітектура – це архітектура програмного забезпечення, яка розбиває додаток на невеликі, самостійні компоненти, які називаються мікросервісами. Кожний мікросервіс відповідає за певну функцію додатку. Мікросервіси взаємодіють один з одним за допомогою API.

Якщо перед командою розробки постала задача розгорнути поточну монолітну систему в Kubernetes, то це буде малоефективно і, можливо, досить дорого. Є два основних типи переходу з моноліту на мікросервіси – це

фрагментація та еволюція. Фрагментація – це процес розбиття монолітного додатку на окремі мікросервіси. Цей процес зазвичай виконується шляхом ідентифікації функцій або компонентів монолітного додатку, які можуть бути винесені в окремі мікросервіси. Фрагментація може бути виконана за один раз або поступово. Якщо фрагментація виконується за один раз, це може бути складно і може призвести до переривання роботи додатку. Якщо фрагментація виконується поступово, це може бути менш складно, але може зайняти більше часу. Еволюція в свою чергу – це процес поступового перетворення монолітного додатку в мікросервісний додаток. Цей процес зазвичай виконується шляхом додавання нових мікросервісів до монолітного додатку.

Еволюція може бути менш складною, ніж фрагментація, але може зайняти більше часу. Цей процес також може бути більш безпечним, оскільки новий мікросервіс можна тестувати окремо від монолітного додатку.

При написанні цих сервісів важливо розуміти принципи `stateless` та `stateful` додатків. Це два різні види додатків, які відрізняються тим, як вони зберігаються стан.

`Stateless` додатки не зберігають стан на сервері. Замість цього вони зберігаються стан на клієнті або зовнішній базі даних. Це означає, що кожен запит до `stateless` додатку є незалежним від попередніх.

`Stateful` додатки зберігають стан на сервері. Це означає, що кожен запит до `stateful` додатку може залежати від попередніх запитів.

Звичайно, є й приклади додатків, які не можуть бути `stateless`. Наприклад, електронні магазини повинні зберігати стан корзини покупок клієнта, а системи управління контентом повинні зберігати стан сеансу користувача. У цих випадках доцільно використовувати `stateful` додатки.

Власне використання мікросервісної архітектури з `stateless` логікою дозволяє масштабувати додаток, як горизонтально, так і вертикально, але що найголовніше, так це те, що `stateless` додаток при масштабуванні дозволить кінцевому користувачеві навіть не помітити різниці. В той же час в `stateful` додатках при масштабуванні потрібно враховувати, що при масштабуванні стан

додатку може бути втрачений. Це може призвести до проблем, таких як втрата даних або розрив сеансів користувачів. Також масштабування може бути нерівномірним та у випадку якщо додаток масштабується занадто швидко, це може призвести до того, що деякі його копії будуть перевантажені, а інші будуть недовантажені. Це може призвести до зниження продуктивності додатку.

Resource requests та limits – це два важливі параметри, які використовуються для управління ресурсами, які виділяються контейнерам у Kubernetes. Вони допомагають налаштувати Pods так, щоб додатки отримували достатньо ресурсів для роботи, не перевантажуючи кластер.

Resource requests – це кількість ресурсів, які контейнер потребує для роботи. Цей параметр використовується для того, щоб Kubernetes міг правильно розподілити ресурси між контейнерами.

Resource limits – це максимальна кількість ресурсів, які контейнер може використовувати. Цей параметр використовується для того, щоб захистити кластер від контейнерів, які використовують надто багато ресурсів. Якщо контейнер перевищує свій ліміт, Kubernetes може призупинити контейнер або навіть видалити його з кластера.

Resource requests та limits важливі у налаштуванні відмовостійкості, оскільки вони допомагають забезпечити, щоб контейнери могли продовжувати працювати, навіть якщо один із них вийде з ладу.

Resource requests можна використовувати для того, щоб гарантувати, що у вас завжди буде достатньо контейнерів, які працюють, щоб підтримувати додаток. Наприклад, якщо в системі є додаток, який вимагає трьох контейнерів для роботи, можна встановити resource requests для кожного контейнера таким чином, щоб завжди було достатньо ресурсів для запуску трьох контейнерів.

Resource limits можна використовувати для того, щоб запобігти тому, щоб один із контейнерів вийшов з ладу і вплинув на інші контейнери. Наприклад, якщо є контейнер з базою даних, можна встановити resource limit для цього контейнера таким чином, щоб він не міг використовувати занадто багато пам'яті.

Це допоможе запобігти тому, щоб контейнер з базою даних вичерпав пам'ять і вийшов з ладу, що може призвести до відмови додатка.

Resource requests та limits також можуть допомогти оптимізувати використання ресурсів у кластері Kubernetes. Наприклад, можна використовувати resource requests для того, щоб розподілити ресурси між контейнерами таким чином, щоб вони були максимально ефективними.

Масштабувати додатки в Kubernetes можна автоматично. HPA (Horizontal Pod Autoscaler) – це компонент Kubernetes, який автоматично масштабує додатки на основі метрик таких, як кількість запитів на секунду, використання процесорного часу або використання пам'яті. Коли HPA визначає, що додаток потрібно масштабувати, він збільшує або зменшує кількість копій додатку. Ця кількість копій визначається правилом масштабування HPA. Правило масштабування HPA зазвичай базується на певній метриці (або метриках), що вказані безпосередньо в правилі.

Наприклад, можна налаштувати HPA таким чином, щоб він збільшував кількість копій додатку, коли кількість запитів на секунду перевищує певне значення.

Для вертикального автоматичного масштабування в Kubernetes використовується VPA (Vertical Pod Autoscaler) – це компонент Kubernetes, який автоматично масштабує додатки на основі метрик ресурсів. VPA використовує метрики, такі як використання процесорного часу або використання пам'яті, для визначення того, коли потрібно збільшити або зменшити ресурси, виділені для кожного Pod. Можливо налаштувати VPA таким чином, щоб він збільшував кількість процесорного часу, виділеного для кожного Pod, коли його використання буде перевищувати певне значення. Важливо зазначити, що VPA при збільшені чи зменшені ресурсів для Pods перестворює його, що може бути критично для stateful додатків. VPA також вміє працювати в режимі рекомендацій, коли він аналізує поточне використання ресурсів, а потім відповідно надає рекомендації.

Наступний тип масштабування є одним з підвидів HPA, і називається він KEDA (Kubernetes Event-driven Autoscaling) – це система оркестрації контейнерів, яка дозволяє масштабувати додатки на основі зовнішніх подій. KEDA використовує метрики, такі як кількість подій, кількість повідомлень або кількість звернень, для визначення того, коли потрібно масштабувати додаток.

KEDA можна використовувати для масштабування додатків, які не підходять для масштабування за допомогою HPA або VPA. Наприклад, KEDA можна використовувати для масштабування додатків, які обробляють зовнішні події, такі як події з Kafka або RabbitMQ. На офіційному сайті KEDA є опис правил за якими KEDA може бути активовано (див. рис.2.1.1)

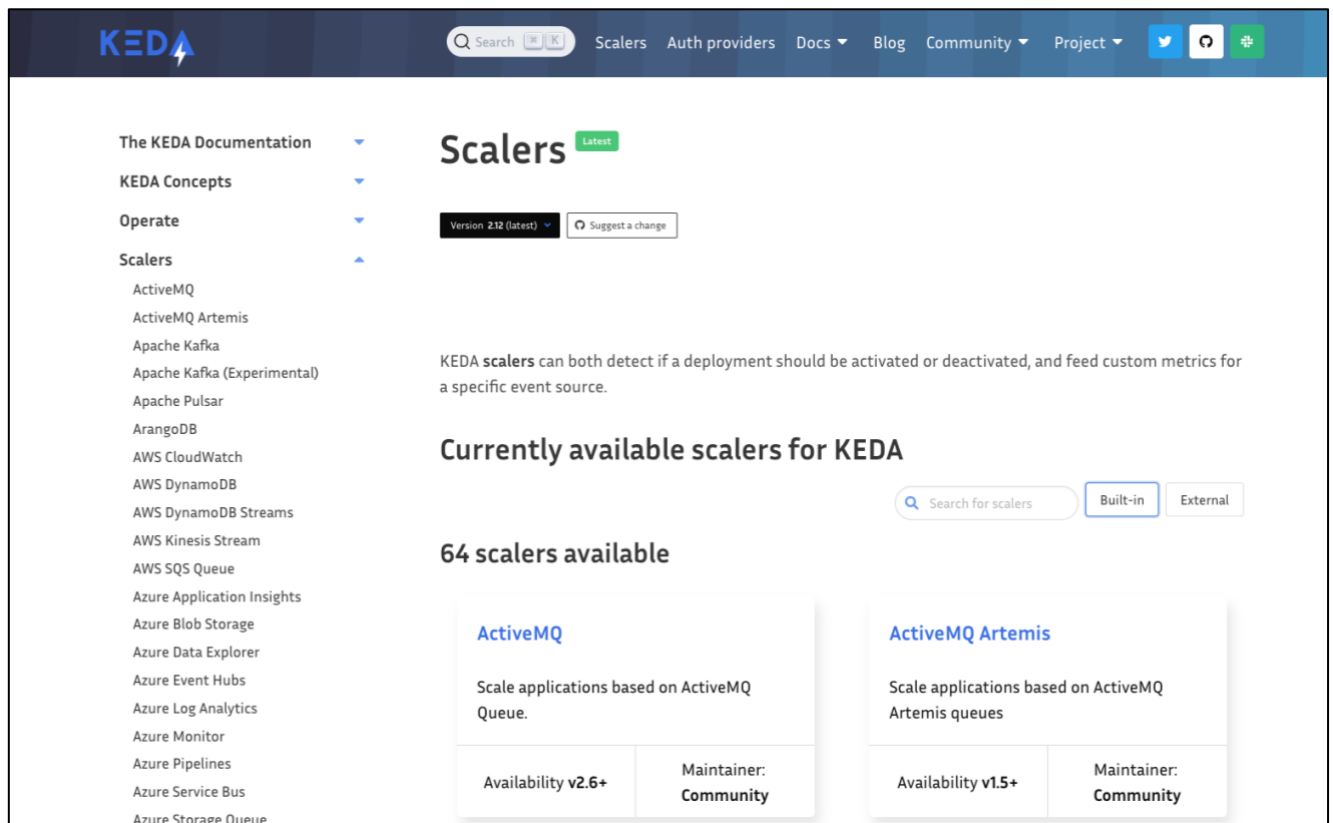


Рис. 2.1.1 Офіційний сайт KEDA, на якому доступний список правил для масштабування

Kubernetes також має вбудовані механізми самовідновлення. Якщо контейнер завершив свою роботу з ненульовим кодом, то kubelet може перезапустити контейнер, якщо це задано в його restartPolicy.

Також в Kubernetes передбачені три типи перевірки стану Pod.

kubelet використовує liveness probes, щоб знати, коли перезапустити контейнер. Наприклад, liveness probes можуть виявити deadlock – ситуація, коли контейнер запущено, він працює, але не виконує одну з основних або основну функцію. Перезапуск контейнера в такому стані може допомогти зробити програму більш стабільною та тимчасово вирішити цю проблему, незважаючи на баг, який викликав цей deadlock.

Загальною схемою для liveness probe є використання того ж самого кінцевої точки (endpoint) додатку, що і для readiness probe (наприклад, /healthz), але з вищим порогом відмови (failureThreshold). Це гарантує, що протягом певного періоду часу Pod буде вважатися не готовим, перш ніж він буде повністю знищений.

kubelet використовує readiness probes, щоб дізнатися, коли контейнер готовий приймати трафік. Pod вважається готовим, коли всі його контейнери готові. Один з прикладів використання readiness probe це перевірка Pod, що використовуються в якості бекенду для Services. Якщо Pod не готовий, він видаляється з load balancer і трафік на нього не поступає.

kubelet використовує startup probe, щоб дізнатися, коли запусився додаток всередині контейнеру запусився. Якщо такий startup probe налаштовано, то liveness та readiness probes не запусяться, доки startup probe не буде успішним, щоб переконатися, що ці probes не заважають запуску додатку. Такий сценарій може бути використано для перевірки життєздатності контейнерів, які повільно запускаються, щоб уникнути їхнього знищення kubelet'ом до того, як вони навіть запусяться [18].

Cluster Autoscaler автоматично додає або видаляє вузли в кластері на основі запитів на ресурси, що надходять від Pods. Він перевіряє Pods, що не були розміщені на існуючих вузлах кожні 10 секунд (частоту сканування можна налаштувати за допомогою параметру --scan-interval). Pod буде в стані незапланованим (unschedulable), коли планувальник Kubernetes не може знайти вузол, який може вмістити його. Наприклад, Pod може мати в requests таке значення для CPU, що жоден наявний вузол кластеру буде не в змозі

задовільнити цю вимогу. Unschedulable Pods розпізнаються за параметром PodCondition. Щоразу, коли планувальник Kubernetes не може знайти місце для запуску Pod, він встановлює для "schedulable" PodCondition в значення false, а причину – "unschedulable". Якщо СА бачить unschedulable Pods, то він спробує знайти нове місце для їх запуску [19]. Передбачається, що кластер Kubernetes розгорнуто на певній групі вузлів, а всередині групи вузлів всі машини мають однакову кількість ресурсів (CPU, memory) та мають однакові мітки. Таким чином, збільшення розміру групи вузлів призведе до створення нових машин, які будуть подібні до тих, що вже є у кластері – просто на них не буде запущено жодного створеного користувачем Pod'у (будуть створені тільки обов'язкові компоненти для вузлів системні ресурси та DaemonSets) [20].

Аналогічно з інтервалом в 10 секунд за замовчуванням СА і перевіряє на вузли-кандидати на видалення. Вузол може бути видалено за наступних умов:

- сума CPU та memory requests усіх Pods, запущених на цьому вузлі (DaemonSet Pods і Mirror pods включено за замовчуванням, але це можна налаштувати за допомогою параметрів --ignore-daemonsets-utilization та --ignore-mirror-pods-utilization), є меншою за 50% від виділеної вузлові ємності. (До версії 1.1.0 використовувалось значення всіх доступних ресурсів на вузлі, а не allocatable). Попіг у 50% можна змінити за допомогою параметру --scale-down-utilization-threshold;
- усі Pods, запущені на вузлі (окрім тих, які запущено на всіх вузлах за замовчуванням, наприклад, manifest-run Pods або Pods запущені за допомогою DaemonSet), можуть бути переміщені на інші вузли. Під час перевірки цієї умови запам'ятовуються нові місця розташування всіх рухомих подів. Таким чином, Cluster Autoscaler знає, куди можна перемістити кожний з них, і які вузли залежать від яких інших вузлів з точки зору міграції. Звісно, може статися так, що з часом планувальник розмістить контейнери десь в іншому місці.

Якщо вузол не використовується більше 10 хвилин, його буде видалено (цей час можна змінити за допомогою спеціальних параметрів). Cluster Autoscaler видаляє один порожній вузол за раз, щоб зменшити ризик створення нових

unschedulable Pods. Наступний вузол може бути завершений одразу після першого, якщо він також був непотрібний більше 10 хвилин. Непотрібні вузли, можна видаляти до 10 вузлів за раз (це число можна налаштувати за допомогою параметру `--max-empty-bulk-delete`).

Що відбувається, коли завершується не порожній вузол? Як було сказано вище, всі Pods повинні бути переміщені на інші вузли. Cluster Autoscaler робить це, виселяючи їх (цей процес називається *eviction*) і вузол переходить у статус *tainted*, що означає, що він помічений як такий, що більше не придатний для розміщення Pods на ньому [21].

Kubernetes пропонує можливості для запуску високодоступних додатків, навіть коли користувачі кластеру мають добровільні перерви.

Власник додатку може створити для кожного застосунку PodDisruptionBudget (PDB). PDB обмежує кількість одночасно недоступних Pods реплікованого застосунку через добровільні перерви. Наприклад, додаток, що базується на голосуванні, має мати певну кількість, яка ніколи не буде нижчою за задану, а фронтенд-додаток має забезпечити, що відсоток Pods ніколи не нижче заданого відсотку від загального числа.

Адміністратори кластерів та хостинг-провайдери повинні використовувати інструменти, які поважають PodDisruptionBudget, викликаючи *eviction* API замість прямого видалення Pods або Deployments.

Наприклад, підкоманда `kubectl drain` дозволяє вам позначити вузол як такий, що виходить з ладу. При запуску цієї команди (`kubectl drain`), інструмент намагається перевести всі Pods на вузлі у стан *evicted* і далі вони мають бути розгорнуті на інших доступних вузлах. *Eviction*-запит, який `kubectl` подає, може бути тимчасово відхилений, тому інструмент періодично повторює всі невдалі запити до тих пір, поки всі Pods на цільовому вузлі не будуть завершені або поки операція не завершиться з помилкою за перевищення часу виконання.

PDB вказує мінімальну кількість реплік, яку додаток може мати, але це не означає, що саме цю кількість він зобов'язаний мати. Наприклад, Deployment, має наступну конфігурацію: `.spec.replicas: 5`, тобто в нормальному стані має бути

розгорнути 5 Pods. Якщо в налаштуваннях PDB мінімальна дозволена кількість Pods: 4, тоді eviction API дозволить добровільне переривання одного (але не двох чи більше) Pods за раз.

Група подів, що складає застосунок, визначається за допомогою селектора міток, такого самого, як той, що використовує контролер застосунку (розгортання, stateful-set тощо).

"Призначена" кількість подів розраховується з `.spec.replicas` ресурсу робочого навантаження, який управляє цими подами. Контрольна панель виявляє власний ресурс робочого навантаження, досліджуючи `.metadata.ownerReferences` пода.

Недобровільних перерв (типу збій апаратного забезпечення кластеру, видалення вузла, паніка ядра, від'днання вузла від кластеру) не можна запобігти за допомогою PDB; однак вони рахуються до `disruption budget`.

Pods, які видаляються або стають недоступними через поступове оновлення застосунку, враховуються у `disruption budget`, але ресурси робочого навантаження (такі як Deployment і StatefulSet) не обмежені PDB при здійсненні оновлень. Натомість обробка помилок під час оновлення застосунків налаштовується у специфікації конкретного ресурсу робочого навантаження.

Рекомендується встановити політику `AlwaysAllow` виселення нездорових Pod-ів для вашого `PodDisruptionBudget`, щоб підтримувати eviction неправильно функціонуючих додатків під час здійснення безпечного видалення вузла (`draining`). За замовчуванням поведінка полягає в очікуванні, поки Pods додатку не стануть здоровими, перш ніж можна буде продовжити безпечне видалення вузла.

Коли Pods видаляється за допомогою eviction API, він завершується поступово («правильно»), враховуючи налаштування `terminationGracePeriod` [22].

2.2. АНАЛІЗ МЕТОДІВ ПІДВИЩЕННЯ ВІДМОВСТІЙКОСТІ ІНФОРМАЦІЙНОЇ СИСТЕМИ LINUX КЛАСТЕРУ AKS

Окрім застосування різних методів та технологій для підвищення відмовостійкості на рівні кластеру Kubernetes, важливо також враховувати конфігурацію кластеру в хмарного провайдера та приділяти пильну увагу конфігураціям, що можуть вплинути на стабільність системи. Більшість ресурсів Azure мають чітко описаний Service Level Agreements (SLA), в якому описуються зобов'язання Microsoft щодо безвідмовної роботи та підключення до онлайн-служб Microsoft. В Standard AKS pricing tier SLA становить 99.9% для Kubernetes API server, який є критичним елементом кластеру; дана опція рекомендована для прод навантажень, а Free AKS pricing tier в свою чергу не містить SLA і рекомендований для тестових середовищ. Для того, щоб мати SLA в 99.95% необхідно увімкнути Availability zones для пулів вузлів кластеру. Availability zones – це рішення, що пропонує покращити доступність та надійність інфраструктури, що захищає програми та дані від збоїв у роботі центру обробки даних. Ці зони – це унікальні фізичні місця в регіоні Azure. Кожна зона включає один або кілька центрів обробки даних, обладнаних незалежним живленням, охолодженням і мережею. Щоб забезпечити відмовостійкість, у всіх регіонах із підтримкою зон завжди є більше однієї зони. Фізичне розділення зон доступності в регіоні захищає програми та дані від збоїв у роботі центрів обробки даних. Кластери AKS, розгорнуті з використанням Availability zones, можуть розподіляти вузли між кількома зонами в межах одного регіону. Наприклад, кластер у регіоні East US 2 може створювати вузли у всіх трьох зонах доступності в East US 2 (див. рис. 2.2.1). Такий розподіл ресурсів кластера AKS підвищує доступність кластера, оскільки вони стійкі до збоїв у певній зоні.

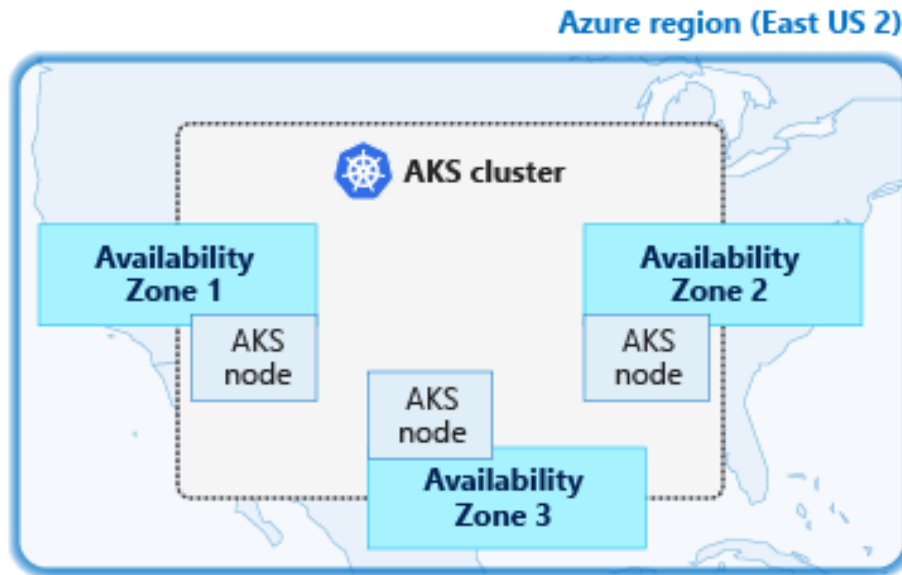


Рис. 2.2.1 Availability zones в регіоні East US 2, та розподілені по зонах пули вузлів

Якщо в кластері використовується `PersistentVolumeClaim`, що при створенні в AKS в Azure створює диск (наприклад, `StandardSSD_ZRS`) та при створенні цього PVC, можливо вказав, що цей диск має використовувати Availability zone і він буде змонтований до Pod в коректній зоні. Також є можливим розгортання об'єкту `PersistentVolumeClaim`, що посилається на `Azure Managed Disk`, у багатозонному кластері AKS, і Kubernetes подбає про розміщення будь якого Pod, що претендує на цей PVC, у правильній Availability zone [23].

Як було згадано раніше, Kubernetes має велику кількість плагінів, як відкритих, так і пропрієтарних. В Azure є можливість вибирати між ними, але Microsoft надають підтримку тільки для пропрієтарних плагінів і у випадку вибору відкритих рішень не варто розраховувати на підтримку. Лише спільнота з розробників даних плагінів та інших користувачів зможуть допомогти організаціям в такому разі, але вони не надають жодних гарантій чи гарантованих термінів відповіді. Тому якщо в проді трапиться нестандартна ситуація, що буде критичною, то краще розраховувати на рішення з підтримкою фахівців, тобто на пропрієтарні.

Розглянемо перший тип плагінів, а саме мережеві плагіни в AKS. Мережевий плагін в AKS відіграє важливу роль у забезпеченні комунікації між контейнерами та між контейнерами та зовнішнім світом. Основна задача цього

плагіну полягає в управлінні IP-адресами для подів, сервісів та інших ресурсів в Kubernetes. Він визначає, як мережевий трафік буде направлятися всередині кластера та як кластер спілкується з зовнішньою мережею. таблицю 2.2.1 з доступними в AKS мережевими плагінами.

Таблиця 2.2.1

Порівняльна таблиця Networking plugins для AKS

Функціонал	Kubenet	Azure CNI	Azure CNI Overlay	Azure CNI Powered by Cilium
Розгортання кластера в існуючій або новій віртуальній мережі	Підтримується - UDRs застосовуються вручну	Підтримується	Підтримується	Підтримується
З'єднання Pod-pod	Підтримується	Підтримується	Підтримується	Підтримується
З'єднання Pod-VM; VM в тому ж Vnet	Працює, коли ініціалізовано від Pod	Підтримується в обидва шляхи	Працює, коли ініціалізовано від Pod	Працює, коли ініціалізовано від Pod
З'єднання Pod-VM; VM в peered Vnet	Працює, коли ініціалізовано від Pod	Підтримується в обидва шляхи	Працює, коли ініціалізовано від Pod	Працює, коли ініціалізовано від Pod
Доступ до on-premise через VPN чи Express Route	Працює, коли ініціалізовано від Pod	Підтримується в обидва шляхи	Працює, коли ініціалізовано від Pod	Працює, коли ініціалізовано від Pod
Надається доступ до K8s, через Load balancer, App Gateway чи ingress controller	Підтримується	Підтримується	Нема підтримки Application Gateway Ingress Controller (AGIC)	Ті ж обмеження, що і в Overlay
Підтримка Windows пулів вузлів	Не підтримується	Підтримується	Підтримується	Доступний тільки для Linux, не для Windows

Власне Azure CNI є пропрієтарним рішенням з підтримкою від Microsoft і відповідно до таблиці має найкращу підтримку для багатьох сценаріїв.

Порівняємо тепер Azure Network Policy Manager та Calico Network Policy (див. табл.2.2.2).

Azure Network Policy Manager має підтримку від Microsoft, підтримує Azure CNI та справляється зі своєю головною функцією: створення правил для вхідного та вихідного трафіку. За замовчуванням в Kubernetes Pods весь вхідний і вихідний трафік дозволений і Pods можуть між собою встановлювати мережевий зв'язок.

Таблиця 2.2.2

Порівняльна таблиця Network Policy

Функціонал	Azure Network Policy Manager	Calico Network Policy
Підтримувані платформи	Linux, Windows Server 2022	Linux, Windows Server 2019 and 2022
Підтримувані мережеві параметри	Azure CNI	Azure CNI (Linux, Windows Server 2019 and 2022) and kubenet (Linux)
Відповідність специфікації Kubernetes	Підтримуються всі типи політик	Підтримуються всі типи політик
Додаткові функції	Відсутні	Розширена модель політики, що складається з глобальної мережевої політики, глобального мережевого набору і кінцевого вузла.
Підтримка	Підтримується командою підтримки та інженерів Azure	Підтримка спільнотою Calico. Наявна додаткова платна підписка
Логування	Логи доступні за допомогою команди <code>kubectl log -n kube-system <network-policy-pod</code>	Інформація доступна на офіційному сайті

Розглянемо наступний важливий компонент Kubernetes кластеру, як ingress controller. Ingress Controller в Kubernetes – це компонент для управління зовнішнім доступом до сервісів у кластері через HTTP та HTTPS маршрутизацію. Він функціонує як вхідна точка, що направляє трафік із зовнішнього світу до відповідних сервісів в кластері здатен до маршрутизації трафіку, забезпечення безпеки через SSL/TLS та ефективного управління зовнішніми IP-адресами. Вибір і налаштування Ingress Controller залежать від конкретних потреб. На сьогодні вибір Ingress controller'ів досить великий (див. рис. 2.2.2), але

сконцентруємо увагу на пропрієтарному рішенні від Microsoft – Application Gateway Ingress Controller (AGIC).

AGIC допомагає усунути необхідність мати ще один Load balancer/Public IP address перед кластером AKS і уникає багатократних переходів у вашому каналі передачі даних, перш ніж запити досягнуть кластера AKS. Application Gateway зв'язується з Pods, використовуючи їхні приватні IP-адреси напряму, і не потребує NodePort або KubeProxy. Ця можливість також забезпечує кращу продуктивність розгорнутих додатків.. Ingress Controller підтримує Web Application Firewall наступних SKU Standard_v2 і WAF_v2, які також забезпечують переваги автомасштабування та безпеки. Application Gateway може реагувати на збільшення або зменшення навантаження трафіку і відповідно масштабуватися, не споживаючи жодних ресурсів вашого кластера AKS

Ingress Controllers

In order for the Ingress resource to work, the cluster must have an ingress controller running.

Unlike other types of controllers which run as part of the `kube-controller-manager` binary, Ingress controllers are not started automatically with a cluster. Use this page to choose the ingress controller implementation that best fits your cluster.

Kubernetes as a project supports and maintains [AWS](#), [GCE](#), and [nginx](#) ingress controllers.

Additional controllers

Note: This section links to third party projects that provide functionality required by Kubernetes. The Kubernetes project authors aren't responsible for these projects, which are listed alphabetically. To add a project to this list, read the [content guide](#) before submitting a change. [More information](#).

- [AKS Application Gateway Ingress Controller](#) is an ingress controller that configures the [Azure Application Gateway](#).
- [Alibaba Cloud MSE Ingress](#) is an ingress controller that configures the [Alibaba Cloud Native Gateway](#), which is also the commercial version of [Higress](#).
- [Apache APISIX ingress controller](#) is an [Apache APISIX](#)-based ingress controller.
- [Avi Kubernetes Operator](#) provides L4-L7 load-balancing using [VMware NSX Advanced Load Balancer](#).
- [BFE Ingress Controller](#) is a [BFE](#)-based ingress controller.
- [Cilium Ingress Controller](#) is an ingress controller powered by [Cilium](#).
- [The Citrix ingress controller](#) works with [Citrix Application Delivery Controller](#).
- [Contour](#) is an [Envoy](#) based ingress controller.
- [Emissary-Ingress API Gateway](#) is an [Envoy](#)-based ingress controller.
- [EnRoute](#) is an [Envoy](#) based API gateway that can run as an ingress controller.
- [Easgress IngressController](#) is an [Easgress](#) based API gateway that can run as an ingress controller.
- [F5 BIG-IP Container Ingress Services for Kubernetes](#) lets you use an Ingress to configure [F5 BIG-IP](#) virtual servers.
- [FortiADC Ingress Controller](#) support the Kubernetes Ingress resources and allows you to manage [FortiADC](#) objects from Kubernetes
- [Gloo](#) is an open-source ingress controller based on [Envoy](#), which offers API gateway functionality.
- [HAProxy Ingress](#) is an ingress controller for [HAProxy](#).
- [Higress](#) is an [Envoy](#) based API gateway that can run as an ingress controller.
- [The HAProxy Ingress Controller for Kubernetes](#) is also an ingress controller for [HAProxy](#).
- [Istio Ingress](#) is an [Istio](#) based ingress controller.
- [The Kong Ingress Controller for Kubernetes](#) is an ingress controller driving [Kong Gateway](#).
- [Kusk Gateway](#) is an [OpenAPI](#)-driven ingress controller based on [Envoy](#).
- [The NGINX Ingress Controller for Kubernetes](#) works with the [NGINX](#) webserver (as a proxy).
- [The ngrok Kubernetes Ingress Controller](#) is an open source controller for adding secure public access to your K8s services using the [ngrok](#) platform.
- [The OCI Native Ingress Controller](#) is an Ingress controller for [Oracle Cloud Infrastructure](#) which allows you to manage the [OCI Load Balancer](#).
- [The Pomerium Ingress Controller](#) is based on [Pomerium](#), which offers context-aware access policy.

Рис. 2.2.2 Список Ingress controller'ів на офіційному сайті Kubernetes

AGIC налаштовується за допомогою ресурсу Kubernetes Ingress, а також Service і Deployments/Pods. Він надає багато функцій, використовуючи нативний Application Gateway L7 load balancer від Azure. Ось деякі з них:

- Маршрутизація URL-адрес;
- Cookie-based affinity;
- TLS termination;
- End-to-end TLS;
- Підтримка публічних, приватних і гібридних веб-сайтів;
- Інтегрований Web Application Firewall [24].

AGIC можливо встановити в AKS кластер, в якості доповнення. Це доповнення повністю підтримується та оновлює через Azure, тому даний тип встановлення є досить легким. Розміщується в кластерві він все одно, як Pod. Не дивлячись на це, всі користувачі AGIC завжди будуть отримувати останню версію доповнення.

Щоб уникнути проблем із надійністю, пов'язаних з обмеженнями Azure Load Balancer при інтенсивному використанні вихідного трафіку для підключення інформаційної системи до зовнішніх сервісів можна використовувати NAT Gateway.

Azure NAT Gateway – це повністю керована та високостійка служба трансляції мережевих адрес (NAT). Azure NAT Gateway можна використовувати в якості шлюзу за замовчуванням для об'єктів приватної підмережі для виходу в Інтернет (в чому числі і AKS), залишаючись при цьому повністю приватними. Через NAT Gateway можуть проходити лише пакети, що надходять у відповідь на вихідне з'єднання. NAT Gateway є рекомендованим методом для систем з великою кількістю вихідного трафіку, де 65534 портів може бути недостатньо для вихідного трафіку.

ВИСНОВКИ ДО ДРУГОГО РОЗДІЛУ

В ході проведених досліджень було складено фреймворк з підвищення відмовостійкості для інформаційної системи, що буде розміщено в Linux кластерів Azure Kubernetes Service.

Фреймворк можна поділити на дві частини:

Загальні принципи та підходи для підвищення відмовостійкості інформаційних систем на базі Kubernetes (незалежно від того, де вони розміщені)

Технології та методи покращення стійкості хмарної інфраструктури з використання Azure Kubernetes service.

В першій частині фреймворку було визначено, що для підвищення відмовостійкості в кластері Kubernetes для інформаційних систем варто використовувати мікросервісну архітектуру, з stateless компонентами. Також варто використовувати один з або одночасно декілька типів масштабування (HPA, VPA, KEDA), readiness, startup та liveness probes, Cluster autoscaler, PDB, а також використовувати resource requests та limits.

В другій частині фреймворку було визначено які архітектурні рішення допоможуть підняти SLA системи, а саме використання Standard AKS pricing tier, для пулів вузлів необхідно використовувати Availability zones, а для більшості компонентів кластеру краще віддати перевагу пропрієтарним рішенням (а саме Azure CNI, Azure Network Policy Manager, Azure Application Gateway Ingress Controller, Azure NAT Gateway), оскільки вони мають офіційну підтримку від Microsoft, що може бути критичним у прод середовищах, підтримують багато сценаріїв реалізації та є зручними в імплементації та майже не вимагають або зовсім не вимагають подальшої підтримки зі сторони користувачів кластеру.

3.ДОСЛІДЖЕННЯ ЗАПРОПОНОВАНИХ ПРАКТИК

3.1.ДОСЛІДЖЕННЯ ПРАКТИК ПОКРАЩЕННЯ ВІДМОВОСТІЙКОСТІ В KUBERNETES

Розглянемо як застосовуються CPU та memory requests та limits (див. рис 3.1.1).

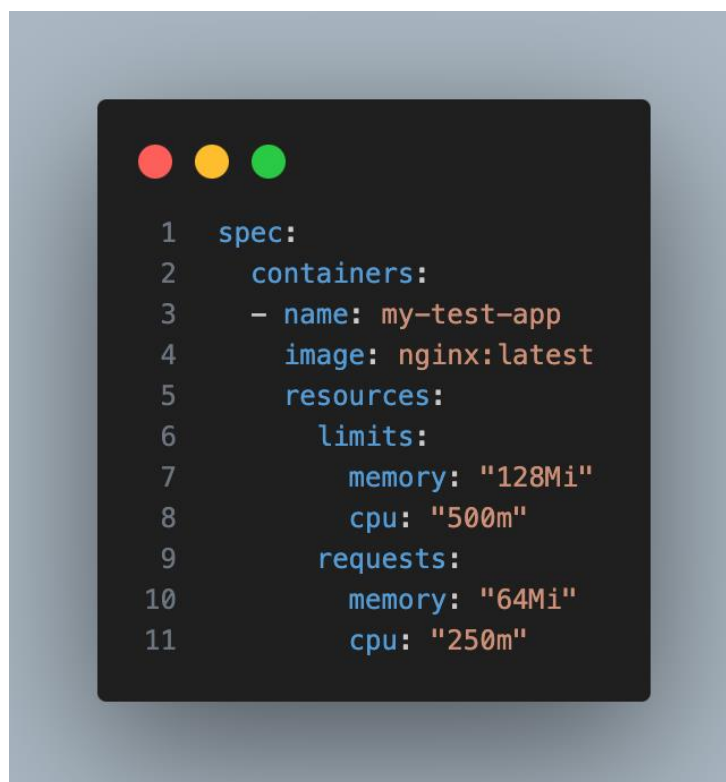


Рис.3.1.1 Застосування requests та limits до контейнеру

Без застосування їх в кластері може бути проблема, коли один додаток, який не має встановлених limits починає використовувати більшість доступних ресурсів, через що може страждати швидкодія інших. Також важливо правильно виставляти значення для requests та limits, для того, щоб планувальник робив раціональні рішення щодо розміщення Pods. Тобто можна побачити як досить просте рішення може значно вплинути на ситуацію розподілення ресурсів в кластері. Вказавши requests та limits ми запобігли так проблемі resource exhaustion.

Далі звернемо увагу на те, як можна налаштувати НРА для нашого Deployment (див рис.3.1.2). Розглянемо як воно працює в даному випадку. При значенні навантаження CPU < 50% буде розгорнуто одну репліку додатку, але якщо навантаження на CPU буде зростати, то спочатку буде розгорнуто ще одну

репліку, а згодом і ще одну. І навпаки, коли навантаження буде спадати, то поступово планувальником буде встановлено один Pod.

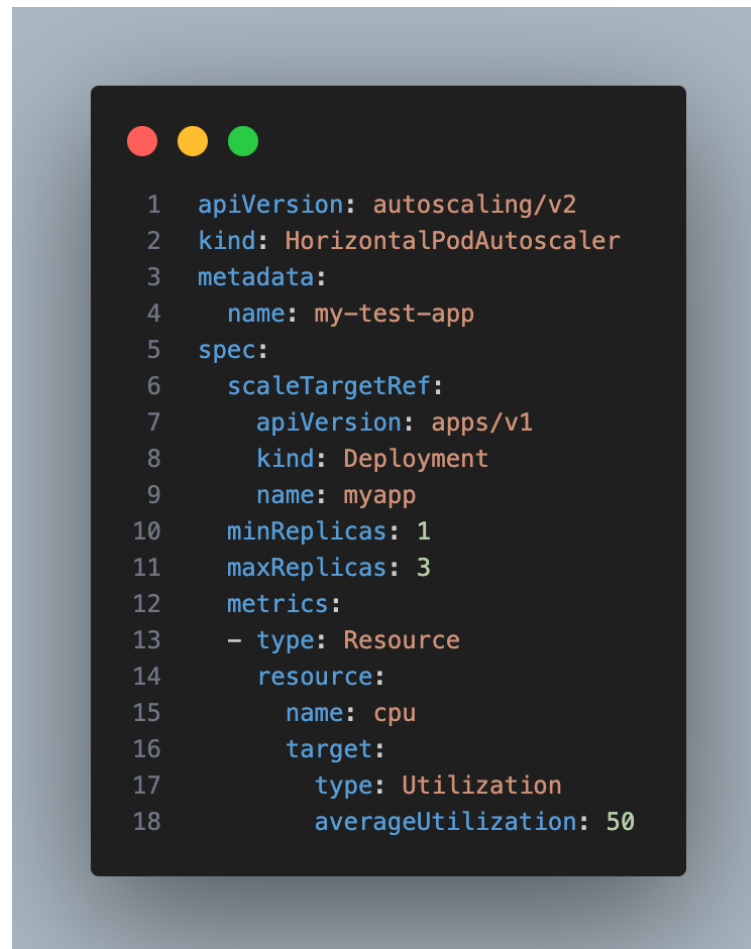


Рис 3.1.2 Базове налаштування HPA для Deployment

Припустимо, що HPA для додатку налаштований у кластері в парі з Cluster autoscaler. Нехай в нас буде ситуація, коли наявних вузлів буде недостатньо для розміщення 3 репліки. В такому випадку ця репліка буде перебувати в стані Pending, Cluster autoscaler створить новий вузол і ця репліка буде розгорнута там. І знову ж таки, коли навантаження CPU буде менше 50%, то репліки будуть видалятися і Cluster autoscaler також видалить і непотрібний вузол. Що потрібно враховувати в роботі Cluster autoscaler, це те, що коли він тригерить масштабування і створюється новий вузол і пулі вузлів, наприклад, AKS, то цей вузол буде мати аналогічну конфігурацію, що й попереді. І Scheduler розгорне на новий вузол тільки Pods, що були у стані Pending. Тобто іноді є шанс, що в кластері будуть створюватись вузли, з ресурсами, що будуть простоювати. Важливо також зазначити, що для роботи HPA потрібно, щоб в кластері був

встановлений компонент, який здатен відображати ці метрики (CPU, memory), наприклад, metrics-server. Також можна використовувати Prometheus.

Схожим чином до HPA буде працювати і KEDA, за виключенням найголовнішої різниці – це тип триггеру, який в KEDA спирається не на метрики CPU чи memory, а на певні події.

Для вертикального автоматичного масштабування було розглянуте відповідне рішення – Vertical Pod Autoscaler (VPA). VPA може працювати в 4-х режимах:

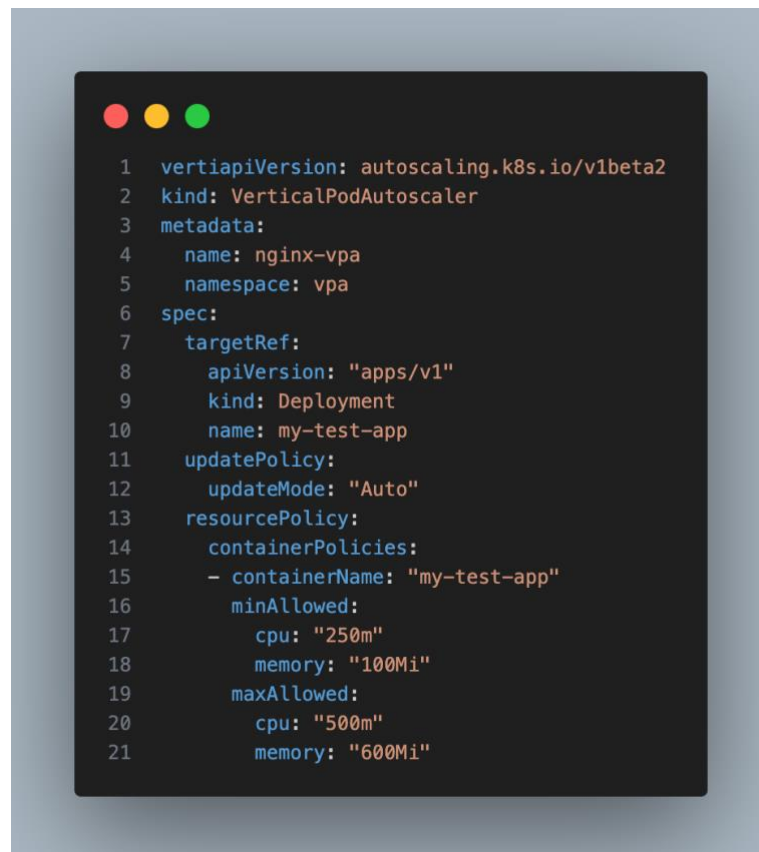
Off – VPA лише надаватиме рекомендації, але не змінюватиме автоматично вимоги до ресурсів.

Initial – VPA призначає запити на ресурси лише при створенні подів і ніколи не змінює їх пізніше.

Recreate – VPA призначає запити на ресурси під час створення кадру та оновлює їх для існуючих кадру, виселяючи та створюючи їх заново.

Auto mode – Створює Pod на основі рекомендацій.

На рис. 3.1.3. розглянуто приклад використання VPA в режимі Auto.



```
1 vertiapiVersion: autoscaling.k8s.io/v1beta2
2 kind: VerticalPodAutoscaler
3 metadata:
4   name: nginx-vpa
5   namespace: vpa
6 spec:
7   targetRef:
8     apiVersion: "apps/v1"
9     kind: Deployment
10    name: my-test-app
11  updatePolicy:
12    updateMode: "Auto"
13  resourcePolicy:
14    containerPolicies:
15    - containerName: "my-test-app"
16      minAllowed:
17        cpu: "250m"
18        memory: "100Mi"
19      maxAllowed:
20        cpu: "500m"
21        memory: "600Mi"
```

Рис. 3.1.3. Конфігурація VPA в режимі Auto

Варто зазначити, що stateful додатки та системи, що не толерують рестартів компонентів з використанням VPA в режимі Auto мають шанс втратити дані або або ж сесія кінцевого користувача обірветься.

Конфігурація для PodDisruptionBudget виглядає досить просто. На рис.3.1.4 ми вказуємо, що мінімум один Pod має бути доступним.

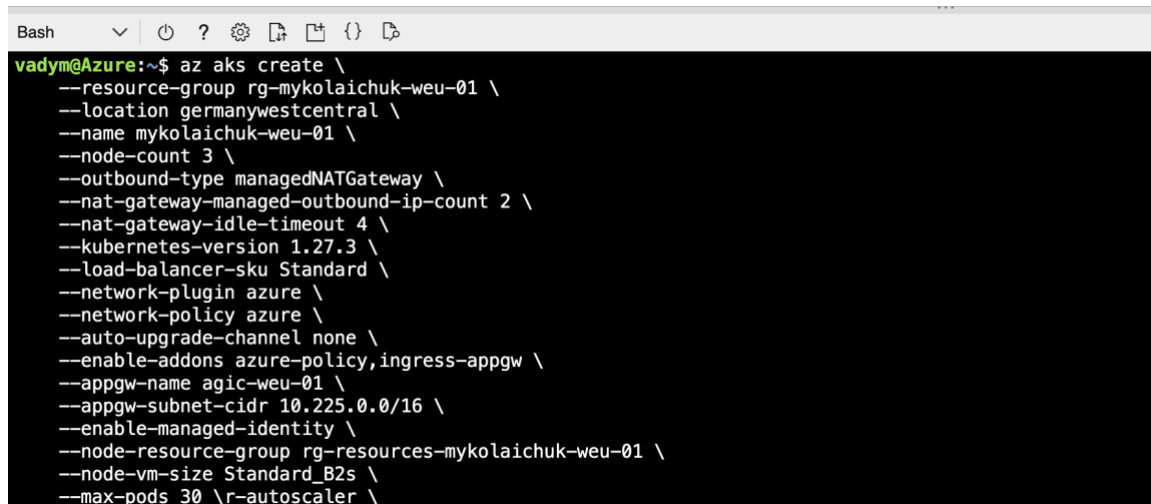


Рис. 3.1.4 Конфігурація PDB

Варто зазначити, що використовуючи PDB адміністратори кластеру можуть допустити помилки, вказавши, наприклад, в конфігурації Deployment однакову кількість Pod з мінімальною кількістю Pod в PDB, що має бути доступна в PDB. В такому випадку, наприклад, при оновленні кластеру, коли виконується операція безпечного видалення вузла зі старою версією Kubernetes (draining), eviction такого ресурсу не вдасться через конфлікт і операція draining завершить по таймауту з помилкою.

3.2. ДОСЛІДЖЕННЯ ПРАКТИК ПОКРАЩЕННЯ ВІДМОВСТІЙКОСТІ КЛАСТЕРУ AZURE KUBERNETES SERVICE

Створимо кластер AKS на основі проаналізованих в підрозділі 2.2, складових в Azure, використовуючи Azure CLI в Azure Cloud Shell (див. рис.3.2.1) та перевіримо основні твердження щодо інфраструктури. Azure CLI, або командний інтерфейс Azure — це набір інструментів командного рядка, який надається Microsoft для керування ресурсами Azure.



```
Bash
vadyim@Azure:~$ az aks create \
  --resource-group rg-mykolaichuk-weu-01 \
  --location germanywestcentral \
  --name mykolaichuk-weu-01 \
  --node-count 3 \
  --outbound-type managedNATGateway \
  --nat-gateway-managed-outbound-ip-count 2 \
  --nat-gateway-idle-timeout 4 \
  --kubernetes-version 1.27.3 \
  --load-balancer-sku Standard \
  --network-plugin azure \
  --network-policy azure \
  --auto-upgrade-channel none \
  --enable-addons azure-policy,ingress-appgw \
  --appgw-name agic-weu-01 \
  --appgw-subnet-cidr 10.225.0.0/16 \
  --enable-managed-identity \
  --node-resource-group rg-resources-mykolaichuk-weu-01 \
  --node-vm-size Standard_B2s \
  --max-pods 30 \r-autoscaler \
```

Рис. 3.2.1 Створення кластеру AKS за допомогою Azure CLI в Azure Cloud Shell

Як було зазначено в підрозділі 2.2 SLA Standard tier для AKS становить 99.9%, а якщо для пули вузлів кластеру використовують Availability Zone, то SLA буде становити 99.95%. На жаль, перевірити SLA експериментально не є можливим, але можна перевірити правильність застосованих конфігурацій, наприклад, через Azure Portal (рис. 3.2.2-3.2.3).

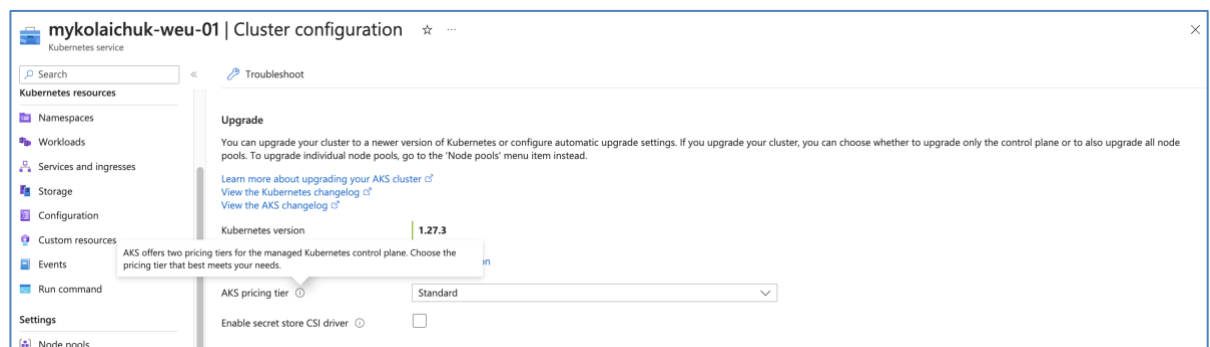


Рис. 3.2.2 Перевірка AKS pricing tier

```
vadym@Azure:~$ kubectl describe nodes | grep -e "Name:" -e "topology.kubernetes.io/zone"
Name: aks-nodepool1-25541466-vmss000000
topology.kubernetes.io/zone=germanywestcentral-1
Name: aks-nodepool1-25541466-vmss000001
topology.kubernetes.io/zone=germanywestcentral-2
Name: aks-nodepool1-25541466-vmss000002
topology.kubernetes.io/zone=germanywestcentral-3
```

Рис. 3.2.3 Перевірка розташування вузлів в Availability Zones

Як бачимо на рис.3.2.2 -3.2.3 – налаштування здійснено вірно, AKS SKU та Availability Zones були застосовані вірно.

В результаті аналізу, проведеному в підрозділі 2.2, та рекомендовано застосовувати пропріетарні рішення Azure CNI та Azure Network Policy Manager.

На рис. 3.2.4 зображені налаштування мережі кластеру.

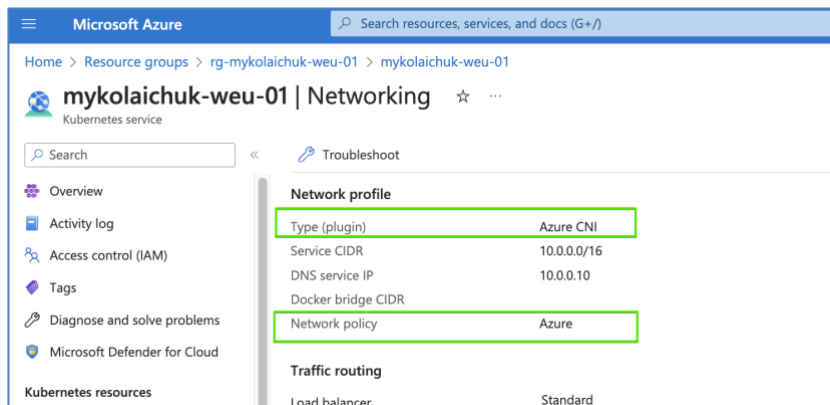


Рис. 3.2.4 Перевірка мережевих налаштувань кластеру

Перевіримо налаштування Ingress Controller в створеному кластері Azure Kubernetes Service. Відповідно до підрозділу 2.2, було розглянуто варіант з імплементацією Application Gateway Ingress Controller, в якості add-on кластеру.

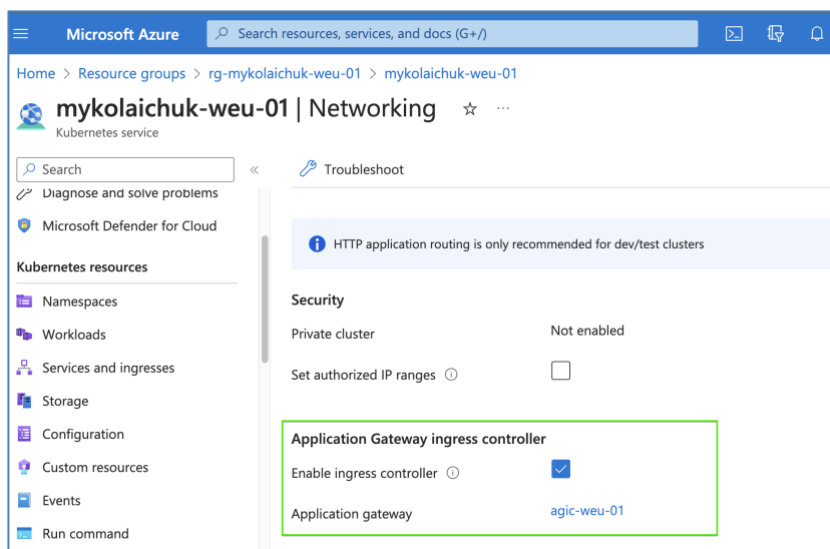


Рис 3.2.5 Перевірка конфігурації Application Gateway Ingress Controller

Як бачимо в рис.3.2.5, Application Gateway Ingress Controller було успішно створено та приєднано до кластеру AKS. Далі перевіримо його базовий функціонал, розгорнувши простий додаток та спробуємо отримати до нього доступ за допомогою протоколу HTTP (рис.3.2.6-3.2.7).

```
vadym@Azure:~$ kubectl apply -f https://raw.githubusercontent.com/Azure/application-gateway-kubernetes-ingress/master/docs/examples/aspnetapp.yaml
pod/aspnetapp created
service/aspnetapp created
ingress.networking.k8s.io/aspnetapp created
vadym@Azure:~$ kubectl get ingress
NAME          CLASS          HOSTS          ADDRESS          PORTS          AGE
aspnetapp     azure-application-gateway  *             20.79.135.214    80            15s
```

Рис. 3.2.6 Розгортання додатку в кластері AKS та перевірка доступних ресурсів Ingress

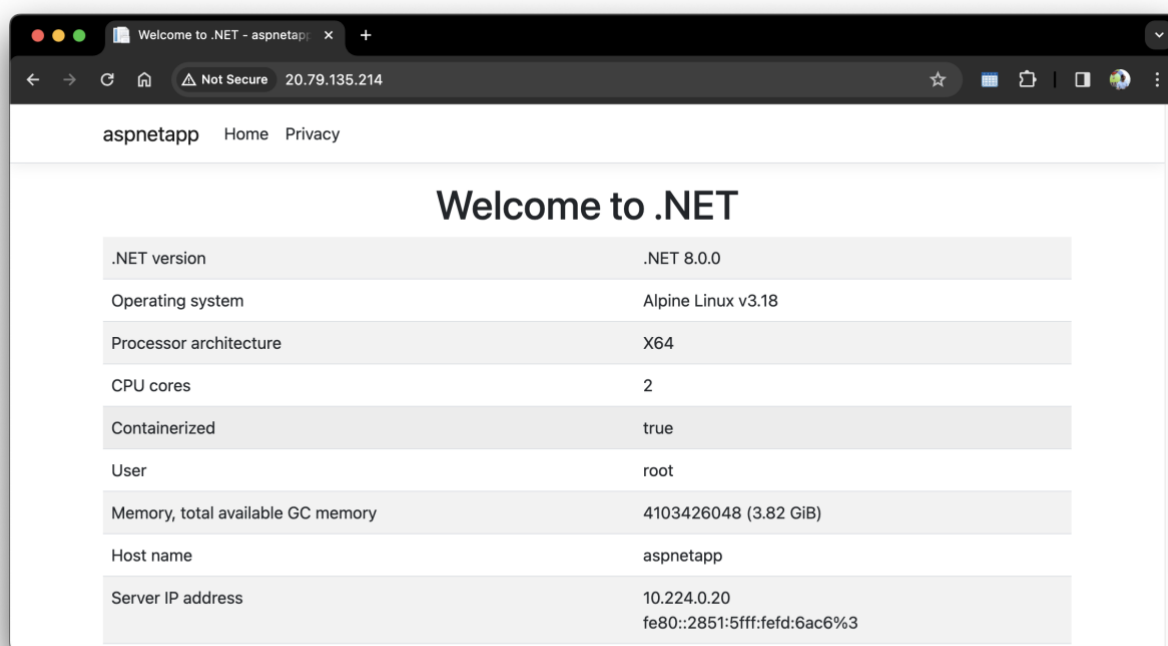


Рис. 3.2.7 Отримуємо доступ до створеного додатку по HTTP, використовуючи IP-адресу AGIC

Перевіримо також налаштування рішення Azure NAT Gateway, який використовується для вихідного трафіку. При створенні кластеру до Azure NAT Gateway було додано дві публічні IP-адреси. Результат перевірки даних налаштувань представлено на рис. 3.2.8-3.2.10.

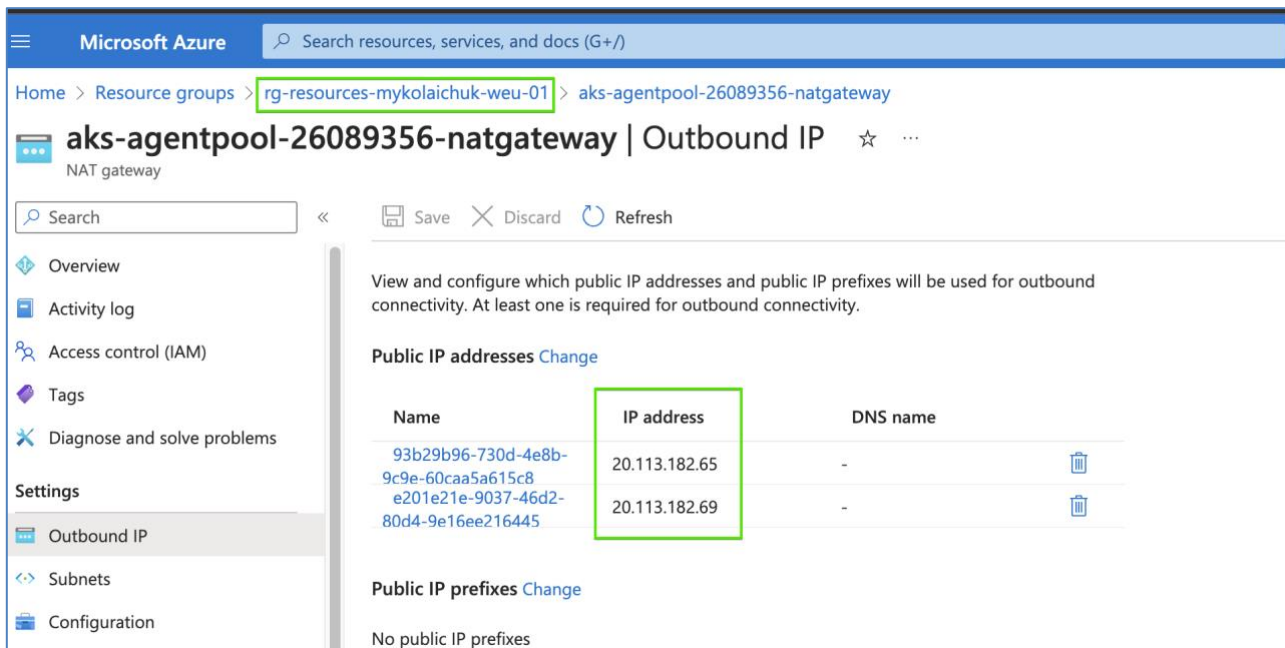


Рис. 3.2.8 Перевірка IP-адрес NAT Gateway

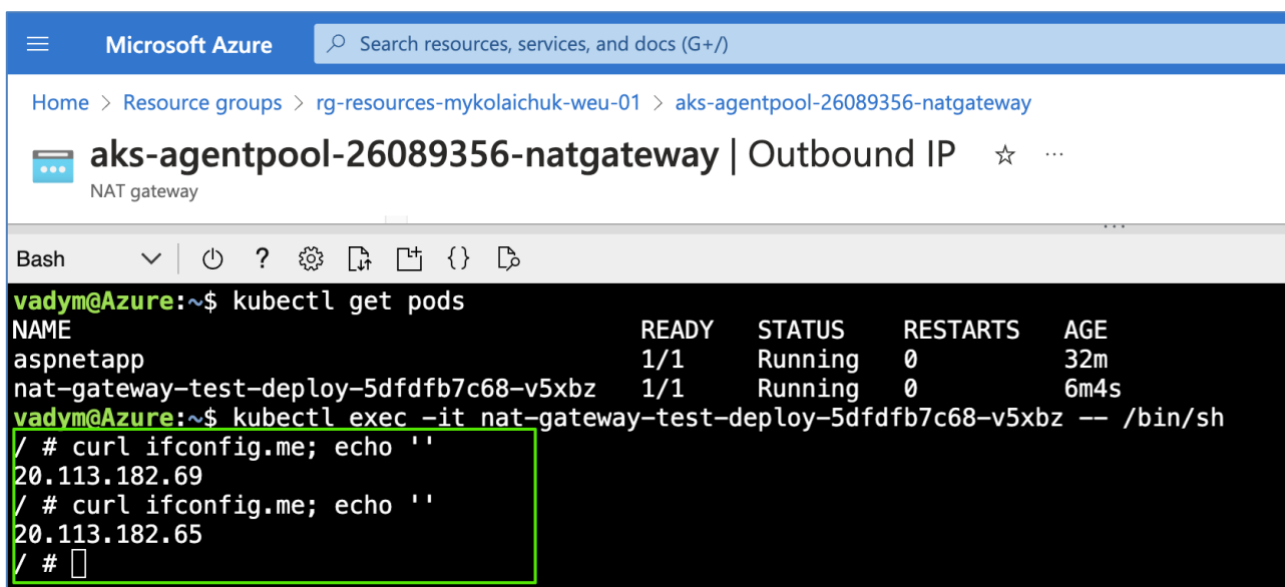


Рис. 3.2.9 Перевірка роботи NAT Gateway з контейнеру Alpine, який розгорнуто в AKS кластері

Також на рис.3.2.8-3.2.9 можливо переконатись, що виконується балансування навантаження вихідного трафіку. При ініціалізації вихідного запиту адреса обирається випадковим чином.

ВИСНОВКИ ДО ТРЕТЬОГО РОЗДІЛУ

В ході виконаних в даному розділі досліджень було перевірено практично рекомендовані підходи для підвищення відмовостійкості інформаційної системи в Linux кластері Azure Kubernetes Service на окремих його складових. Власне в підрозділі 3.2.1 було проаналізовано особливості та поведінку при застосуванні resource requests, resource limits, HPA, VPA, KEDA (частково), PDB. В підрозділі 3.2.2 було проаналізовано та підтверджено, що проаналізовані в 2-у розділі рекомендації для інфраструктури можливо реалізувати на практиці. Перевірено конфігурацію наступних конфігурацій та компонентів кластеру: AKS SKU, Availability Zones, Azure CNI, Azure Network Policy Manager, Application Gateway Ingress Controller, NAT Gateway.

Слід зауважити, що не було можливості перевірити всі вказані технології, конфігурації та рішення, що було згадано в 2-у розділі.

ВИСНОВКИ

В даній кваліфікаційній роботі було досліджено методи підвищення відмовостійкості інформаційної системи на базі кластеру Kubernetes на прикладі Linux кластеру Azure Kubernetes Service.

У першому розділі було детально розглянуто теоретичні відомості про особливості технології контейнеризації, розглянуто складові типового кластеру Kubernetes, та розглянуто особливості кластерів Azure Kubernetes service. Дані теоретичні відомості дали підґрунтя для подальшого аналізу проблематики даної кваліфікаційної роботи у другому розділі.

Власне у другому розділі розглянуто рекомендації для покращення відмовостійкості інформаційних систем у кластері Kubernetes, які справедливі для будь-якої реалізації та розгортання кластеру Kubernetes. Також було досліджено які архітектурні складові кластеру Azure Kubernetes Service можуть підвищити його надійність.

В третьому розділі було перевірено практично рекомендовані підходи для підвищення відмовостійкості, розглянуті у другому розділі рекомендації, як для класичного кластеру Kubernetes, так і його реалізації в Azure Kubernetes Service.

Таким чином основна ціль кваліфікаційної роботи з побудови фреймворку з методами та технологічними рішеннями для підвищення відмовостійкості було виконано. Слід також зауважити, що дані список не є вичерпним і залишає місце для подальших досліджень. Це зумовлено стрімким розвитком технологій, зміною вимог до інформаційних систем, їх архітектурою та випуском все нових і нових рішень від технологічних гігантів, типу Microsoft та незалежних розробників продуктів з відкритим вихідним кодом, що набувають все більшої і більшої популярності.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

- 1.Вікіпедія – Віртуальна машина [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Віртуальна_машина.
- 2.Microsoft Learn – Containers vs. virtual machines [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/virtualization/windowscontainers/about/containers-vs-vm>.
- 3.Cloud Native Wiki by Aqua – 3 Types of Container Runtime and the Kubernetes Connection [Електронний ресурс] – Режим доступу до ресурсу:.
- 4.Cloud Native Wiki by Aqua – containerd: What You Should Know [Електронний ресурс] – Режим доступу до ресурсу: <https://www.aquasec.com/cloud-native-academy/container-platforms/containerd/>.
- 5.Microsoft Azure – What is a private cloud? [Електронний ресурс] – Режим доступу до ресурсу: <https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-a-private-cloud>.
- 6.Microsoft Azure – What are public, private, and hybrid clouds? [Електронний ресурс] – Режим доступу до ресурсу: <https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-are-private-public-hybrid-clouds/>.
- 7.Magic Quadrant for Strategic Cloud Platform Services [Електронний ресурс] – Режим доступу до ресурсу: <http://bit.ly/48z4a8W>.
- 8.Що таке Kubernetes? [Електронний ресурс] – Режим доступу до ресурсу: <https://kubernetes.io/uk/docs/concepts/overview/what-is-kubernetes/>.
- 9.Kubernetes Components [Електронний ресурс] – Режим доступу до ресурсу: <https://kubernetes.io/docs/concepts/overview/components/>.
- 10.Tools for Monitoring Resources [Електронний ресурс] – Режим доступу до ресурсу: <https://kubernetes.io/docs/tasks/debug/debug-cluster/resource-usage-monitoring/>.
- 11.Logging Architecture [Електронний ресурс] – Режим доступу до ресурсу: <https://kubernetes.io/docs/concepts/cluster-administration/logging/>.

12. What is Azure Kubernetes Service? [Электронный ресурс] – Режим доступа до ресурсу: <https://learn.microsoft.com/en-us/azure/aks/intro-kubernetes>.
13. AKS – Control plane [Электронный ресурс] – Режим доступа до ресурсу: <https://learn.microsoft.com/en-us/azure/aks/concepts-clusters-workloads#control-plane>.
14. AKS – Nodes and node pools [Электронный ресурс] – Режим доступа до ресурсу: <https://learn.microsoft.com/en-us/azure/aks/concepts-clusters-workloads#nodes-and-node-pools>.
15. AKS – Resource reservations [Электронный ресурс] – Режим доступа до ресурсу: <https://learn.microsoft.com/en-us/azure/aks/concepts-clusters-workloads#resource-reservations>.
16. AKS – Node pools [Электронный ресурс] – Режим доступа до ресурсу: <https://learn.microsoft.com/en-us/azure/aks/concepts-clusters-workloads#node-pools>.
17. AKS – Node resource group [Электронный ресурс] – Режим доступа до ресурсу: <https://learn.microsoft.com/en-us/azure/aks/concepts-clusters-workloads#node-resource-group>.
18. Kubernetes – Configure Liveness, Readiness and Startup Probes [Электронный ресурс] – Режим доступа до ресурсу: <https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/>.
19. kubecost – Cluster Autoscaler [Электронный ресурс] – Режим доступа до ресурсу: <https://www.kubecost.com/kubernetes-autoscaling/kubernetes-cluster-autoscaler/>.
20. kubernetes / autoscaler [Электронный ресурс] – Режим доступа до ресурсу: <https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#how-does-scale-up-work>.
21. kubernetes / autoscaler [Электронный ресурс] – Режим доступа до ресурсу: <https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#how-does-scale-down-work>.

22.Kubernetes - Pod Disruption Budget [Электронный ресурс] – Режим доступа до ресурсу: <https://kubernetes.io/docs/concepts/workloads/pods/disruptions/#pod-disruption-budgets>.

23.Create an Azure Kubernetes Service (AKS) cluster that uses availability zones [Электронный ресурс] – Режим доступа до ресурсу: <https://learn.microsoft.com/en-us/azure/aks/availability-zones>.

24.What is Application Gateway Ingress Controller? [Электронный ресурс] – Режим доступа до ресурсу: <https://learn.microsoft.com/en-us/azure/application-gateway/ingress-controller-overview>.