МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)
червня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 – Комп'ютерних наук,
освітньо-наукової програми «Інформатика»
на тему: «Система автоматичного запуску та управління контейнеризованими додатками в Kubernetes-кластері»
здобувачки групи ІН-06-2 Ващенко Ольги Олександрівни

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Ольга ВАЩЕНКО
(підпис)

Керівник,
старший викладач кафедри комп'ютерних наук, к.ф.-м.н. Дмитро ВЕЛИКОДНИЙ _____
(підпис)

Суми – 2024

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
на здобуття освітнього ступеня бакалавра
зі спеціальності 122 – Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувачки групи ІН-06-2 Ващенко О. О.

1. Тема роботи: «Система автоматичного запуску та управління контейнеризованими додатками в Kubernetes-кластері» затверджую наказом по СумДУ від «22» квітня 2024 № 0414-VI
2. Термін здачі здобувачем кваліфікаційної роботи до 13 травня 2024 року
3. Вхідні дані до кваліфікаційної роботи
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми предметної області, постановка й формування завдань дослідження.
2) Огляд технологій, що використовуються для розробки інформаційного та програмного забезпечення системи автоматичного запуску та управління контейнеризованими додатками в Kubernetes-кластері. 3) Розробка інформаційного та програмного забезпечення системи автоматичного запуску та управління контейнеризованими додатками в Kubernetes-кластері.
4) Аналіз результатів.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до _____ (підпис)
Керівник _____ _____ (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	Аналіз проблеми предметної області, постановка й формування завдань дослідження		
2	Огляд технологій для розробки системи автоматичного запуску та управління контейнеризованими додатками в Kubernetes-кластері.		
3	Розробка інформаційної системи автоматичного запуску та управління контейнеризованими додатками в Kubernetes-кластері		
4	Аналіз отриманих результатів		
5	Оформлення пояснювальної записки до кваліфікаційної роботи		

Здобувач вищої освіти _____ (підпис)

Керівник _____ (підпис)

АНОТАЦІЯ

Записка: 73 стор., 34 рис., 2 додаток, 42 джерел.

Обґрунтування актуальності теми роботи – тема кваліфікаційної роботи належить до актуальних напрямків розвитку інформаційних технологій, оскільки в сучасному світі велике значення набувають автоматизовані системи управління та розгортання програмного забезпечення. Проблема ефективного управління контейнеризованими додатками в Kubernetes-кластері вирішується розробкою відповідних інформаційних систем, які сприятимуть оптимізації процесів розгортання та моніторингу програмних продуктів.

Об'єкт дослідження – процес автоматичного запуску та управління контейнеризованими додатками в Kubernetes-кластері.

Мета роботи – розробка інформаційної системи для автоматизованого управління контейнеризованими додатками в середовищі Kubernetes. Ця система має на меті забезпечити зручний та ефективний процес розгортання, моніторингу та керування додатками в хмарному середовищі.

Методи дослідження – у роботі використовуються методи аналізу та розробки програмного забезпечення, зокрема використання інструментів контейнеризації, оркестрації, моніторингу, телеметрії та принципів Інфраструктури як коду (IaC).

Результати – розроблено інформаційну систему, яка дозволяє автоматизувати процес запуску та управління контейнеризованими додатками в Kubernetes-кластері. Система включає в себе інструменти для ефективного моніторингу та керування додатками, що дозволяє забезпечити надійну та швидку роботу програмних продуктів в хмарному середовищі.

ІНФОРМАЦІЙНА СИСТЕМА, KUBERNETES, DOCKER, PULUMI, IAC, КОНТЕЙНЕРИЗАЦІЯ, GRAFANA/PROMETHEUS, ОРКЕСТРАЦІЯ, ХМАРНІ СЕРВІСИ, МОНІТОРИНГ, TYPESCRIPT.

Зміст

ВСТУП

Актуальність. У світі швидкого розвитку інформаційних технологій контейнеризація додатків та їх оркестрація в розподілених середовищах стає не лише тенденцією, але й необхідністю. Зростаюча складність та обсяг інфраструктури створюють потребу в ефективному управлінні процесами розгортання та масштабування контейнеризованих додатків. Ця проблема стає особливо актуальною, оскільки швидкість, стабільність та масштабованість інформаційних систем визначають успіх бізнесу сьогодні. Забезпечення безперебійної роботи додатків у зростаючих обсягах та розподілених середовищах стає пріоритетним завданням.

У цьому контексті, система автоматичного запуску та управління контейнеризованими додатками в Kubernetes-кластері виявляється необхідною для оптимізації ресурсів та прискорення процесу розгортання, що важливо для досягнення успіху в сучасному інформаційному середовищі.

Об’єкт дослідження. Процес автоматизованого управління контейнеризованими додатками в Kubernetes-кластері.

Предмет дослідження. Методи та засоби для ефективного запуску, моніторингу та оркестрації додатків у середовищі Kubernetes.

Гіпотеза. Ефективне управління додатками у Kubernetes-кластері можливе завдяки розробці та впровадженню системи, яка буде автоматизовано керувати життєвим циклом додатків з використанням засобів контейнеризації.

Наукова новизна. У порівнянні з існуючими підходами, запропонована система надасть можливість забезпечити більш високу ступінь автоматизації та гнучкість управління додатками, що розгортаються в Kubernetes-кластері.

Структура. Робота складатиметься з вступу, інформаційного огляду, вибору методів для розв’язання задачі, детального огляду програмного та інформаційного забезпечення системи, аналізу результатів та висновків, а також списку використаних джерел і додатків.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Важливість контейнеризації

Контейнеризація – це метод упаковки програми та всіх її залежностей в ізольований контейнер, який може запускатися на будь-якій системі, яка підтримує контейнерну технологію.

Контейнеризація має ряд переваг, які роблять її важливою для сучасного розробки та розгортання програмного забезпечення. Ці переваги включають:

- Легкість і ефективність: Контейнери є легкими і ефективними, що робить їх ідеальними для розгортання та управління розподіленими застосунками.
- Ізоляція: Контейнери ізольовані один від одного, що забезпечує безпеку та стабільність.
- Переносимість: Контейнери можуть запускатися на будь-якій системі, яка підтримує контейнерну технологію.
- Автоматизація: Контейнери можна легко автоматизувати, що полегшує розгортання та управління розподіленими застосунками.

Контейнеризація є більш оптимальною технологією для розгортання програмного забезпечення, ніж традиційні методи, такі як віртуалізація на рівні апаратного забезпечення. Контейнери є легшими і ефективнішими, що робить їх більш економічно вигідними.

Контейнеризовані застосунки, як правило, працюють швидше, ніж традиційні застосунки, розгорнуті на віртуальних машинах. Це пов'язано з тим, що контейнери не вимагають повного набору інструментів та драйверів, які потрібні для віртуальних машин (рис.1.1). Це дозволяє контейнеризованим застосункам використовувати більше ресурсів ядра, що призводить до підвищення продуктивності.

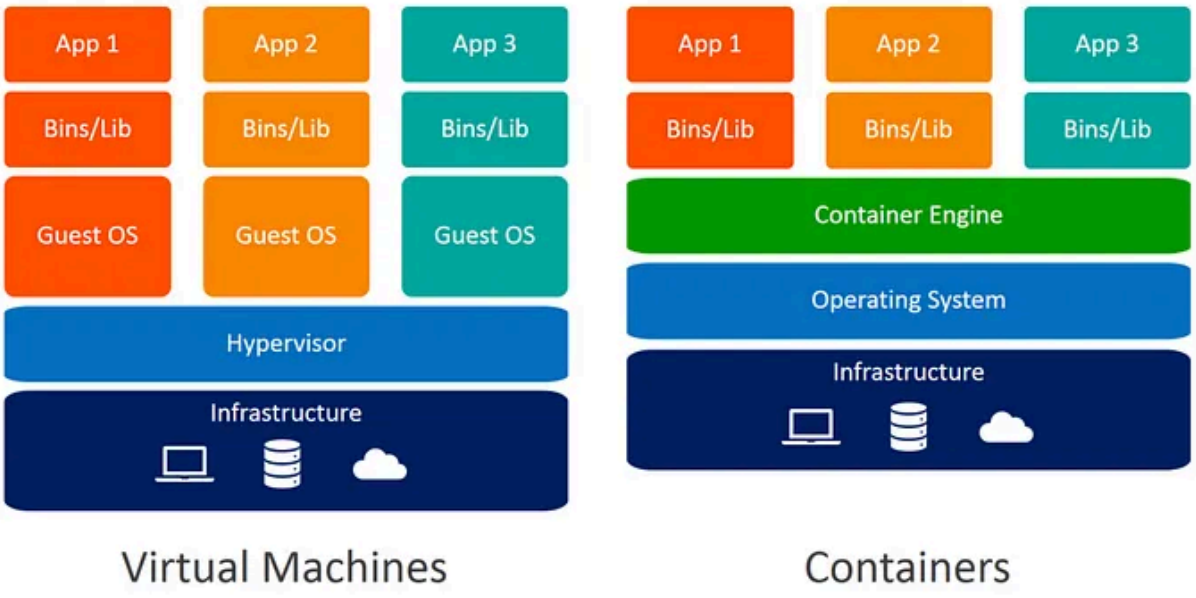


Рисунок 1.1 – Порівняння контейнеризації з віртуалізацією [1]

Контейнеризація дозволяє забезпечити сумісність програм з різними операційними системами та апаратними конфігураціями. Контейнер містить усе необхідне для запуску програми: код, бібліотеки, налаштування, змінні середовища тощо. Таким чином, програма може працювати однаково на будь-якому комп’ютері, який підтримує контейнерний двигун. Це спрощує розгортання, оновлення та масштабування програм.

Контейнеризація сприяє безпеці та стабільності програм. Контейнери ізолюють програми одна від одної, що запобігає конфліктам залежностей, помилкам сумісності та втручанню зловмисників. Контейнери також дозволяють легко створювати резервні копії, переносити та відновлювати стан програм. Контейнери також полегшують тестування та налагодження програм, оскільки вони гарантують однакове середовище на всіх етапах розробки.

Отже, контейнеризація має багато переваг, таких як сумісність з різними системами, продуктивність та ефективність використання ресурсів, безпека та стабільність програм, а також дозволяє розробникам, адміністраторам та користувачам легше створювати, розгортати та управляти розподіленими застосунками.

1.2 Управління і оркестрація

У сучасному світі розгортання та управління мікросервісами і контейнеризованими додатками стає все складнішим завданням. Спростити цей процес і забезпечити ефективне функціонування системи допомагає оркестрація, що є ключовим елементом в сучасних інфраструктурах для розгортання. Оркестрація надає можливість автоматизованого розподілу ресурсів та керування життєвим циклом контейнерів, що робить розробку та експлуатацію додатків більш ефективною та надійною.

Управління і оркестрація – це процеси, які дозволяють координувати, контролювати та автоматизувати різні компоненти системи, що працюють разом. У сучасному світі програмного забезпечення все більше використовуються архітектури, які розбивають систему на окремі частини, які можуть працювати незалежно одна від одної. Такі частини називаються клієнтами, серверами або мікросервісами.

Клієнт-серверна архітектура – це модель, в якій один або декілька клієнтських комп’ютерів з’єднуються з центральним сервером через мережу. Клієнти виконують запити до сервера, а сервер виконує запити та повертає відповідь. Така архітектура дозволяє розподілити навантаження між клієнтами та сервером, а також спростити розробку та супровід програмного забезпечення.

Мікросервісна архітектура – це модель, в якій система складається з багатьох невеликих сервісів, які працюють незалежно один від одного та спілкуються за допомогою легковагових протоколів. Кожен сервіс виконує певну функцію та може бути розгорнутий, масштабований та оновлюваний окремо від інших. Така архітектура дозволяє покращити продуктивність, гнучкість та надійність системи, а також сприяє модульності та розподіленості.

Оскільки клієнт-серверна та мікросервісна архітектури складаються з багатьох компонентів виникає потреба в управлінні і оркестрації.

Управління – це процес, який забезпечує функціонування окремих компонентів системи, таких як моніторинг, налаштування, безпека, резервне копіювання тощо.

Оркестрація – це процес, який забезпечує спільну роботу різних компонентів системи, таких як розгортання, масштабування, балансування навантаження, маршрутизація тощо (рис 1.2).

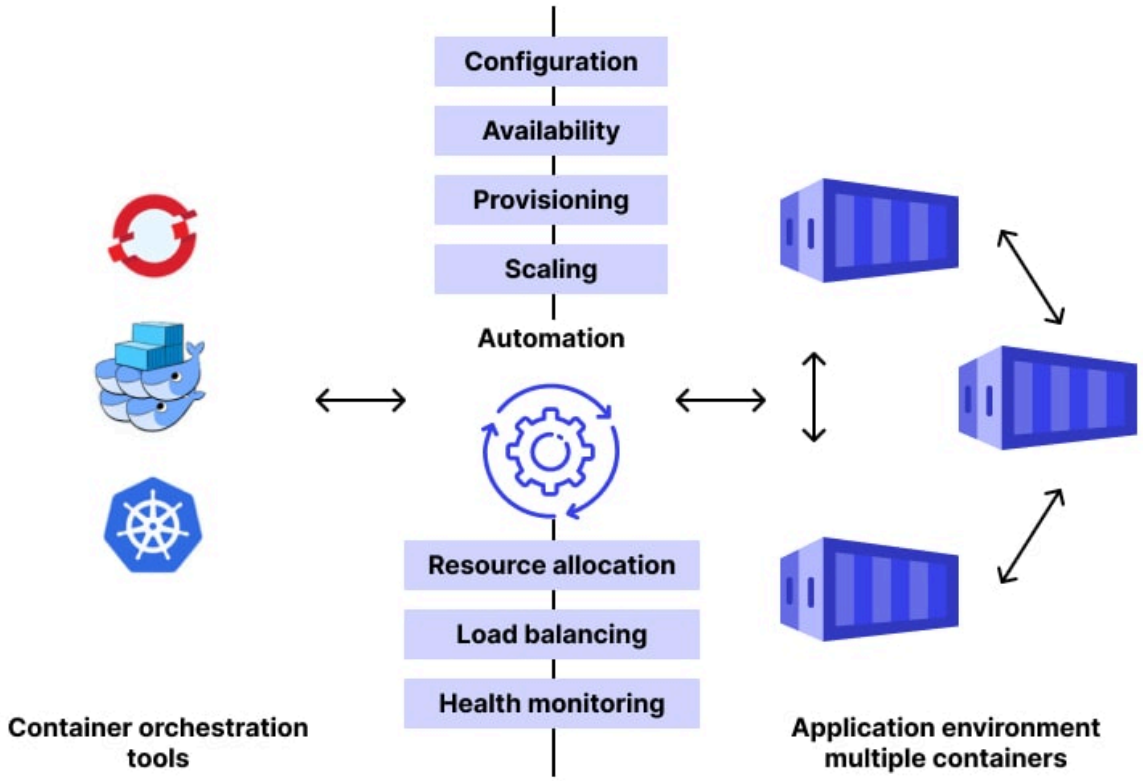


Рисунок 1.2 – Робочий процес оркестратора [2]

1.3 Принцип IaC (Інфраструктура як код)

Автоматизація розгортання інфраструктури є важливим етапом у розвитку і сучасному управлінні IT-проектами.

IaC [3], або інфраструктура як код, – це підхід до керування інфраструктурою в комп'ютерних системах, де вся конфігурація і параметри інфраструктури визначаються у вигляді програмного коду, тобто замість

ручного налаштування серверів і ресурсів стає можливим автоматизувати цей процес за допомогою скриптів на різних мовах програмування.

IaC полегшує інфраструктурне управління та розгортання нових ресурсів. Замість того, щоб витрачати години на ручне створення серверів, мереж, баз даних тощо, розробники і адміністратори можуть сконфігурувати всю цю інфраструктуру за допомогою коду, що робить цей процес більш ефективним, швидким і репродуктивним.

IaC (Інфраструктура як код) функціонує на основі принципів "декларативного програмування" і "принципу поточного стану системи".
Декларативне програмування визначає бажаний стан інфраструктури, представляючи її у вигляді конфігураційних файлів або коду. Основна ідея полягає в тому, щоб описати, як має виглядати інфраструктура, а не як досягти цього стану через послідовність кроків.

Принцип поточного стану системи включає в себе зберігання інформації про поточний стан інфраструктури. Ця інформація описує, як система функціонує на даний момент і зазвичай зберігається у спеціальних файлах або системах управління станом.

Після визначення бажаного стану інфраструктури, система IaC автоматично порівнює його з поточним станом інфраструктури. Вона визначає ресурси, які потрібно створити, змінити або видалити, щоб забезпечити відповідність новому стану. Це відбувається автоматично, без необхідності вручну втручатися. Далі, система автоматично виконує необхідні дії для досягнення бажаного стану інфраструктури (рис.1.3).

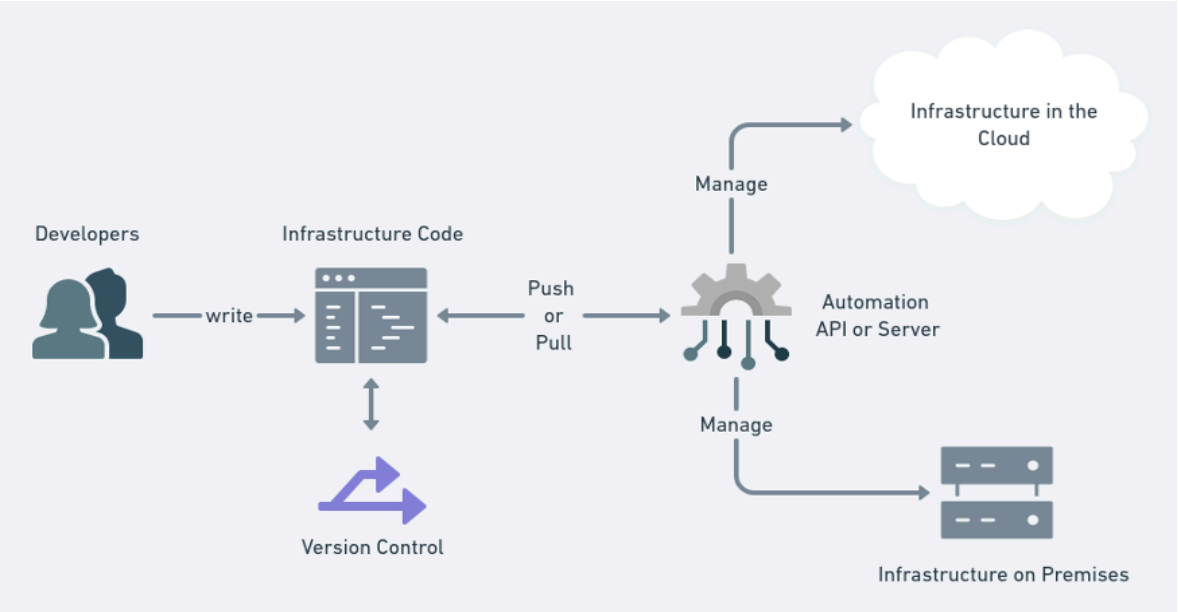


Рисунок 1.3 – Робочий процес управління інфраструктурою через код [4]

Інфраструктура як код є ключовим елементом сучасних практик розробки і управління інфраструктурою, дозволяючи підвищити продуктивність, стабільність і безпеку комп'ютерних систем.

1.4 Моніторинг та телеметрія

Моніторинг та телеметрія – це важливий компонент будь-якої сучасної системи або процесу, що забезпечує збір, аналіз і інтерпретацію даних для ефективного управління та прийняття рішень. Ці технології дозволяють отримувати в реальному часі важливу інформацію про стан системи, її параметри, а також відстежувати різноманітні параметри, такі як температура, тиск, вологість, рівень навантаження тощо.

Моніторинг та телеметрія представляють собою важливі аспекти розвитку та підтримки додатків, які допомагають забезпечити їх ефективну та надійну роботу.

Моніторинг [5] – це процес систематичного спостереження за роботою додатків з метою виявлення можливих проблем або несправностей. Він включає в себе збір та аналіз даних про продуктивність, навантаження та інші параметри

функціонування додатків. Моніторинг дозволяє оперативно виявляти та усувати проблеми, що можуть виникнути, що сприяє покращенню якості та стабільності додатків.

Телеметрія – це збір та відправлення даних про використання додатків, їх функціональність та взаємодію з користувачами. Ці дані можуть включати в себе інформацію про відгуки користувачів, час використання, взаємодію з різними функціями тощо. Телеметрія дозволяє розробникам краще розуміти потреби користувачів та вчасно вносити відповідні зміни та покращення до додатків.

Переваги систем з моніторингом та телеметрією:

- Підвищення ефективності та продуктивності: Шляхом постійного спостереження та аналізу продуктивності додатків можна виявити потенційні проблеми та оптимізувати їх роботу для досягнення максимальної ефективності.
- Зменшення витрат часу та ресурсів: Раннє виявлення проблем дозволяє уникнути витрат часу та ресурсів на їх виправлення у випадку, якщо вони стануть більш серйозними пізніше.
- Покращення користувацького досвіду: Шляхом усунення проблем та несправностей, виявлених через моніторинг, можна підвищити задоволення користувачів від використання додатків та покращити їх досвід.
- Мінімізація ризиків та підвищення надійності: Моніторинг дозволяє попереджати можливі проблеми та відмови, тим самим зменшуючи ризики втрати даних або перерв у роботі.

Усі ці процеси є важливими складовими в розробці додатків, оскільки вони допомагають забезпечити їх стабільну та ефективну роботу, а також підтримувати високий рівень задоволення користувачів. Шляхом постійного

моніторингу та аналізу телеметрії розробники можуть швидко реагувати на зміни у вимогах користувачів та ринкових умов, що сприяє підвищенню конкурентоспроможності додатків.

1.5 Використання хмарних сервісів

Хмарні сервіси – це інфраструктурні, платформенні або програмні рішення, які надаються через Інтернет та розміщені на серверах замість локальних комп'ютерів або серверів. Вони дозволяють користувачам отримувати доступ до різних ресурсів та функціональності за допомогою хмарного середовища.

Важливість використання хмарних сервісів:

- Масштабованість: Хмарні сервіси дозволяють легко масштабувати інфраструктуру та ресурси відповідно до потреб.
- Вартість: Використання хмарних сервісів може допомогти зменшити витрати на придбання, налаштування та підтримку апаратного забезпечення.
- Доступність: Хмарні сервіси надають можливість отримувати доступ до даних та програмних продуктів з будь-якого місця та пристрою, підключеного до Інтернету.
- Безпека: Багато хмарних провайдерів вкладають значні зусилля в забезпечення безпеки даних та дотримання відповідних стандартів.
- Швидкість розгортання: Хмарні сервіси дозволяють швидко розгортати та налаштовувати рішення, що полегшує введення нових продуктів або оновлення існуючих.

- Переваги використання хмарних сервісів:

- Ефективність витрат: Оплата лише за використані ресурси дозволяє зменшити витрати на інфраструктуру та підтримку.

- Доступність та мобільність: Користувачі можуть працювати з даними та програмами з будь-якого місця та пристрою.

- Масштабованість та гнучкість: Хмарні сервіси легко масштабуються, що дозволяє адаптувати інфраструктуру під зростаючі потреби.

- Забезпечення безпеки і конфіденційності: Багато провайдерів володіють високими стандартами безпеки та пропонують засоби шифрування даних.

- Швидкість та продуктивність: Можливість швидкого розгортання та відмінна продуктивність дозволяють підприємствам ефективно працювати з додатками та сервісами.

Хмарні сервіси стають ключовою складовою для багатьох сфер діяльності, надаючи компаніям можливість досягти ефективності, надійності та інновацій. Вони перетворюють спосіб, яким бізнеси працюють та взаємодіють з технологією, і надають засоби для реалізації амбіційних цілей у цифровому світі (рис.1.3).

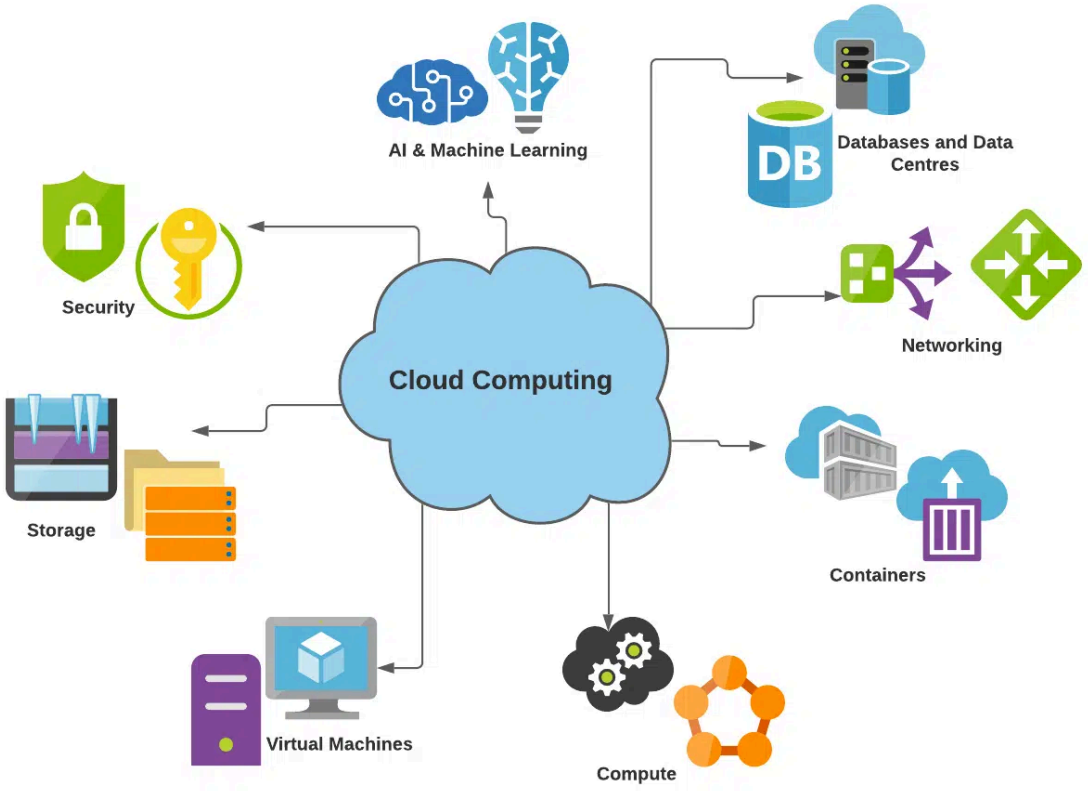


Рисунок 1.3 – Хмарні обчислення [6]

1.6 Постановка задачі

Метою роботи є розробка системи автоматичного запуску та управління контейнеризованими додатками в Kubernetes-кластері для підрозділу Сумського державного університету. Ця робота входить у склад практичного застосування технологій хмарних обчислень, автоматизації та відповідає вимогам студентів бакалавратури зі спеціальності "Комп'ютерні науки".

Система має забезпечувати наступні функціональні можливості:

- Автоматичний запуск контейнерів на Kubernetes-кластері.
- Моніторинг та управління ресурсами кластера для оптимального використання обчислювальних потужностей.

- Масштабування додатків в залежності від навантаження та потреб користувачів.
- Забезпечення надійності та доступності веб-додатків за допомогою автоматичного відновлення та резервування контейнерів.

Для досягнення поставлених цілей необхідно виконати наступні завдання:

- Визначити оптимальну архітектуру для розгортання додатків у Kubernetes-кластері.
- Обрати технології для автоматизації розгортання та управління контейнеризованими додатками.
- Реалізувати систему управління контейнерами, використовуючи Kubernetes API та інструменти автоматизації.
- Протестувати систему автоматичного запуску та управління контейнеризованими додатками.

Предметом дослідження є методи та інструменти автоматизованого розгортання та управління контейнеризованими додатками в Kubernetes-кластері.

Наукова новизна полягає в розробці ефективної системи автоматизованого управління додатками у Kubernetes-середовищі, що використовує сучасні підходи до оркестрації контейнерів.

Практичне значення полягає в підвищенні ефективності та надійності розгортання веб-додатків підрозділу СумДУ за допомогою автоматизованої системи на основі Kubernetes.

2 ВИБІР МЕТОДІВ РОЗВ'ЯЗАННЯ ЗАДАЧІ

2.1 Інструменти для контейнеризації додатків

Для контейнеризації додатків існує багато інструментів, які надають різні функції та можливості. Два найпопулярніших інструменти – це Docker [7] і Podman [8].

Docker – це програмне забезпечення, яке дозволяє створювати, запускати та керувати контейнерами. Контейнер Docker – це легкий виконуваний пакет, який містить все необхідне для запуску програми, таке як код, бібліотеки та змінні середовища. Контейнери Docker відокремлені від головної системи та один від одного, тому вони можуть працювати надійно в різних налаштуваннях хостової системи. Docker також надає інструменти для розповсюдження контейнерних образів через репозиторії, такі як Docker Hub [9].

Особливості Docker:

- Портативність: контейнери Docker можна легко переносити між різними платформами, такими як Linux, Windows або Mac OS. Це полегшує розробку, тестування та розгортання додатків.
- Швидкодія: контейнери Docker запускаються безпосередньо на головному ядрі операційної системи, тому вони споживають менше ресурсів, ніж віртуальні машини. Контейнери Docker також можуть бути запущені паралельно без конфліктів.
- Безпека: контейнери Docker ізольовані один від одного та від головної системи за допомогою простору імен (namespaces) та груп контролю (control groups). Це запобігає несанкціонованому доступу або витоку інформації між контейнерами.

Переваги Docker:

- Спрощення розробки: Docker дозволяє розробникам створювати та запускати додатки у стандартизованому середовищі, яке можна легко відтворити на будь-якому комп’ютері. Це зменшує проблеми сумісності та залежностей.
- Гнучкість розгортання: Docker дозволяє розгортати додатки на будь-якому сервері, який підтримує Docker Engine. Це збільшує доступність та масштабованість додатків.
- Інтеграція з іншими інструментами: Docker сумісний з багатьма іншими інструментами, такими як Kubernetes, OpenShift, Rancher тощо. Це розширює можливості управління та оркестрації контейнерних додатків.

Робочий процес створення контенерів та образів за допомогою Docker зображено на рисунку 2.1.

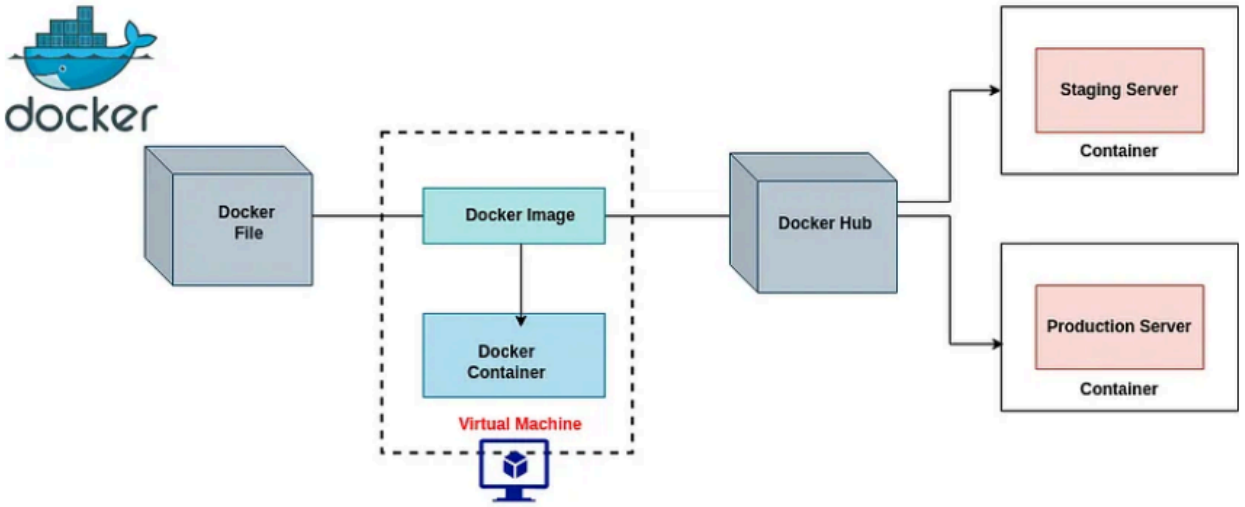


Рисунок 2.1 – Робочий процес Docker [10]

Podman – це альтернатива Docker, яка дозволяє створювати та керувати контейнерами без потреби в демоні Docker. Podman також підтримує Kubernetes YAML для розгортання контейнерних додатків.

Особливості Podman:

- Без демона: Podman не потребує демона Docker для запуску контейнерів. Замість цього, він використовує бібліотеку libpod, яка надає API для керування контейнерами. Це зменшує навантаження на систему та покращує безпеку.
- Сумісність з Docker: Podman може використовувати ті ж самі команди, образи та репозиторії, що й Docker. Це полегшує перехід від Docker до Podman.
- Підтримка Kubernetes: Podman може генерувати та застосовувати Kubernetes YAML для розгортання контейнерних додатків. Це спрощує інтеграцію з Kubernetes.

Переваги Podman:

- Безпека: Podman запускає контейнери як звичайні процеси, які належать користувачеві, який їх створив. Це запобігає привілейованому доступу або ескалації привілеїв через демон Docker.
- Мультиплатформність: Podman може запускати контейнери на різних типах операційних систем, таких як Linux, Windows або Mac OS. Це збільшує портативність та гнучкість додатків.
- Спрощення управління: Podman надає інструменти для керування контейнерами на різних рівнях, таких як поди (pods), образи (images), мережа (network) тощо. Це полегшує моніторинг та налаштування контейнерних додатків.

Порівняння Docker і Podman:

- Docker має більш широкий функціонал та популярність, ніж Podman. Docker є стандартом де-факто для контейнеризації додатків та має

багато ресурсів та спільнот для підтримки. Docker також має багато інтеграцій з іншими хмарними сервісами та платформами.

- Podman має багато переваг у питаннях безпеки та простоти використання, ніж Docker. Podman не потребує демона Docker, який може бути потенційною точкою вразливості або проблемою сумісності. Podman також пропонує багато інструментів для спрощення управління контейнерами на різних рівнях.

Docker і Podman – це два потужних інструменти для контейнеризації додатків, які мають свої переваги та недоліки. Вибір між ними залежить від різних факторів, таких як потреби, цілі та уподобання розробників та адміністраторів. Однак, за загальною оцінкою, Docker є кращим та поширенішим інструментом для контейнеризації додатків, тому було обрано саме Docker.

Причини, чому Docker є кращим за Podman:

- Більш зрілий та стабільний: Docker існує з 2013 року та має багатий досвід та експертизу в галузі контейнеризації додатків. Docker є надійним та перевіреним часом рішенням, яке використовується багатьма великими компаніями та організаціями. Podman, з іншого боку, є новим та експериментальним проектом, який був запущений у 2018 році. Podman може мати багато помилок та проблем, які ще не виявлено або не вирішено.

- Більш популярний та підтриманий: Docker має велику та активну спільноту розробників та користувачів, які надають підтримку, поради. Docker також має багато документації, посібників, курсів та ресурсів, які допомагають навчитися та використовувати Docker. Podman, навпаки, має меншу та менш активну спільноту, яка не може надати такого ж рівня підтримки та ресурсів.

- Більш сумісний та інтегрований: Docker сумісний з багатьма іншими інструментами для контейнеризації додатків, такими як Kubernetes [11], OpenShift [12], Rancher [13] тощо. Docker також має багато інтеграцій з іншими хмарними сервісами та платформами, такими як Amazon Web Services (AWS) [14], Microsoft Azure [15], Google Cloud Platform (GCP) [16] тощо.

2.2 Інструменти для оркестрації

Оркестрація контейнерних додатків – це процес автоматичного керування розгортанням, масштабуванням та координацією контейнерних додатків на кластері. Оркестрація контейнерних додатків має багато переваг, таких як забезпечення високої доступності, надійності, продуктивності та безпеки контейнерних додатків, оптимізація використання ресурсів та інфраструктури, спрощення управління та моніторингу контейнерних додатків. Оркестрація контейнерних додатків також сприяє розвитку хмарних або мікросервісних архітектур, які забезпечують гнучкість та модульність.

Для оркестрації контейнерних додатків існує багато інструментів, які надають різні функції та можливості. Два з найпопулярніших інструменти – це Kubernetes і OpenShift.

Kubernetes – це система для оркестрації контейнерних додатків. Kubernetes дозволяє запускати, масштабувати та керувати контейнерами на кластері. Kubernetes також надає сервіси для забезпечення комунікації, балансування навантаження тощо.

Особливості Kubernetes:

- Кластеризація: Kubernetes організовує сервери в кластери, які складаються з вузлів (nodes). Вузли можуть бути фізичними або віртуальними машинами, які запускають контейнери. Контейнери групуються в поди (pods), які є основною одиницею розгортання в Kubernetes. Поди можуть бути

запущені на будь-яких доступних вузлах за допомогою планувальника (scheduler), який враховує ресурси, політики, прив’язки тощо.

- Самовідновлення: Kubernetes стежить за станом подів та вузлів і здатний автоматично відновлювати або перезапускати ті, які зазнали збою, зупинилися або стали недоступними. Kubernetes також може масштабувати поди вгору або вниз в залежності від навантаження або запитів.
- Сервіси: Kubernetes надає сервіси для забезпечення комунікації між подами та зовнішнім світом. Сервіси можуть бути внутрішніми (clusterIP), зовнішніми (nodePort, loadBalancer) або іменованими (headless). Сервіси також дозволяють виявляти поди за допомогою метаданих, таких як мітки (labels) та селектори (selectors).
- Конфігурація: Kubernetes дозволяє керувати конфігурацією контейнерних додатків за допомогою ресурсів, таких як конфігураційні карти (configMaps) та секрети (secrets). Конфігураційні карти дозволяють зберігати та передавати параметри середовища, а секрети дозволяють зберігати та передавати чутливі дані, такі як паролі, ключі тощо.

Переваги Kubernetes:

- Висока продуктивність: Kubernetes забезпечує ефективне використання ресурсів та інфраструктури за допомогою оптимального планування, масштабування та балансування навантаження контейнерних додатків. Kubernetes також покращує швидкодію контейнерних додатків за допомогою сервісного виявлення, маршрутизації та кешування.
- Висока надійність: Kubernetes гарантує високу доступність, стабільність та безпеку контейнерних додатків за допомогою самовідновлення, резервного копіювання, шифрування та ізоляції. Kubernetes також підтримує

безперебійне розгортання, оновлення та відкат контейнерних додатків за допомогою стратегій, таких як rolling update, blue-green/canary deployment тощо.

- Висока гнучкість: Kubernetes дозволяє розгортати та керувати контейнерними додатками на будь-якому типі інфраструктури, такому як власний сервер, віртуальна машина або хмарний провайдер. Kubernetes також підтримує розширення та налаштування за допомогою плагінів, адаптерів, операторів тощо.

Архітектура та компоненти Kubernetes зображено на рисунку 2.2.

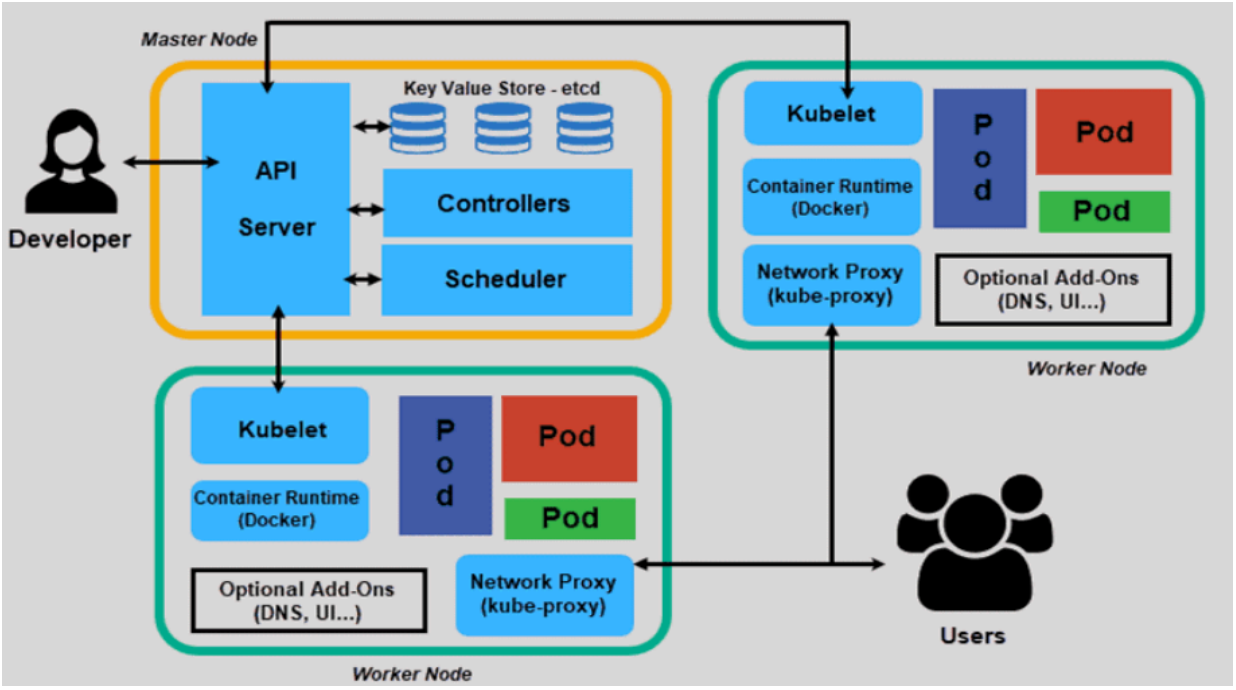


Рисунок 2.2 – Архітектура та компоненти Kubernetes [17]

OpenShift – це платформа для розгортання та управління контейнерних додатків, яка базується на Kubernetes. OpenShift надає додаткові функції, такі як безпека, моніторинг, автоматизація та інтеграція з іншими хмарними сервісами.

Особливості OpenShift:

- Безпека: OpenShift забезпечує високий рівень безпеки контейнерних додатків за допомогою сертифікатів, шифрування, аудиту, політик, ролей та

груп. OpenShift також використовує SELinux для ізоляції контейнерів та запобігання проникненню.

- Моніторинг: OpenShift надає інструменти для моніторингу стану, продуктивності та ресурсів контейнерних додатків. OpenShift також підтримує алерти, сповіщення, логування та трасування для виявлення та вирішення проблем.
- Автоматизація: OpenShift дозволяє автоматизувати процеси розгортання, оновлення, масштабування та видалення контейнерних додатків. OpenShift також підтримує неперервну інтеграцію та доставку (CI/CD) за допомогою Jenkins [18], GitLab [19], GitHub [20] тощо.
- Інтеграція: OpenShift інтегрується з багатьма іншими хмарними сервісами та платформами, такими як AWS, Azure, Google Cloud, IBM Cloud [21] тощо. OpenShift також надає доступ до ринку (marketplace), де можна знайти та встановити готові рішення для контейнерних додатків.

Переваги OpenShift:

- Спрощення розробки: OpenShift спрощує розробку контейнерних додатків за допомогою готових шаблонів, каталогів, консольного доступу та веб-інтерфейсу. OpenShift також надає середовище для локальної розробки (CodeReady Containers) [22], яке емулює хмарний кластер на власному комп’ютері.
- Гнучкість розгортання: OpenShift дозволяє розгортати контейнерні додатки на будь-якому типі інфраструктури, такому як власний сервер, віртуальна машина або хмарний провайдер. OpenShift також підтримує гібридні та багатохмарні архітектури, які дозволяють комбінувати різні типи інфраструктури.

- Розширення функціоналу: OpenShift розширює функціонал Kubernetes за допомогою додаткових компонентів, таких як Operators [23], Service Mesh [24], Serverless [25], Quarkus [26] тощо. Ці компоненти додають нові можливості для керування, оптимізації та інновації контейнерних додатків.

Порівняння Kubernetes і OpenShift:

- Kubernetes є відкритим та стандартним рішенням для оркестрації контейнерних додатків. Kubernetes має багато функцій та можливостей, які дозволяють запускати, масштабувати та керувати контейнерами на кластері серверів. Kubernetes також має велику та активну спільноту розробників та користувачів, які надають підтримку, поради, вчення та сприяння.

- OpenShift є комерційною та розширеною платформою для розгортання та управління контейнерних додатків, яка базується на Kubernetes. OpenShift надає додаткові функції, такі як безпека, моніторинг, автоматизація та інтеграція з іншими хмарними сервісами. OpenShift також надає простоту використання, гнучкість розгортання та розширення функціоналу.

Kubernetes і OpenShift – це два потужних інструменти для оркестрації контейнерних додатків, які мають свої переваги та недоліки. Вибір між ними залежить від різних факторів, таких як потреби, цілі та уподобання розробників та адміністраторів. Однак, за загальною оцінкою, Kubernetes є кращим та поширенішим інструментом для оркестрації контейнерних додатків, тому було обрано саме Kubernetes. Ось деякі причини, чому Kubernetes є кращим за OpenShift:

- Більш відкритий та стандартний: Kubernetes є відкритим проектом, який підтримується фондом Cloud Native Computing Foundation (CNCF) [27]. Kubernetes є стандартом де-факто для оркестрації контейнерних додатків та має багато ресурсів та спільнот для підтримки. OpenShift, з іншого боку, є

комерційною платформою, яка належить компанії Red Hat. OpenShift має менше відкритості та сумісності з іншими інструментами.

- Більш гнучкий та налаштований: Kubernetes дозволяє налаштовувати та розширювати свої функції за допомогою плагінів, адаптерів, операторів тощо.

2.3 Інструменти IaC

Інструменти IaC, або Інфраструктура як код, дозволяють автоматизувати процеси створення та управління обчислювальною інфраструктурою за допомогою коду.

Існує багато інструментів IaC, розглянемо два найпопулярніші з них: Terraform і Pulumi.

Terraform [28] – це інструмент для інфраструктури як коду (IaC), розроблений компанією HashiCorp [29]. Він дозволяє розробникам та інженерам описувати інфраструктуру та ресурси, необхідні для розгортання та управління цією інфраструктурою, у вигляді конфігураційного коду.

Terraform використовує мову опису конфігурації під назвою HCL (HashiCorp Configuration Language). HCL дозволяє описувати ресурси, їх властивості та залежності.

Він підтримує багато провайдерів, включаючи AWS, Azure, Google Cloud, Docker, Kubernetes, та багато інших. Terraform можна використовувати для управління різними типами інфраструктури, включаючи хмарні, локальні та гібридні середовища.

Terraform зберігає стан інфраструктури у файлику terraform.tfstate, що дозволяє відстежувати стан інфраструктури та здійснювати зміни безпечно. Також є можливість створювати модулі, що спрощує перевикористання коду та організацію конфігурації в більших проектах.

Робочий процес створення інфраструктури за допомогою Terraform зображено на рисунку 2.3.

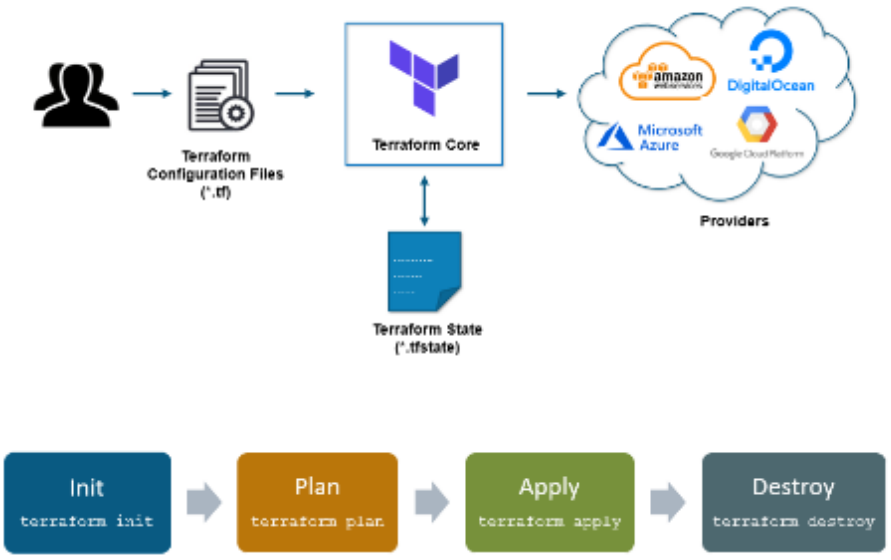


Рисунок 2.3 – Робочий процес Terraform [30]

Terraform – це потужний інструмент для автоматизації інфраструктури, але, як і будь-який інструмент, він має свої недоліки. В Terraform немає обробки помилок, що означає неможливість використання try-catch. Несправні конфігурації або помилки в коді можуть призвести до блокування ресурсів або незрозумілих станів, що вимагатимуть втручання для вирішення.

Також Terraform має обмежену підтримку для обробки окремих дій над ресурсами після їх створення, таких як налаштування чи перевстановлення певних додаткових програм або налаштувань.

У порівнянні з іншими інструментами IaC, Terraform може бути менш зручним для реалізації складної логіки або управління конфігураціями, якщо ця логіка не може бути виражена в межах мови HCL.

Terraform не має вбудованих інструментів для модульного тестування, і тестування коду може виявитися важким завданням.

Крім того, 10 серпня 2023 року HashiCorp оголосив про зміну ліцензії на свої продукти, включаючи Terraform. Приблизно через 9 років після того, як

Terraform став відкритим джерелом за ліцензією MPL v2 [31], його перевели під закриту джерелом ліцензію BSL v1.1 [32], починаючи з наступної 1.6 версії Terraform. Ліцензія BSL не дозволяє використовувати Terraform, якщо це буде конкурувати з HashiCorp і забороняє розміщувати або вбудовувати Terraform у власні продукти. Саме через зміну ліцензії спільнота відвертається від використання Terraform в своїх проєктах.

Pulumi [33] – це відкритий інструмент IaC, який дозволяє користувачам використовувати знайомі мови програмування для створення декларативних конфігурацій, що виключає необхідність вивчення специфічних для постачальника мов шаблонів.

Pulumi підтримує різні мови програмування, включаючи Python [34], JavaScript [35], TypeScript [36], Go [37], C# [38], Java [39], YAML. Це дає розробникам можливість вибрати мову, з якою вони найкраще ознайомлені та якою вони найкраще володіють.

Pulumi підтримує багато хмарних провайдерів, таких як AWS, Azure, Google Cloud, Kubernetes і багато інших, що дозволяє використовувати Pulumi для автоматизації інфраструктури на різних хмарних платформах.

Pulumi дозволяє створювати бібліотеки та модулі для перевикористовувуваного коду, що спрощує роботу з більш складними конфігураціями.

Pulumi має активну спільноту користувачів, особливо після зміни ліцензії Terraform, та комерційну підтримку, яка може надавати допомогу у вирішенні питань та проблем.

Pulumi зберігає інформацію про стан інфраструктури в файлі стану. Цей файл може бути локальним або зберігатися в хмарних службах, таких як AWS S3 або Azure Blob Storage. При кожній операції Pulumi автоматично синхронізує місце для збереження стану з поточним станом інфраструктури. Це дозволяє відстежувати зміни та управляти інфраструктурою. Pulumi підтримує

версіонування файлу стану, що дозволяє відстежувати зміни та відновлювати попередні стани інфраструктури.

Робочий процес створення інфраструктури за допомогою Pulumi зображено на рисунку 2.4.

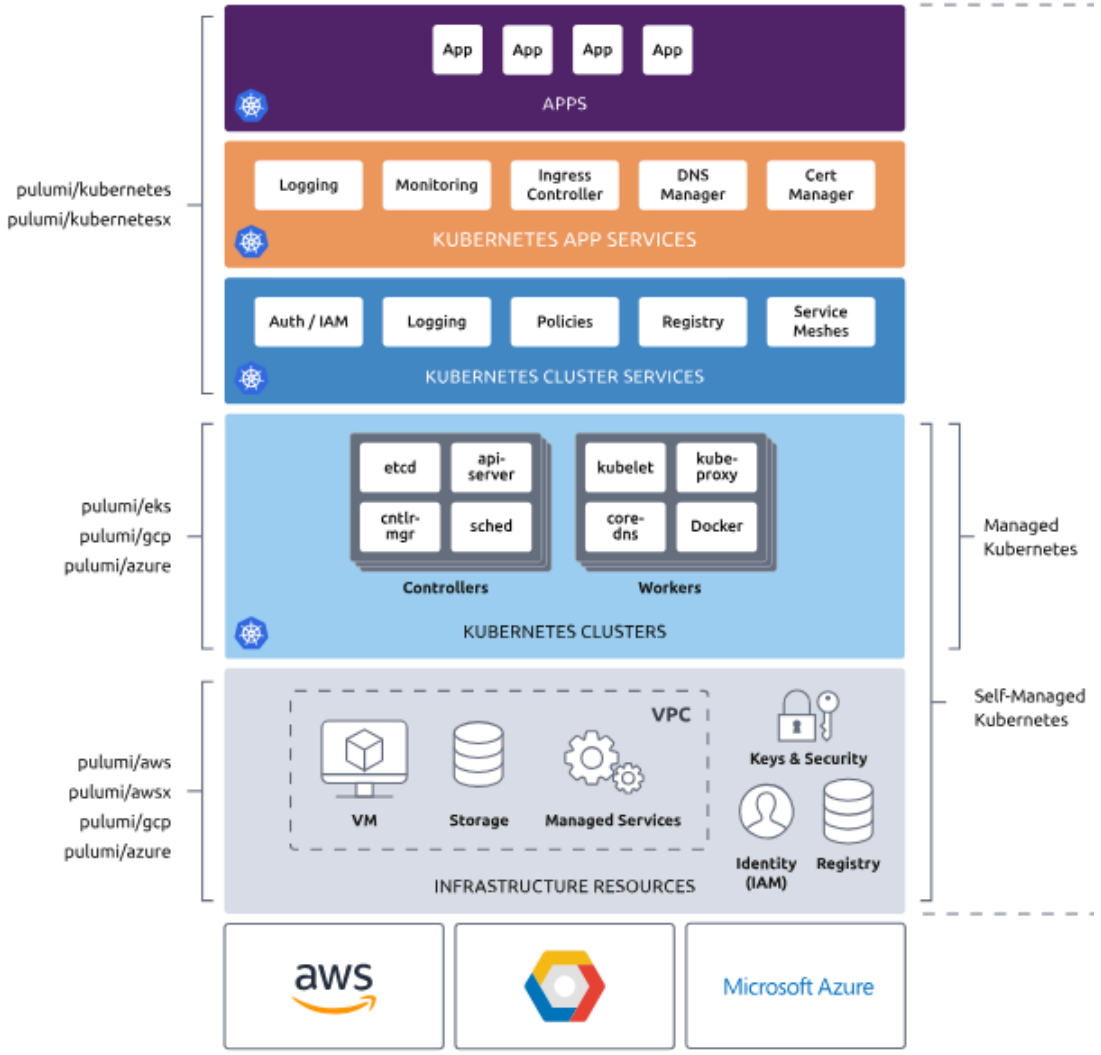


Рисунок 2.4 – Робочий процес Pulumi [40]

Terraform і Pulumi – це два потужні інструменти для опису та управління інфраструктурою як кодом, які мають свої власні переваги та особливості. Вибір між ними залежить від потреб та вподобань розробників та адміністраторів. У даному випадку обрано Pulumi.

Причини, чому Pulumi може бути кращим вибором:

- Простота та зручність: Pulumi використовує сучасні мови програмування, такі як Python, JavaScript, або TypeScript для опису інфраструктури як коду. Це робить процес створення та керування інфраструктурою більш зручним і призначеним для розробників.
- Програмовані ресурси: Pulumi дозволяє створювати власні програмовані ресурси, що надає більше гнучкості та можливостей для налаштування інфраструктури.
- Більша гнучкість у виборі хмарних провайдерів: Pulumi підтримує різні хмарні провайдери, включаючи AWS, Azure, Google Cloud, і багато інших, що дозволяє легко переносити інфраструктуру між різними обліковими записами і середовищами.

2.4 Інструменти для моніторингу та телеметрії

Інструменти моніторингу та телеметрії дозволяють здійснювати нагляд, аналіз та відстеження працездатності, використання ресурсів і загального стану систем. Ці інструменти є критичними для забезпечення надійності та ефективності інфраструктури.

Розглянемо два найпопулярніші інструменти в цій області: Grafana/Prometheus та DataDog.

Grafana та Prometheus – це два потужних інструменти, які часто використовуються в сфері моніторингу та аналітики даних.

Grafana – це інструмент візуалізації даних, який надає можливість створювати красиві та інтерактивні графіки, діаграми, панелі та інші візуальні елементи. Він підтримує різноманітні джерела даних, такі як бази даних, моніторингові системи та API, і дозволяє відображати дані в реальному часі або у вигляді історичних даних. Grafana має дружній інтерфейс користувача та

широкі можливості налаштування, що робить його популярним інструментом для візуалізації метрик та аналізу даних.

Prometheus – це система моніторингу та попередження, яка спеціалізується на зборі, зберіганні та обробці часових рядів даних. Вона забезпечує гнучку та масштабовану архітектуру для збору метрик з різноманітних джерел, таких як вузли, контейнери, програмні додатки тощо. Prometheus також має вбудовану систему тривог, яка дозволяє налаштовувати сповіщення про відхилення метрик від заданих порогів.

Разом Grafana та Prometheus утворюють потужну комбінацію для моніторингу та аналізу даних. Prometheus забезпечує збір та агрегацію даних, в той час як Grafana дозволяє візуалізувати ці дані у зручній формі для аналізу та виявлення залежностей (рис.2.5). Вони часто використовуються разом для створення повноцінного рішення моніторингу та аналітики даних в області інформаційних технологій.

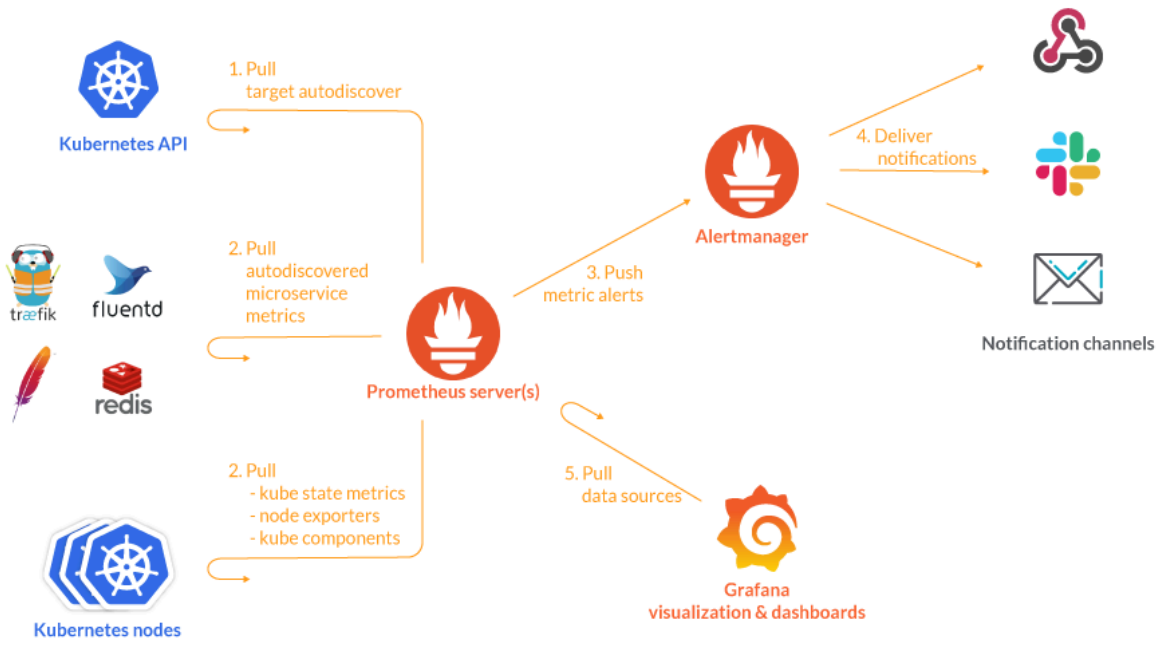


Рисунок 2.5 – Робочий процес Grafana/Prometheus [41]

DataDog – це інтегрована платформа моніторингу, що дозволяє компаніям наглядати за працездатністю та використанням ресурсів своєї інфраструктури. Основною метою DataDog є забезпечення доступності, надійності та ефективності роботи систем.

Ця платформа дозволяє збирати метрики з різних джерел, включаючи сервери, хмарні сервіси, контейнери та додатки. Вона надає інтуїтивний інтерфейс для візуалізації цих метрик у вигляді графіків, діаграм та панелей, що дозволяє оперативно аналізувати стан системи.

Окрім цього, DataDog дозволяє налаштовувати попередження на основі різних метрик та умов, щоб оперативно реагувати на можливі проблеми. Це допомагає забезпечити неперервну роботу системи та уникнути витрат на час та ресурси через недоліки.

Крім моніторингу, DataDog також підтримує аналіз логів, що дозволяє швидко виявляти та вирішувати проблеми за допомогою обробки та аналізу логів подій. Це робить DataDog цінним інструментом для компаній будь-якого розміру, які прагнуть забезпечити надійність та ефективність своїх систем.

- Обидва набори інструментів, Grafana/Prometheus та DataDog, мають

свої переваги, але для середовища Kubernetes може бути кращим вибором Grafana/Prometheus з декількох причин:

- Глибока інтеграція з Kubernetes: Prometheus спеціально розроблений

для моніторингу Kubernetes. Він може автоматично відкривати нові служби та контейнери, що дозволяє миттєво відстежувати зміни у розмірі кластера та стані додатків.

- Гнучкість та масштабованість: Prometheus – це система з відкритим

вихідним кодом, що дозволяє користувачам налаштовувати його згідно зі своїми потребами. Крім того, він легко масштабується від невеликих кластерів до великих інфраструктур.

- Спрощений моніторинг архітектури Kubernetes: Prometheus надає вбудовану підтримку для моніторингу різних компонентів Kubernetes, таких як API-сервер, контролери розгортання, служби, та інші, без необхідності встановлення та налаштування додаткових компонентів.
- Гнучка візуалізація з Grafana: Grafana надає потужні інструменти візуалізації, що дозволяють створювати красиві та інтерактивні графіки для аналізу метрик Kubernetes. Вона інтегрується з Prometheus для зручного відображення даних.

В цілому, Grafana/Prometheus може бути більш привабливим вибором для моніторингу Kubernetes завдяки своїй глибокій інтеграції з цією платформою, гнучкості та широким можливостям налаштування.

2.5 Огляд хмарних провайдерів

Хмарні провайдери – це компанії, які надають послуги зберігання та обробки даних у хмарному середовищі. Ці провайдери стали незамінними у сучасному світі інформаційних технологій, дозволяючи підприємствам та індивідуальним користувачам зберігати та обробляти великі обсяги даних без необхідності власного обладнання та його обслуговування.

На сьогоднішній день існують три найпопулярніші хмарні провайдери, які надають широкий спектр послуг у сфері хмарних обчислень: AWS, Azure, Google Cloud Platform. На рисунку 2.6 наведена діаграма, що ілюструє частку ринку постачальників хмарних послуг на початок 2023 року.

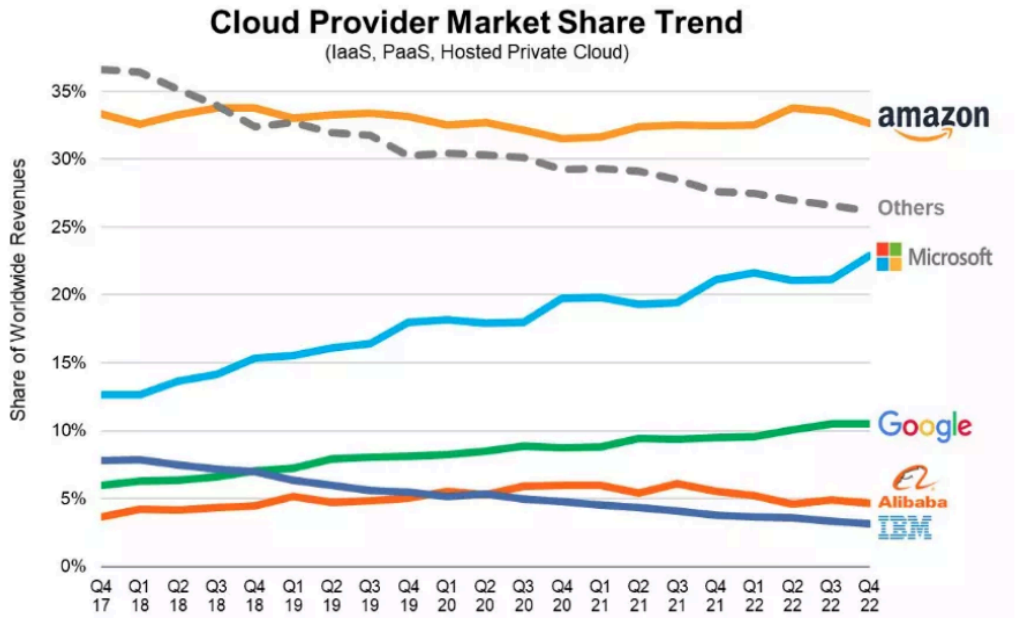


Рисунок 2.6 – Частка ринку постачальників хмарних послуг на початок 2023 року [42]

Amazon Web Services (AWS) – це один з найбільших та найпопулярніших хмарних провайдерів у світі. AWS був запущений компанією Amazon у 2006 році і став важливим гравцем у сфері хмарних обчислень.

AWS пропонує вражаючий асортимент послуг і продуктів, які включають в себе обчислення (EC2), зберігання (S3), бази даних (Amazon RDS), інструменти для розробки, штучний інтелект (Amazon SageMaker) і багато інших.

AWS має дата-центри і регіональні центри обчислень у багатьох країнах світу. Це дозволяє користувачам розміщувати свої додатки та дані в найближчому до їх місця роботи регіоні, забезпечуючи швидкий доступ та надійність.

AWS надає широкий спектр інструментів і служб для забезпечення безпеки інфраструктури та даних. Вони пропонують шифрування, ідентифікацію та автентифікацію, моніторинг безпеки та інші заходи безпеки.

AWS продовжує розвиватися і розширювати свій асортимент послуг, що робить його одним із ключових гравців у галузі хмарних обчислень і надає можливість бізнесам у всьому світі використовувати сучасні технології для свого розвитку та інновацій.

Microsoft Azure – це хмарна обчислювальна платформа від Microsoft, яка надає широкий спектр послуг та ресурсів для обчислень у хмарному середовищі.

Azure пропонує різноманітні хмарні послуги, включаючи обчислення (Virtual Machines), зберігання (Azure Blob Storage), бази даних (Azure SQL Database), машинне навчання (Azure Machine Learning), аналітику даних (Azure Data Lake Analytics) та багато інших. Однією з ключових переваг Azure є інтеграція з іншими продуктами Microsoft, такими як Windows Server, Active Directory і Office 365.

Microsoft має центри обчислень та дата-центри Azure у багатьох країнах та регіонах світу, що дозволяє користувачам розміщувати свої дані і додатки в найближчому до них регіоні, забезпечуючи швидкий доступ і високу доступність.

Microsoft Azure є популярним вибором для підприємств, які шукають хмарні обчислювальні рішення та інфраструктуру, особливо тим, які вже використовують продукти Microsoft у своїй діяльності.

Google Cloud Platform (GCP) – це хмарна платформа для обчислень та послуг, яку надає Google. GCP пропонує широкий спектр хмарних послуг та інфраструктури для розробників, підприємств та дослідників.

GCP відомий своєю експертизою в області машинного навчання та аналізу даних. Вони пропонують інструменти, такі як TensorFlow, для створення і тренування моделей машинного навчання, а також послуги для аналізу даних, такі як BigQuery, для швидкого інтерактивного аналізу великих обсягів інформації.

GCP надає віртуальні машини (Google Compute Engine), контейнери (Google Kubernetes Engine) та інші обчислювальні ресурси для розгортання додатків та сервісів. GCP має послуги для зберігання різних типів даних, включаючи файлове сховище, об'єктне сховище, бази даних (Google Cloud SQL, Firestore, Bigtable) та інші варіанти для зберігання інформації.

GCP надає розширені можливості для розробки та управління проектами IoT, забезпечуючи збір, аналіз та використання даних, що надходять від підключених пристроїв.

Google Cloud Platform широко використовується у різних галузях, включаючи технологічні стартапи, підприємства та дослідницькі організації, і він продовжує розвиватися та розширювати свої можливості для задоволення потреб користувачів.

AWS, Microsoft Azure і GCP – це три великі хмарні платформи, кожна з яких має свої унікальні переваги та особливості.

Проте, за загальною оцінкою, обрано Microsoft Azure через його широкий функціонал і надійність.

Причини, чому Microsoft Azure може бути кращим вибором:

- Масштабність і надійність: Microsoft Azure є однією з найбільших і найпоширеніших хмарних платформ у світі, що гарантує високу доступність та масштабність додатків.
- Широкий набір послуг: Microsoft Azure пропонує широкий спектр хмарних послуг, включаючи обчислення, зберігання, бази даних, машинне навчання, інтернет-речей та багато інших, що дає велику гнучкість у розробці та впровадженні різних типів додатків.
- Глибока інтеграція: Microsoft Azure інтегрується з багатьма іншими сервісами та інструментами, що робить його зручним для використання в різних сценаріях.

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Впровадження хмарних сервісів

Під час впровадження хмарних сервісів було використано хмарний провайдер Azure для розгортання та управління різними інфраструктурними та програмними ресурсами.

Для організації та керування ресурсами, що були використані для розгортання зазначених сервісів, було створено спеціальну ресурсну групу, що дозволило зберігати всі пов'язані ресурси разом, спрощуючи їх управління та моніторинг.

На рисунку 3.1 зображено сторінку створення ресурсної групи.

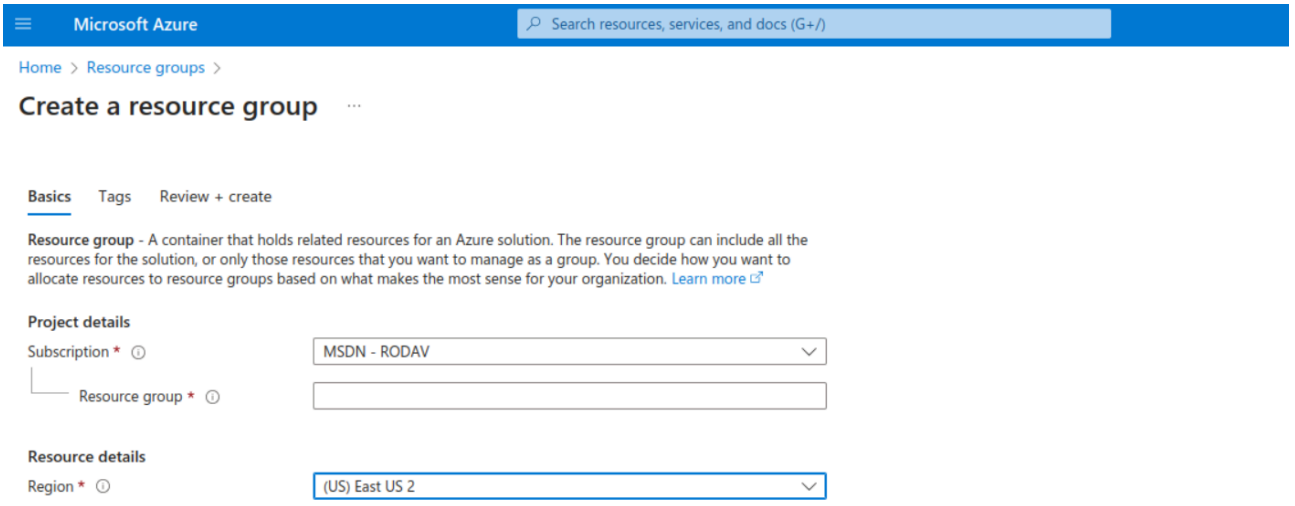


Рисунок 3.1 – Сторінка створення ресурсної групи Azure

Результат створення ресурсної групи зображено на рисунку 3.2.

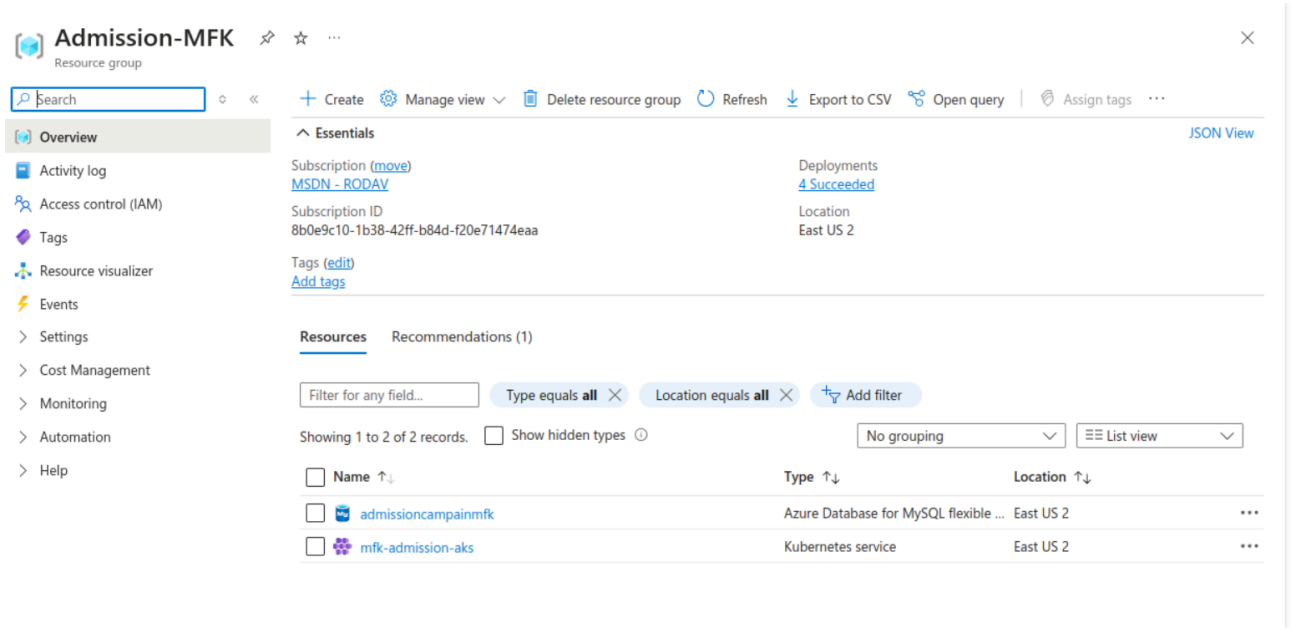


Рисунок 3.2 – Створена ресурсна група

Для ефективного розгортання інформаційної системи на Kubernetes-кластері було обрано сервіс Azure Kubernetes Services (AKS).

Azure Kubernetes Services (AKS) – це керована служба контейнерів, яка надає можливість легко розгортати, керувати та масштабувати контейнеризовані додатки за допомогою оркестратора Kubernetes у хмарному середовищі Azure.

Використання AKS дозволило оптимізувати процес розгортання, управління та масштабування системи, забезпечивши надійність та ефективність роботи. Використання цього сервісу автоматизувало багато аспектів розгортання та управління інфраструктурою, включаючи автоматичне масштабування ресурсів та забезпечення високої доступності. Це дозволило зосередитися на розробці та функціональності системи, мінімізуючи зусилля, витрачені на управління.

На рисунку 3.3 зображено сторінку створення AKS кластеру.

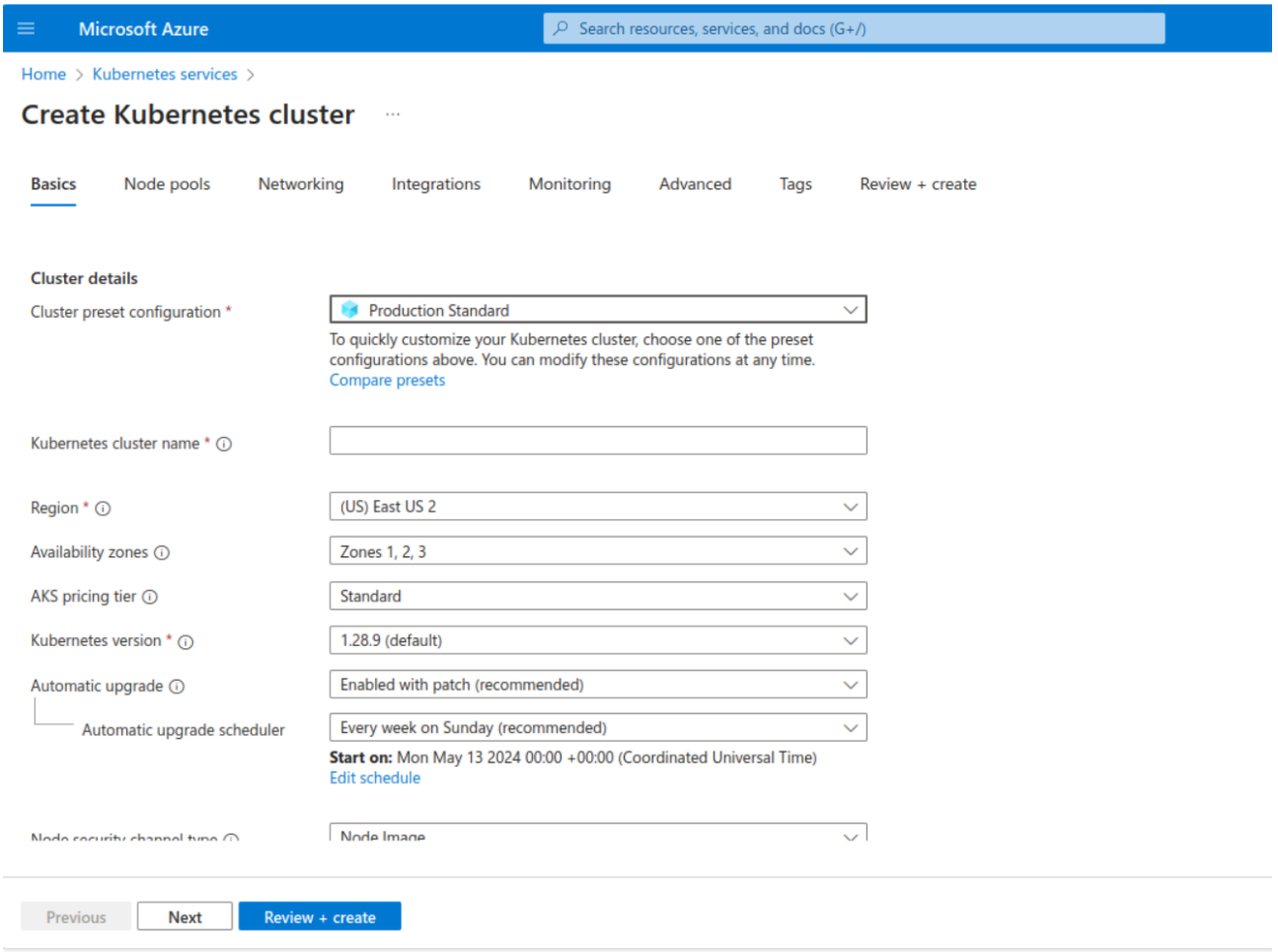


Рисунок 3.3 – Сторінка створення AKS кластеру

На рисунку 3.4 зображено результат створення AKS кластеру для інформаційної системи.

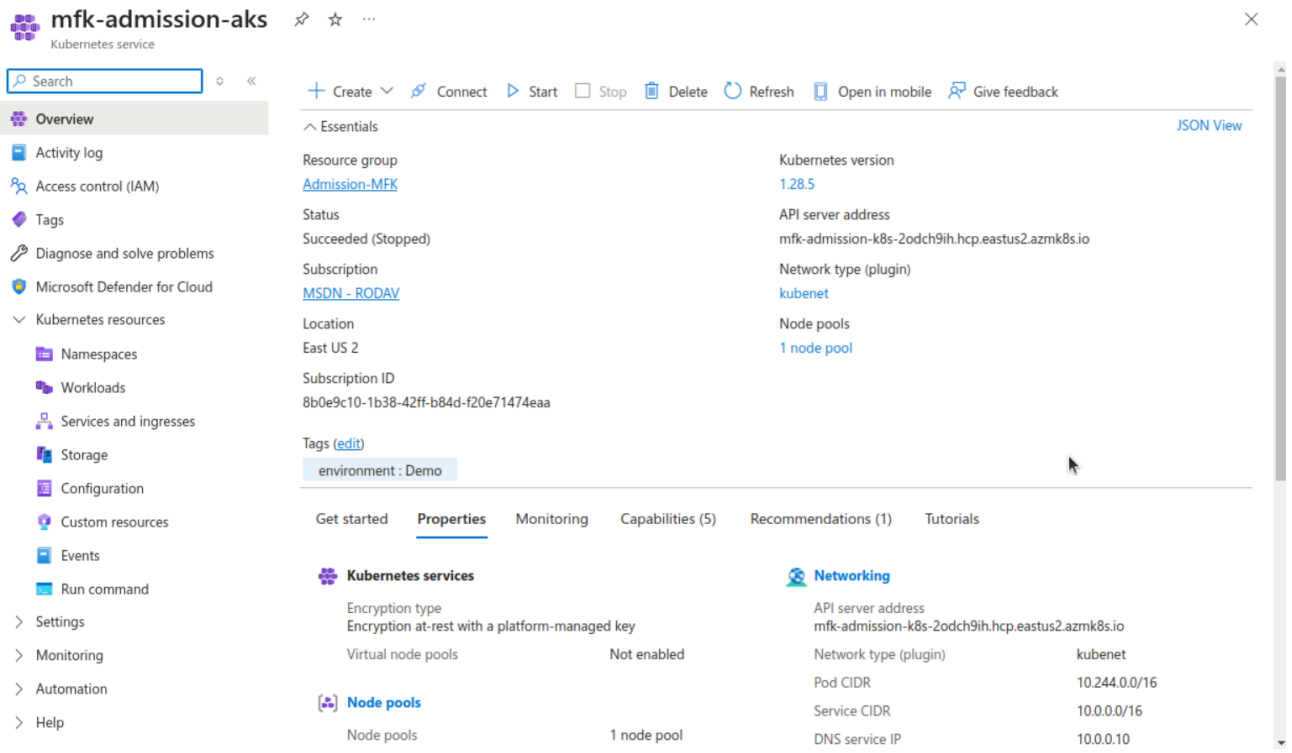


Рисунок 3.4 – Створений AKS кластер

Використання Azure Kubernetes Services стало ключовим фактором в успішному впровадженні інформаційної системи, забезпечивши швидкість, надійність та масштабованість операцій.

Для функціонування інформаційної системи необхідна база даних, тому було обрано скористатись сервісом Azure Database for MySQL.

Azure Database for MySQL – це повністю керована база даних MySQL у хмарному середовищі Azure. Цей сервіс надає можливість створення, управління та масштабування баз даних MySQL без необхідності управління фізичною інфраструктурою. За допомогою Azure Database for MySQL користувачі можуть швидко розгортати нові бази даних, забезпечуючи високу доступність, безпеку та продуктивність своїх додатків. Крім того, цей сервіс автоматично керує резервним копіюванням даних, виконує моніторинг та налагодження, що дозволяє зосередитися на розробці додатків, а не на управлінні базами даних.

Сторінка створення Azure Database for MySQL зображена на рисунку 3.5.

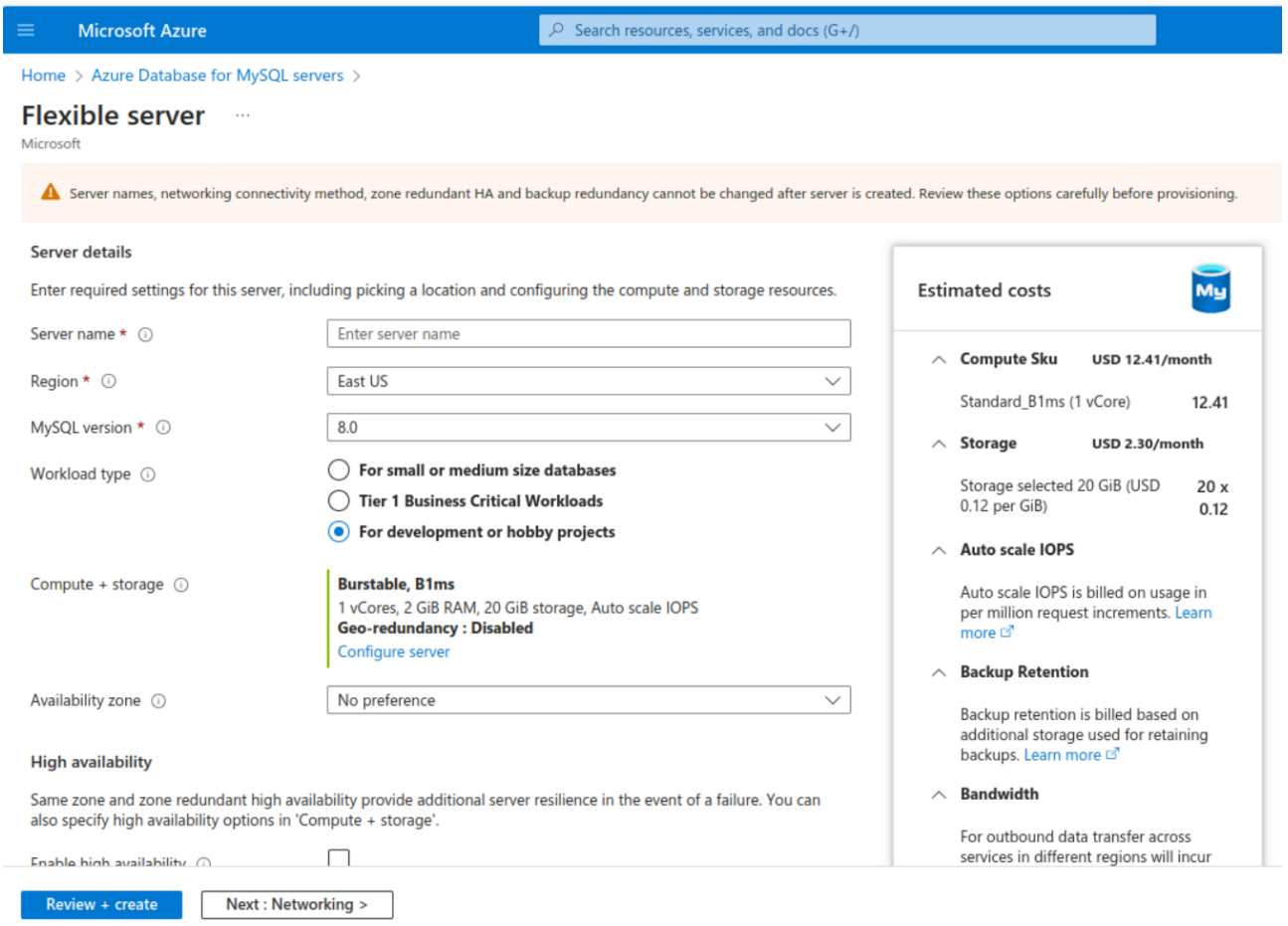


Рисунок 3.5 – Сторінка створення Azure Database for MySQL

Результат створення бази даних в сервісі Azure Database for MySQL для інформаційної системи зображено на рисунку 3.6.

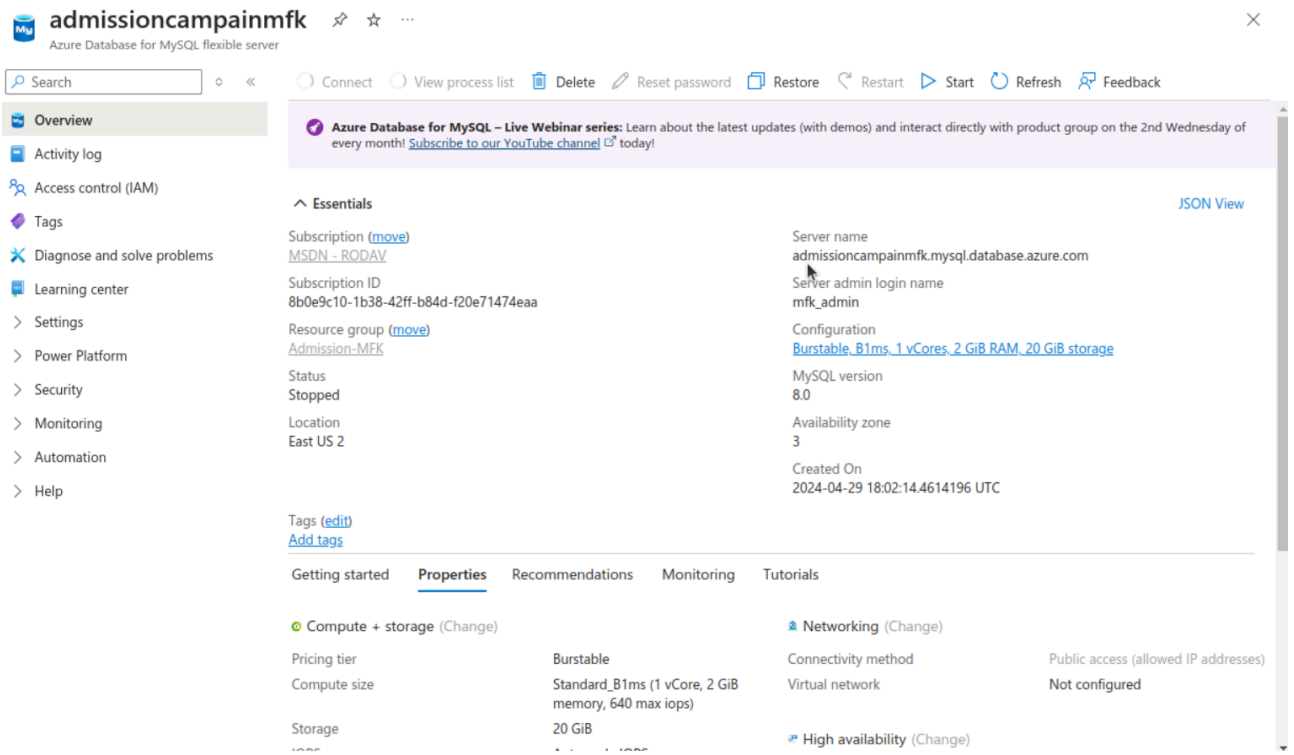


Рисунок 3.6 – Створена база даних Azure Database for MySQL

Інтеграція з Azure Database for MySQL дозволила забезпечити безпеку та надійність для зберігання даних, що використовуються у додатках. Цей сервіс надає розширені можливості керування базами даних у хмарному середовищі, що дозволяє забезпечити високий рівень продуктивності та доступності.

У цілому, використання Azure Kubernetes Services разом з Azure Database for MySQL та відповідне створення ресурсної групи в хмарному середовищі Azure дозволило забезпечити надійність, масштабованість та ефективність інфраструктури для інформаційної системи.

3.2 Контейнеризація інформаційної системи

У роботі було обрано Docker як інструмент для контейнеризації інформаційної системи.

Як додаток для контейнеризації було обрано інформаційну систему, що була розроблена як дипломна робота молодшого спеціаліста [], а саме сайт вступної кампанії МФК СумДУ.

Для забезпечення ефективного розгортання та управління інформаційною системою в будь-якому середовищі було створено конфігураційний файл для Docker. Код Dockerfile наведено в додатку А.

Dockerfile включає такі кроки для побудови образу:

- Базовий образ: Використовуючи debian:stable як базовий образ.
- Оновлення пакетів: Виконується команда apt-get update для оновлення списку пакетів з репозиторіїв Debian.
- Встановлення програмного забезпечення:
  - Apache2: apache2 – веб-сервер.
  - PHP: php – мова програмування для веб-розробки.
  - Модуль Apache для PHP: libapache2-mod-php – дозволяє Apache взаємодіяти з PHP.
  - Розширення PHP для роботи з MySQL: php-mysql – дозволяє PHP взаємодіяти з базами даних MySQL.
  - CLI для PHP: php-cli – командний рядок PHP для виконання сценаріїв.
  - Розширення PHP для роботи з PDO: php-pdo – дозволяє PHP взаємодіяти з різними системами управління базами даних через PDO.
- Активація модуля rewrite: Виконується команда a2enmod rewrite, щоб активувати модуль rewrite для Apache.

- Перезапуск Apache: Виконується команда service apache2 restart для перезапуску сервера Apache та застосування змін, зроблених під час установки та налаштування.
- Робочий каталог: Встановлюється поточний робочий каталог в /var/www/html/ всередині контейнера, де зазвичай знаходяться файли веб-сайту.
- Копіювання файлів: Копіюються файли з поточного каталогу (де розміщений Dockerfile) до каталогу /var/www/html/ всередині контейнера. Це включає всі файли та папки, які можуть бути потрібні для веб-сайту.
- Експонування порту: Об'являється EXPOSE 80, щоб вказати, що контейнер повинен експонувати порт 80 для доступу до веб-сервера Apache.
- Команда запуску: Встановлюється команда запуску CMD ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"], яка запускає Apache в режимі переднього плану всередині контейнера.

Ці кроки разом створюють образ Docker, який містить налаштований веб-сервер Apache з підтримкою PHP та готовий для розгортання веб-сайтів.

Для побудови образу було використано команду docker build. Результат збірки образу зображено на рисунку 3.7.

Рисунок 3.7 – Побудова Docker image

Після створення власного образа для того, щоб його розповсюдити було використано Docker Hub. Це хмарне сховище для Docker-образів, яке дозволяє завантажувати, зберігати та ділитися образами контейнерів.

Використання Docker Hub є корисним для розгортання та управління контейнерами.

Зібраний докер образ інформаційної системи було завантажено до Docker Hub (рис.3.8).

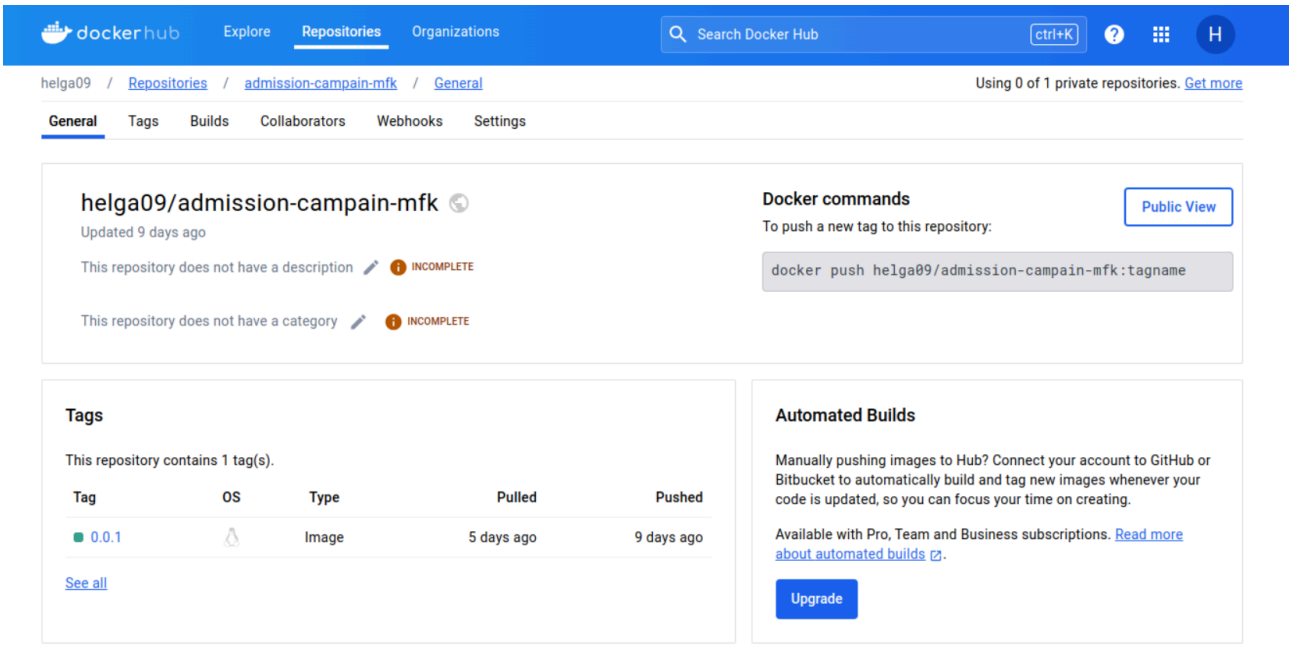


Рисунок 3.8 – Образ в Docker Hub

Завантажений образ інформаційної системи підтверджує успішну роботу з контейнеризацією та готовність до подальшого використання.

3.3 Програмна реалізація IaC

У роботі було обрано Pulumi як інструмент для опису конфігурації інфраструктури. Хоча pulumi підтримує кілька мов програмування, включаючи Python, Go, та інші, було обрано TypeScript через його динамічність і чіткість

коду, що сприяє більшій надійності і зручності у виразі інфраструктурних компонентів.

Ініціалізація проекту з використанням pulumi включає кілька кроків. Перш ніж почати розробку, необхідно встановити pulumi та забезпечити належні налаштування середовища. Для цього можна скористатися офіційною документацією pulumi.

Після успішної установки можна створити новий проект за допомогою командного рядка. Ініціалізація нового проекту з TypeScript шаблоном дозволяє використовувати TypeScript для опису інфраструктурних ресурсів.

Після створення проекту для конфігурації Kubernetes з використанням pulumi, необхідно налаштувати доступ до кластера Kubernetes, з яким буде взаємодіяти pulumi. Це включає налаштування аутентифікаційних даних для доступу до кластера Kubernetes через хмарний провайдер Azure.

Після успішної аутентифікації, pulumi зможе взаємодіяти з кластером Kubernetes, створюючи, оновлюючи та керуючи ресурсами Kubernetes за допомогою коду, написаного на TypeScript. Такий підхід дозволяє легко автоматизувати процеси управління Kubernetes інфраструктурою, забезпечуючи швидке та надійне розгортання та конфігурацію.

Після успішного налаштування pulumi проект має структуру зображену на рисунку 3.9.

Проект містить такі файли:

- .gitignore: Файл, який містить список файлів та каталогів, які система керування версіями Git повинна ігнорувати під час збереження змін.
- index.ts: Файл з кодом програми, написаним на мові програмування TypeScript. Містить код для створення та керування інфраструктурою за допомогою pulumi.

- node_modules/: Каталог, який містить всі залежності проекту, такі як бібліотеки та модулі, які використовуються в проекті.
- package.json: Файл, який містить метадані про проект, такі як назва, версія, залежності та скрипти, необхідні для розробки та виконання проекту.
- package-lock.json: Файл, який містить інформацію про точні версії всіх залежностей проекту, щоб забезпечити повторюваність установки залежностей.
- Pulumi.yaml: Файл конфігурації для pulumi, який містить параметри проекту, такі як назва стеку та облікові дані для підключення до інфраструктури хмарного провайдера.
- tsconfig.json: Файл конфігурації TypeScript, який містить параметри компіляції для проекту, такі як налаштування модулів, шляхів до файлів та параметри типізації.

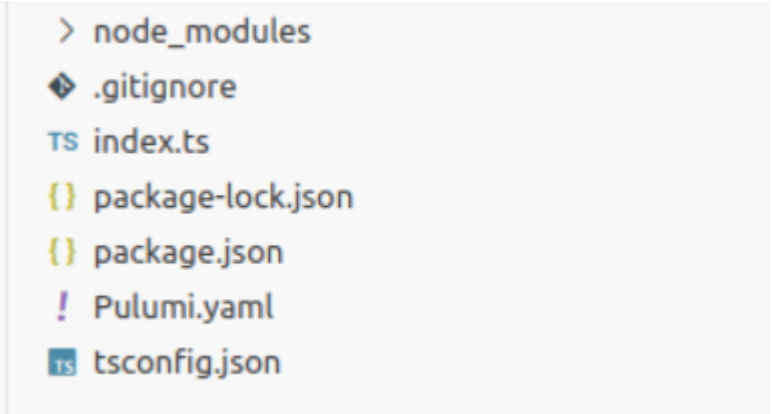


Рисунок 3.9 – Структура pulumi проекту

Файл index.ts містить код програми, написаний на мові програмування TypeScript, який використовується для створення та керування інфраструктурою Kubernetes за допомогою pulumi.

У основному файлі:

- Імпортуються необхідні модулі, які дозволяють взаємодіяти з Kubernetes з використанням pulumi.
- Оголошуються змінні та об'єкти для створення різних ресурсів Kubernetes, таких як Deployment, Service, Secret та інші.
- Встановлюються параметри для кожного ресурсу, включаючи кількість реплік, образ Docker, порти, середовищні змінні тощо.
- Створюються ресурси Kubernetes, такі як Deployment, Service, Secret, Namespace та інші.
- Експортуються значення зовнішніх ресурсів.

Цей файл є основним модулем програми та містить код, який підтримує роботу Kubernetes-інфраструктури через pulumi. Фрагмент коду наведено в додатку Б.

Після написання коду для інфраструктури за допомогою pulumi, можна взаємодіяти з ним через командний рядок. Основні команди включають:

- pulumi up: Ця команда використовується для розгортання чи оновлення інфраструктури згідно з описаним кодом. Pulumi аналізує код та здійснює необхідні зміни в інфраструктурі, щоб відповідати описаному стану.
- pulumi preview: Ця команда використовується для попереднього перегляду змін, які будуть внесені до інфраструктури при виконанні команди pulumi up. Вона дозволяє переглянути прогнозовані зміни, не роблячи реальних змін в інфраструктурі.

- pulumi stack: За допомогою цієї команди можна керувати стеками pulumi, що дозволяє використовувати різні середовища (наприклад, розробка, тестування, виробництво) для керування різними версіями інфраструктури.
- pulumi destroy: Ця команда використовується для знищення інфраструктури, яка була розгорнута за допомогою pulumi. Вона видаляє всі ресурси, створені за допомогою коду.
- pulumi stack output: Ця команда дозволяє переглянути виведені значення, які були експортовані з інфраструктури за допомогою команди export у коді pulumi.

Приклад результату команди розгортання інфраструктури за допомогою коду наведено на рисунку 3.10.

Команда pulumi up діє таким чином:

- Аналіз коду: Pulumi аналізує код проекту, визначаючи всі ресурси і залежності, описані в ньому.
- Планування змін: Після аналізу pulumi створює план змін, що потрібно внести до інфраструктури, щоб забезпечити її відповідність коду.
- Попередній перегляд змін: Pulumi показує вам перелік змін, які планується зробити в інфраструктурі, включаючи створення нових ресурсів, оновлення існуючих або видалення непотрібних.
- Застосування змін: Після підтвердження pulumi починає виконувати зміни, перетворюючи інфраструктуру в стан, описаний у коді.

- Підтвердження розгортання: Після завершення процесу pulumi повідомляє вас про успішне або не успішне завершення операції розгортання, а також надає додаткову інформацію про зміни, які були внесені.

Команда pulumi up дозволяє легко та ефективно управляти інфраструктурою, забезпечуючи її актуальність та відповідність вимогам коду. Інші команди діють схожим чином.

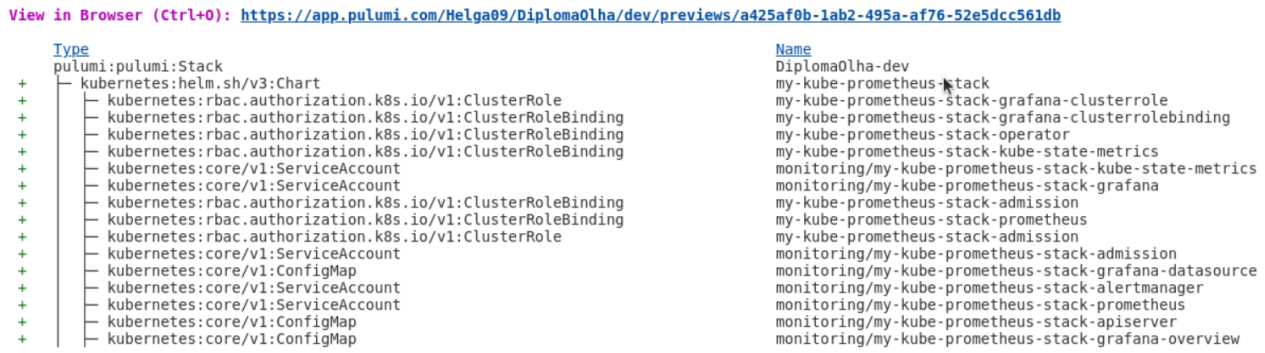


Рисунок 3.10 – Вивід команди pulumi up

За допомогою цих команд pulumi можна легко керувати розгортанням, оновленням, переглядом та видаленням інфраструктури, описаної за допомогою pulumi.

Також Pulumi підтримує не лише рядковий інтерфейс, а й повноцінний веб інтерфейс, який надає графічний інтерфейс для керування інфраструктурними проектами, розгортанням, переглядом стану і багато іншого через веб-браузер.

Основні можливості Pulumi UI включають:

- Створення проектів: Ви можете створювати нові проекти прямо з веб-інтерфейсу, обираючи мову програмування та область хмарних послуг.

- Редагування коду: Pulumi UI надає можливість редагувати код проекту прямо у веб-інтерфейсі, що спрощує внесення змін і додавання нових функцій.

- Розгортання та керування стеками: Ви можете розгортати свої проекти, створювати та керувати стеками pulumi безпосередньо через веб-інтерфейс.
- Попередній перегляд змін: Перед розгортанням ви можете переглянути плановані зміни, щоб переконатися, що вони відповідають очікуванням.
- Моніторинг та журнали: Pulumi UI надає можливість переглядати моніторинг та журнали ресурсів і подій проекту для відстеження його стану та діагностики проблем.
- Керування конфігураціями та секретами: Ви можете керувати конфігураційними параметрами та секретами проекту через інтерфейс користувача.

Pulumi UI доповнює командний рядок, дозволяючи командам та командам розробників працювати з pulumi у зручному та інтуїтивно зрозумілому середовищі.

На рисунку 3.11 наведено приклад однієї з вкладок з Pulumi UI для аналізу ресурсів.

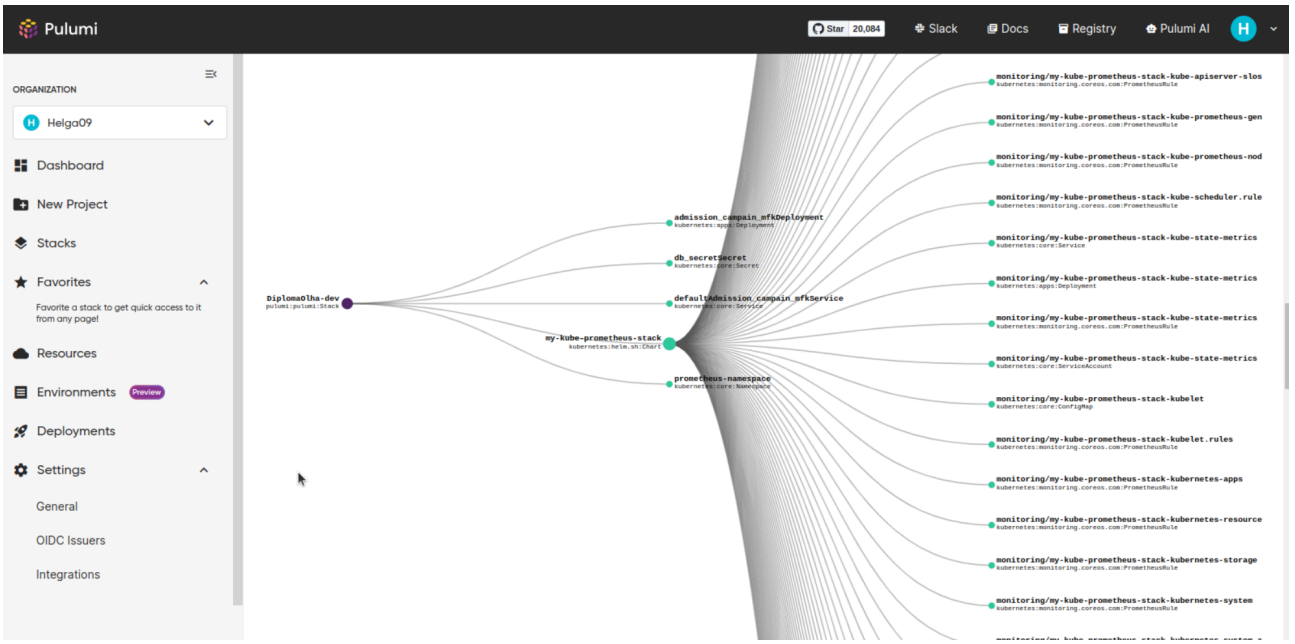


Рисунок 3.11 – Аналіз ресурсів в Pulumi UI

3.4 Інтеграція системи моніторингу

У роботі було обрано Grafana/Prometheus для інтеграції системи моніторингу. Цей вибір зроблено з урахуванням ефективності та широких можливостей аналізу даних.

Для реалізації цієї системи моніторингу було також використано підхід Infrastructure-as-Code за допомогою інструменту pulumi.

Pulumi підтримує створення Helm charts, які є популярними засобами для управління пакунками та ресурсами Kubernetes. Використаємо Pulumi для генерації, налаштування та розгортання Helm charts безпосередньо з коду. Це робить процес створення та управління Kubernetes-ресурсами більш зручним та автоматизованим, дозволяючи вам легко налаштовувати та розгортати додатки в Kubernetes-середовищі.

Для розгортання моніторингу використаємо chart Kube-prometheus-stack (рис. 3.12).

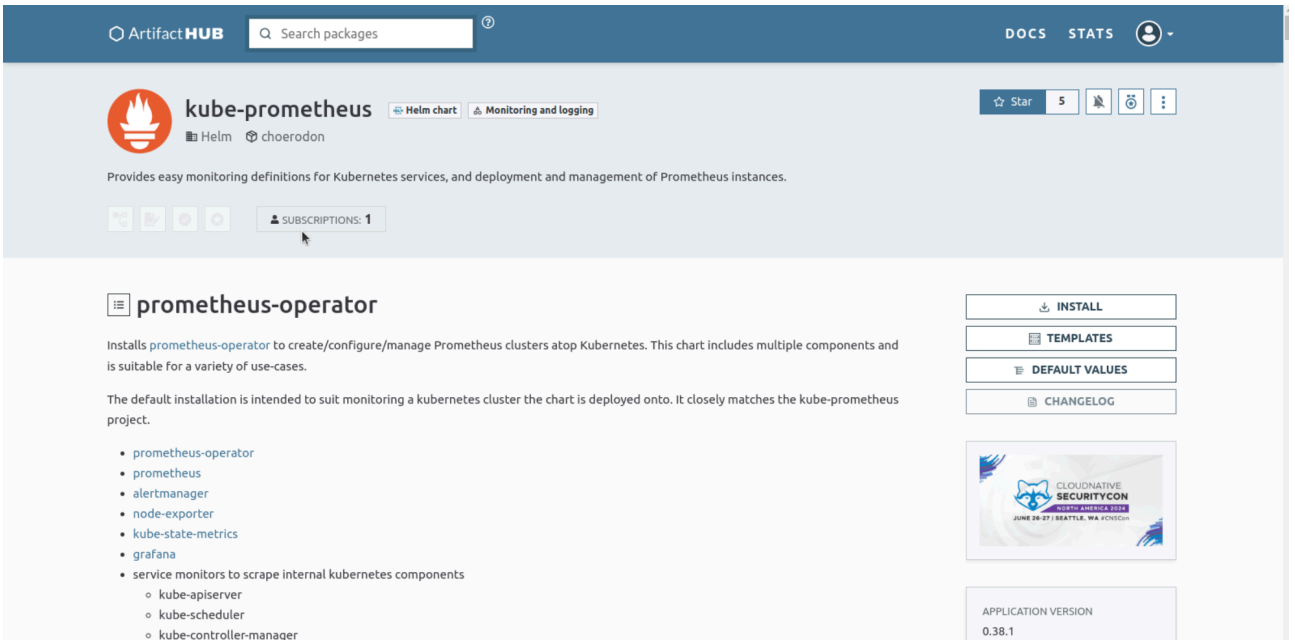


Рисунок 3.12 – Kube-prometheus-stack

Kube-prometheus-stack – це Helm chart, який надає можливість швидкого та простого розгортання системи моніторингу Prometheus та візуалізації даних за допомогою Grafana в середовищі Kubernetes. Цей Helm chart містить в собі повний стек компонентів, необхідних для належного функціонування системи моніторингу:

- Prometheus Operator: Цей компонент керує різними складовими Prometheus, включаючи моніторингові правила, служби та власні ресурси.
- Prometheus Server: Сервер Prometheus, який збирає, зберігає та обробляє дані моніторингу.
- Grafana: Веб-інтерфейс для візуалізації та аналізу даних моніторингу за допомогою готових або власних панелей.
- Alertmanager: Компонент для управління та оповіщення про події, що створюються Prometheus.

- Exporter: Різноманітні експортери для збору метрик з різних додатків та систем.

Крім того, kube-prometheus-stack надає можливість налаштування різних параметрів, що дозволяє гнучко адаптувати конфігурацію моніторингу під власні потреби. Використання kube-prometheus-stack спрощує процес розгортання системи моніторингу в Kubernetes, забезпечуючи швидке та надійне впровадження моніторингового середовища для додатку.

Код розгортання kube-prometheus-stack зображено на рисунку 3.13.

```
const prometheusNamespace = new k8s.core.v1.Namespace("prometheus-namespace", {
    metadata: {
        name: "monitoring",
    },
});

const prometheusChart = new k8s.helm.v3.Chart("my-kube-prometheus-stack", {
    namespace: prometheusNamespace.metadata.name,
    chart: "kube-prometheus-stack",
    version: "58.4.0",
    fetchOpts: {
        repo: "https://prometheus-community.github.io/helm-charts",
    },
    values: {
        grafana: {
            service: {
                type: "LoadBalancer",
            },
        }
    },
});
```

Рисунок 3.13 – Розгортання chart

3.5 Аналіз результатів розгортання на кластері Kubernetes

Після успішного розгортання інфраструктури за допомогою коду на кластері AKS перевіримо, що всі необхідні ресурси Kubernetes створились. Для цього використаємо інструмент k9s, інтерактивний терміналовий клієнт для керування Kubernetes. Він надає зручний спосіб взаємодії з кластерами

Kubernetes через текстовий інтерфейс. K9s дозволяє оперативно переглядати, моніторити та керувати ресурсами кластера, такими як події, поділений доступ, стан ресурсів тощо, все це зручно в терміналі.

На рисунку 3.14 зображено вивід k9s, що показує усі запущені поди на кластері.

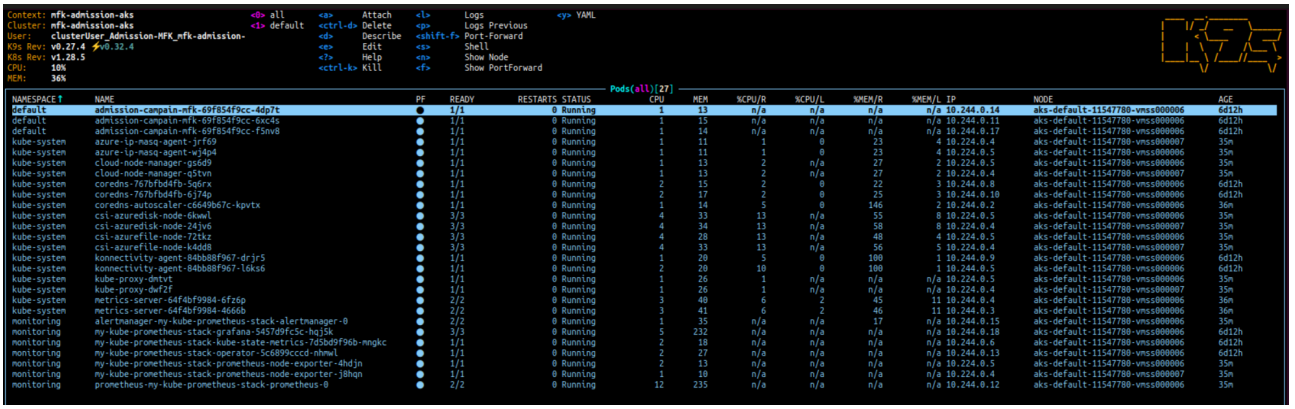
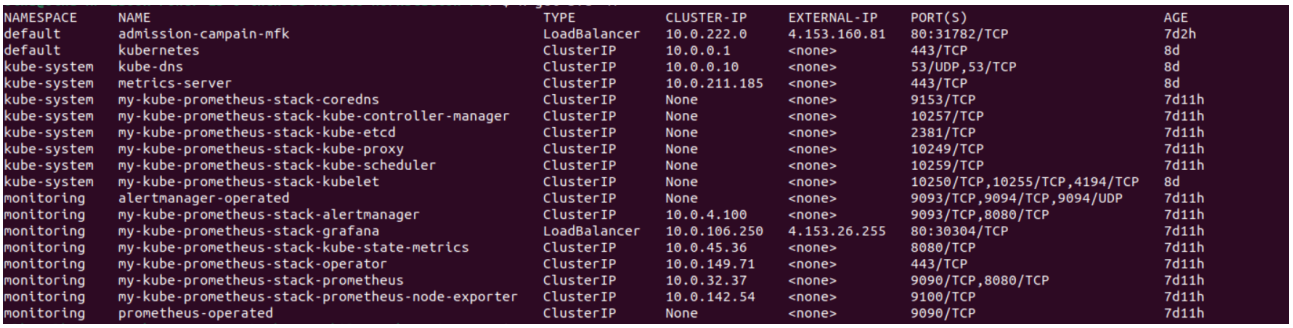


Рисунок 3.14 – K9s вивід

Як бачимо з рисунку 3.14 всі поди перебувають у стані Running, це свідчить про стабільну роботу кластера, де всі конейнеризовані додатки працюють належним чином. Це важливий показник для забезпечення надійності та продуктивності додатків, що розгортаються в середовищі Kubernetes.

Для перевірки успішного розгортання додатків в Kubernetes перевіримо наявність розгорнутих сервісів та переконаймося, що вони потрібні сервіси типу Load Balancer мають необхідні зовнішні IP-адреси. Це важливий крок для забезпечення доступності додатків ззовні та їх коректної роботи в середовищі кластера Kubernetes.

На рисунку 3.15 зображено вивід команди отримання всіх сервісів в Kubernetes.



NAMESPACE	NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
default	admission-campain-mfk	LoadBalancer	10.0.222.0	4.153.160.81	80:31782/TCP	7d2h
default	kubernetes	ClusterIP	10.0.0.1	<none>	443/TCP	8d
kube-system	kube-dns	ClusterIP	10.0.0.10	<none>	53/UDP,53/TCP	8d
kube-system	metrics-server	ClusterIP	10.0.211.185	<none>	443/TCP	8d
kube-system	my-kube-prometheus-stack-coredns	ClusterIP	None	<none>	9153/TCP	7d11h
kube-system	my-kube-prometheus-stack-kube-controller-manager	ClusterIP	None	<none>	10257/TCP	7d11h
kube-system	my-kube-prometheus-stack-kube-etcd	ClusterIP	None	<none>	2381/TCP	7d11h
kube-system	my-kube-prometheus-stack-kube-proxy	ClusterIP	None	<none>	10249/TCP	7d11h
kube-system	my-kube-prometheus-stack-kube-scheduler	ClusterIP	None	<none>	10259/TCP	7d11h
kube-system	my-kube-prometheus-stack-kubelet	ClusterIP	None	<none>	10250/TCP,10255/TCP,4194/TCP	8d
monitoring	alertmanager-operated	ClusterIP	None	<none>	9093/TCP,9094/TCP,9094/UDP	7d11h
monitoring	my-kube-prometheus-stack-alertmanager	ClusterIP	10.0.4.100	<none>	9093/TCP,8080/TCP	7d11h
monitoring	my-kube-prometheus-stack-grafana	LoadBalancer	10.0.106.250	4.153.26.255	80:30304/TCP	7d11h
monitoring	my-kube-prometheus-stack-kube-state-metrics	ClusterIP	10.0.45.36	<none>	8080/TCP	7d11h
monitoring	my-kube-prometheus-stack-operator	ClusterIP	10.0.149.71	<none>	443/TCP	7d11h
monitoring	my-kube-prometheus-stack-prometheus	ClusterIP	10.0.32.37	<none>	9090/TCP,8080/TCP	7d11h
monitoring	my-kube-prometheus-stack-prometheus-node-exporter	ClusterIP	10.0.142.54	<none>	9100/TCP	7d11h
monitoring	prometheus-operated	ClusterIP	None	<none>	9090/TCP	7d11h

Рисунок 3.15 – Вивід всіх сервісів в Kubernetes-кластері

Як бачимо на рисунку 3.15 всі Load Balancer отримали зовнішні Ip.

Після успішного розгортання інфраструктури за допомогою TS коду на кластері AKS, Pulumi вивів зовнішні URL, які дають можливість звертатися до інформаційної системи. Ці URL є ключовим елементом для доступу до розгорнутої системи ззовні (рис.3.16).

```
Outputs:
    externalIp     : "4.153.160.81/Main/html/index.php"
    externalIpAdmin: "4.153.160.81/Admin"
```

Рисунок 3.16 – Pulumi output

Перевірка відповідності цих URL системі дозволяє впевнитися, що вона доступна та працює так, як очікується. Це важливий етап після розгортання, оскільки підтверджує функціональність і готовність системи до використання. Важливо переконатися, що URL вказують на правильні ресурси та що вони доступні для взаємодії з користувачами. Цей процес допомагає уникнути потенційних проблем та перевірити коректність роботи системи в реальному середовищі.

На рисунку 3.17 зображено, що після звернення за відповідною URL інформаційна система відповідає.

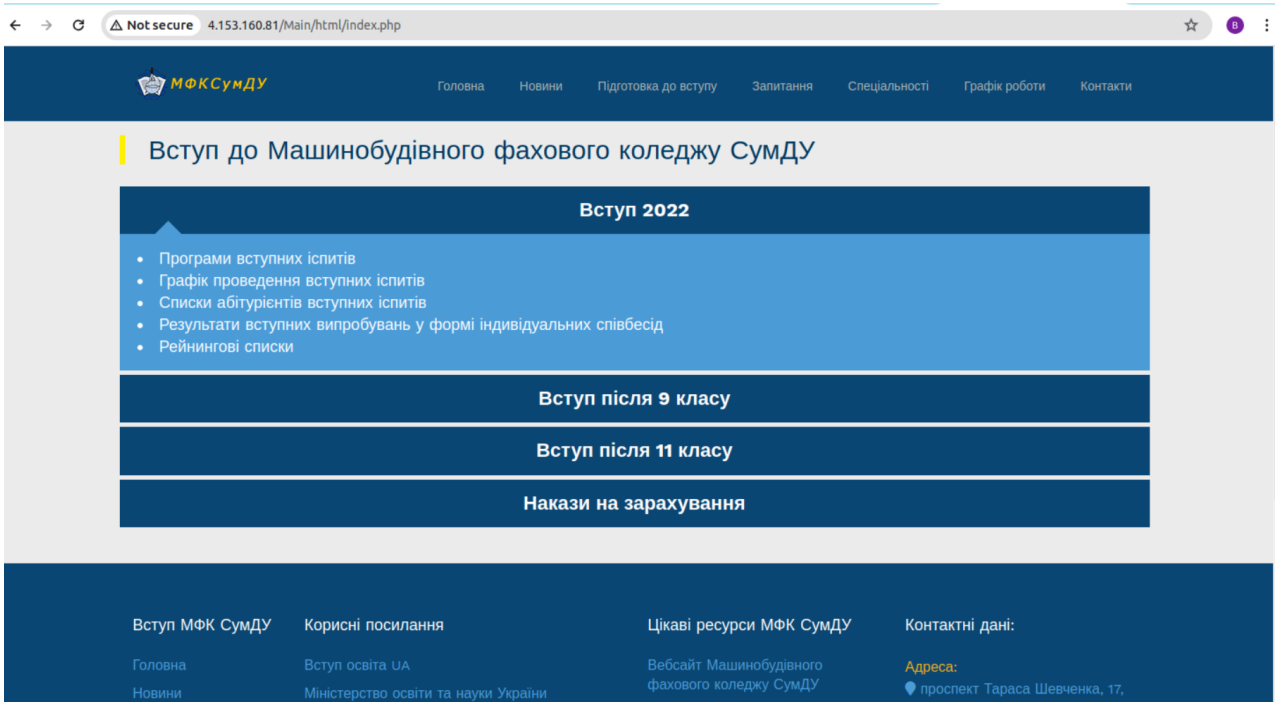


Рисунок 3.17 – Головна сторінка інформаційної системи

Після звернення за URL адмін частини відкривається сторінка авторизації (рис.3.18).

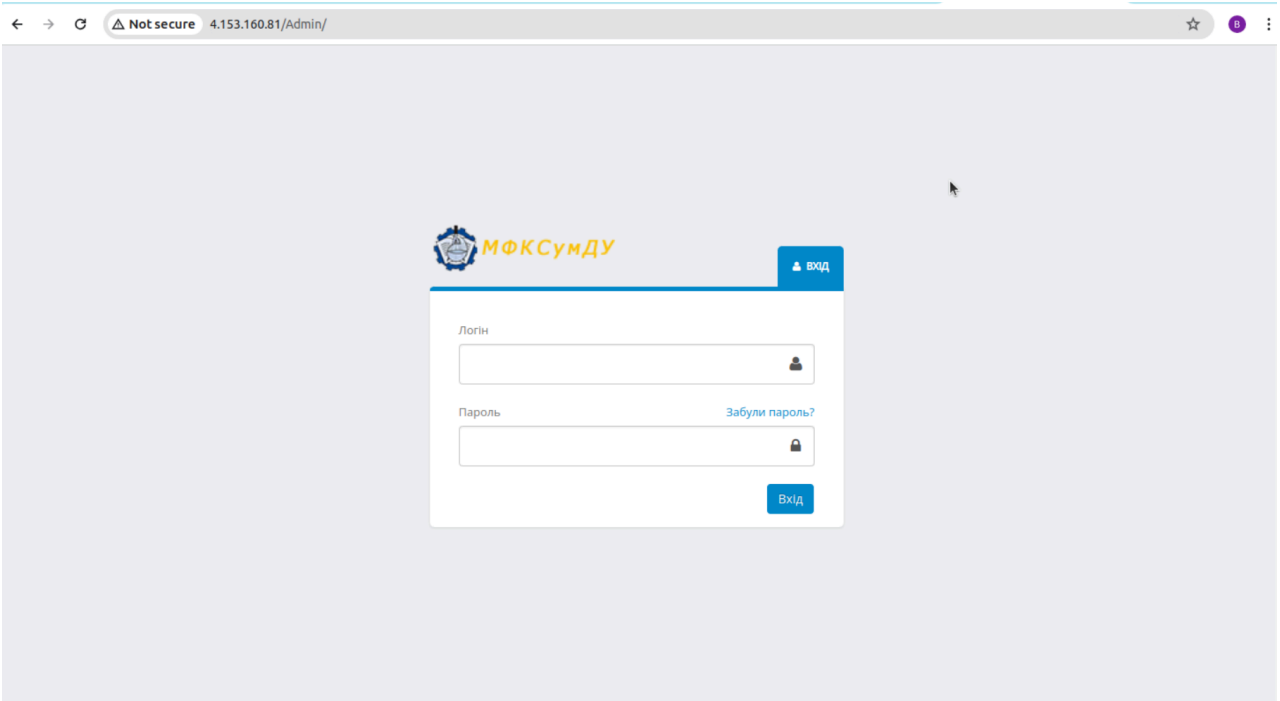


Рисунок 3.18 – Авторизація в адмін частину

Після авторизації в адмін частину відкривається головна сторінка інформаційної системи, зображена на рисунку рисунку 3.19.

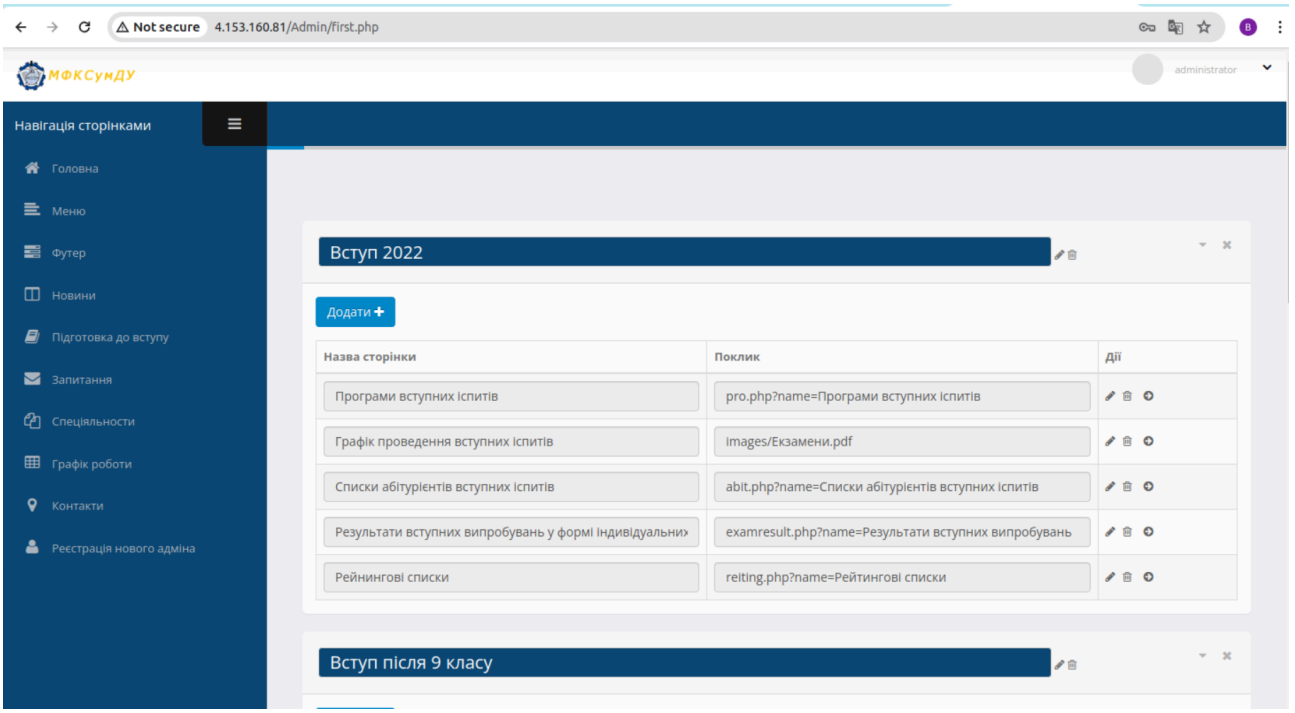


Рисунок 3.19 – Головна сторінка адмін частини інформаційної системи

Для перевірки успішного розгортання моніторингової системи звернемось до зовнішньої ip Grafana. На рисунку 3.20 зображено сторінку авторизації в Grafana.

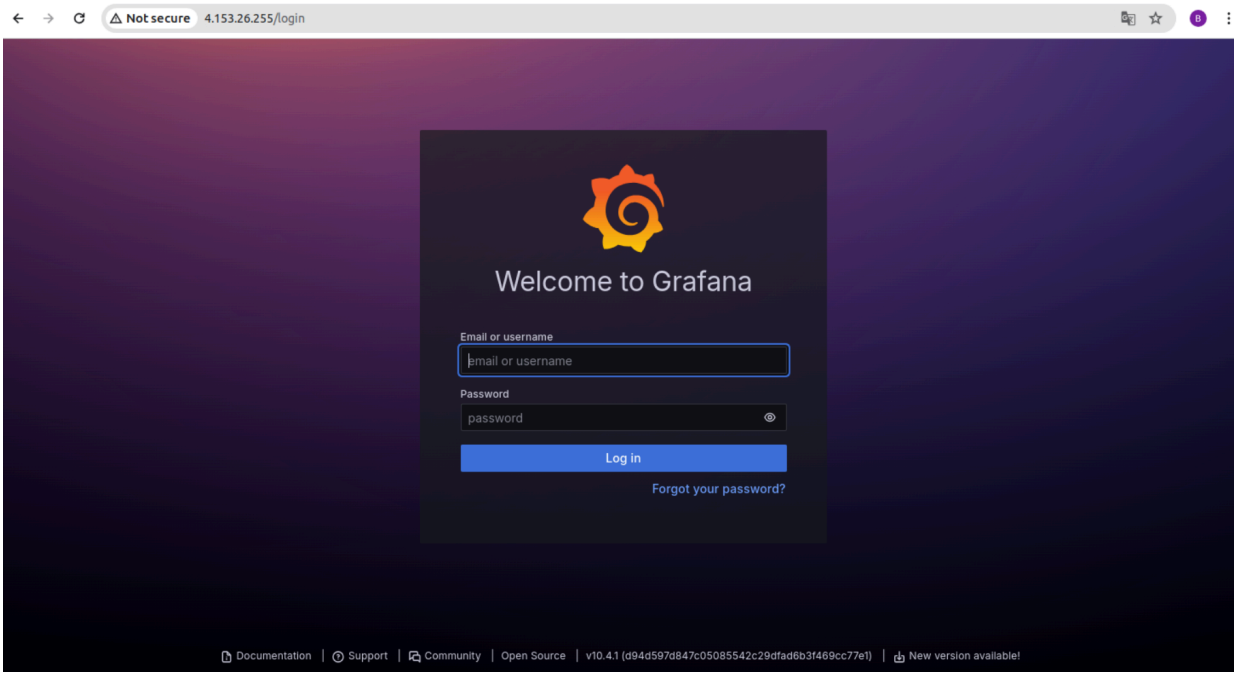


Рисунок 3.20 – Авторизація в Grafana

Після авторизації в Grafana відкривається головна сторінка моніторингового сервісу з меню (рис. 3.21).

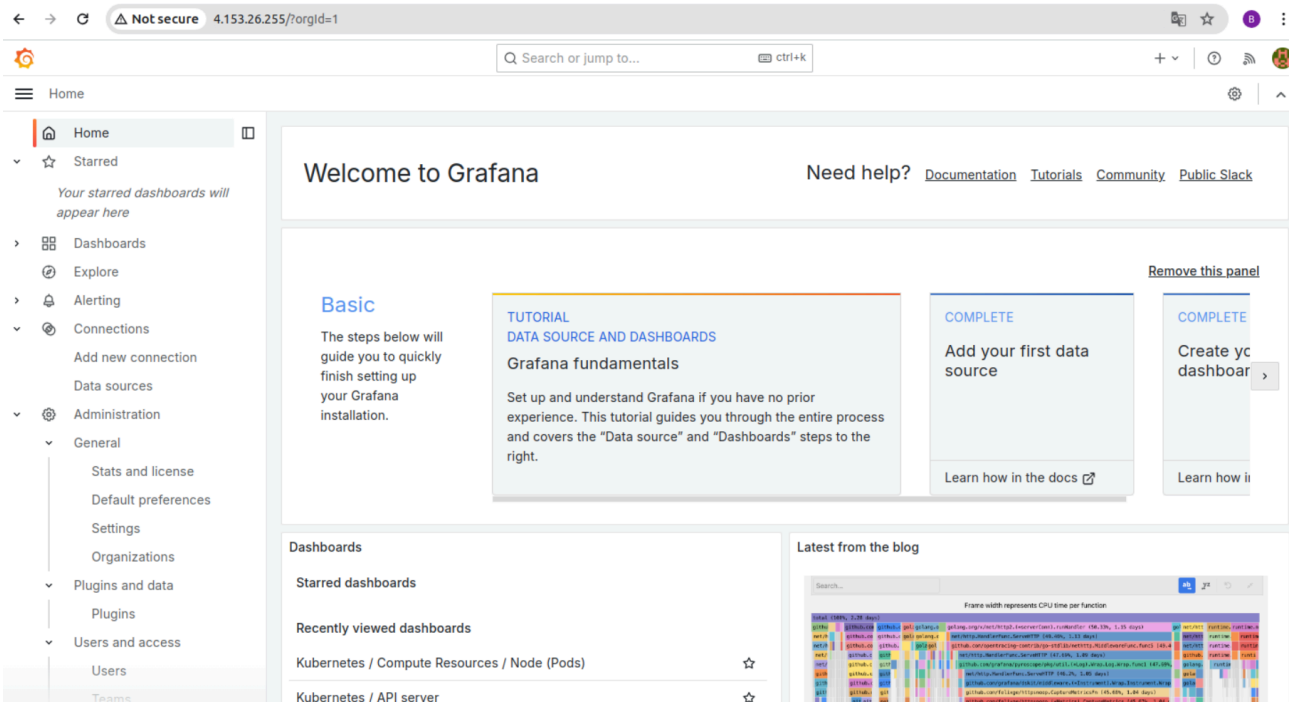


Рисунок 3.21 – Головна сторінка Grafana

На рисунку 3.22 зображено джерела даних для Grafana, які вона використовує для візуалізації даних.

Дрежера включають:

– Alertmanager: джерело даних, яке Grafana може використовувати для отримання сповіщень від Alertmanager. Використовується для створення панелей та сповіщень на основі метрик, які надсилає Alertmanager.

– Prometheus: джерело даних, яке Grafana використовує для отримання метрик від Prometheus. Відображення метрик з Prometheus у вигляді графіків та діаграм – один з основних функціональних можливостей Grafana.

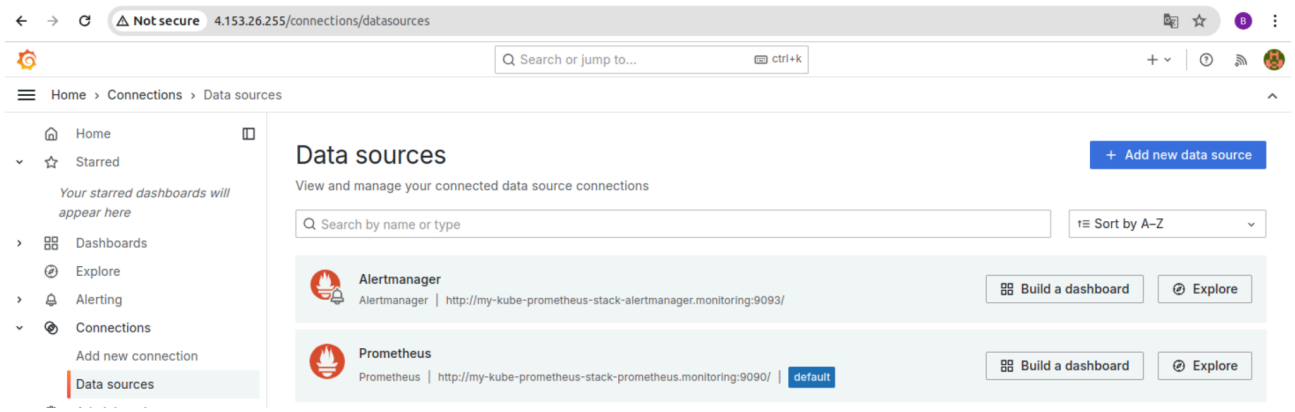


Рисунок 3.22 – Джерела даних Grafana

На рисунку 3.23 зображено сторінку Dashboards в Grafana, центр управління візуалізацією даних.

Тут маємо можливість створювати, редагувати та переглядати графіки, панелі та звіти на основі даних, які отримуються з різних джерел, таких як бази даних, сервіси моніторингу тощо.

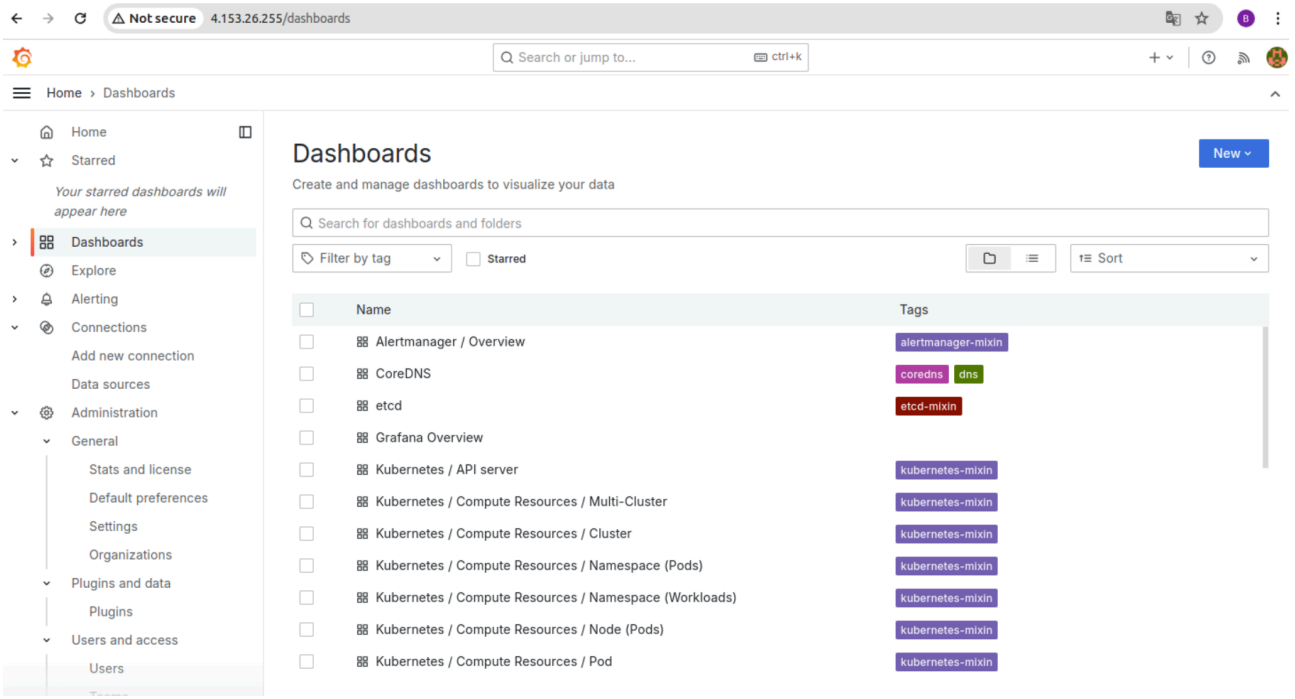


Рисунок 3.23 – Сторінка Dashboards Grafana

На рисунку 3.24 зображено Dashboard Node Exporter / Nodes" в Grafana, що призначений для моніторингу різних параметрів системи в реальному часі. Цей дашборд надає зручні панелі для візуалізації даних про центральний процесор (CPU), пам'ять, диск та мережу.

У розділі CPU є дві панелі, які відображають використання процесора. Перша панель може показувати загальне використання CPU на всіх вузлах системи, тоді як друга може деталізувати це використання для кожного окремого вузла.

Панелі для пам'яті також поділені на дві частини. Одна з них може відображати загальний обсяг використаної оперативної пам'яті, в той час як інша може показати розподіл використання пам'яті за різними типами (наприклад, кеш, буфери, активна пам'ять тощо) для кожного вузла.

Панелі для диску відображають стан обсягу даних на кожному вузлі, включаючи загальний обсяг дискового простору, використаний обсяг та доступний залишок. Можливо також показати швидкість введення/виведення (I/O) на диску.

У розділі мережі панелі можуть відображати загальний обсяг передачі даних по мережі, а також швидкість передачі даних (завантаження) для кожного вузла окремо.

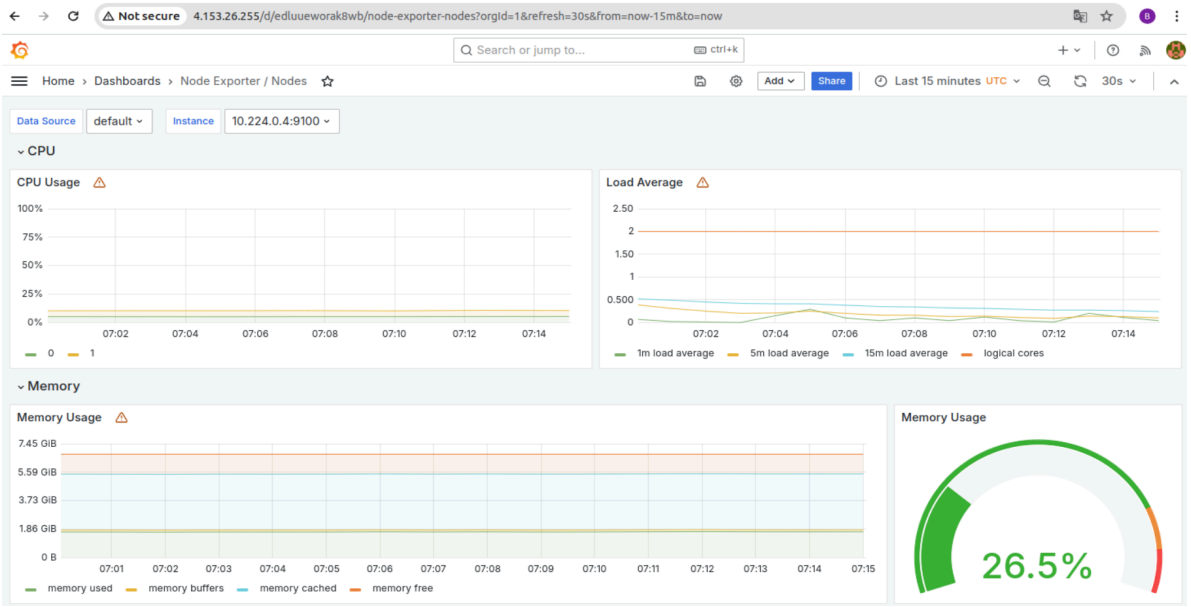


Рисунок 3.24 – Dashboards "Node Exporter / Nodes"

За допомогою цього дашборда можна швидко моніторити стан різних ресурсів системи, виявляти проблеми та вчасно реагувати на них.

Дашборд "Kubernetes / Compute Resources / Namespace (Pods)" (рис.3.25) у Grafana призначений для моніторингу використання ресурсів обчислювальних вузлів Kubernetes у вказаному просторі імен (namespace). Цей дашборд дозволяє відстежувати рівень використання CPU, пам'яті, мережі та вводу/виводу (I/O) для кожного POD-у в обраному просторі імен.

Ключові компоненти цього дашборду включають:

- Namespace: Дозволяє вибрати конкретний простір імен Kubernetes для аналізу.
- Headlines: Надають короткий огляд статистики для обраного простору імен, може містити, наприклад, кількість та стан POD-ів.

- CPU Usage / CPU Quota / Memory Usage / Memory Quota (1 панель кожна): Відображають використання та квоти CPU та пам'яті для POD-ів у обраному просторі імен.
- Current Network Usage: Показує поточне використання мережі.
- Bandwidth / Rate of Packets / Rate of Packets Dropped: Надають інформацію про пропускну здатність мережі та швидкість передачі пакетів та їх відкидання.
- Storage IO / Storage IO – Distribution: Відображають дані щодо вводу/виводу на сховище та його розподіл.

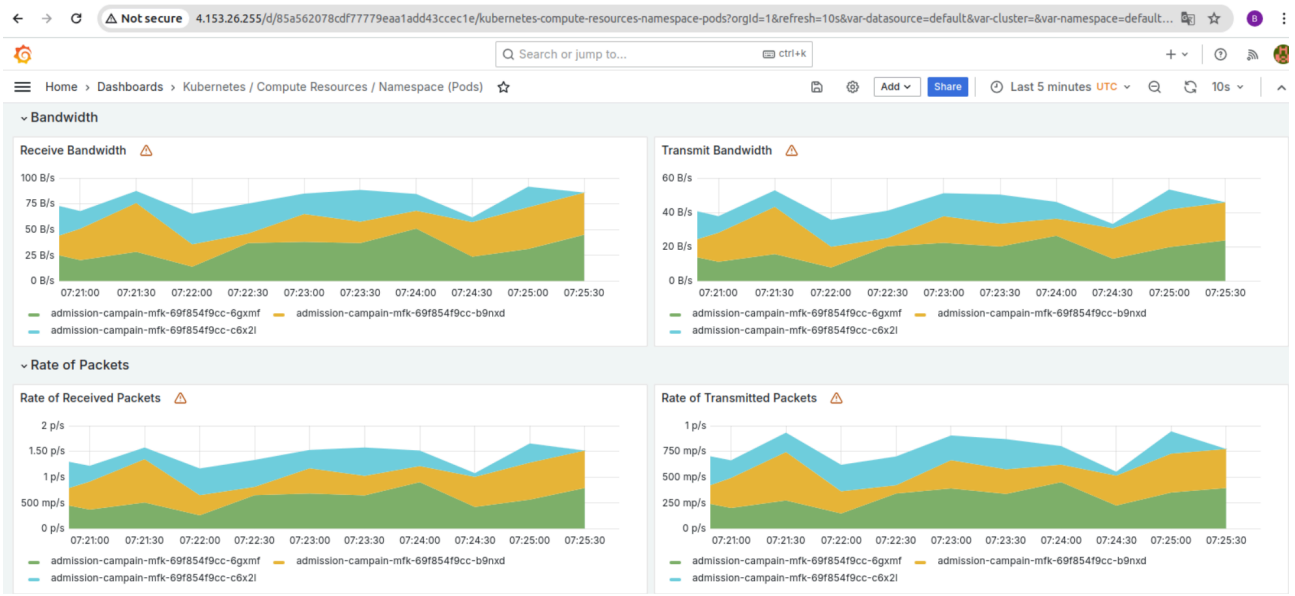


Рисунок 3.25 – Dashboards "Kubernetes / Compute Resources / Namespace (Pods)"

Цей дашборд допомагає отримувати повну картину використання ресурсів у вказаному просторі імен, що дозволяє вчасно виявляти проблеми та ефективно керувати ресурсами кластера.

Крім згаданих дашбордів у Grafana, існує ще багато інших дашбордів, які надають різноманітну інформацію про різні аспекти роботи системи. Ці

дашборди можуть містити панелі з метриками щодо використання ресурсів, продуктивності, навантаження, стану сервісів, мережевого трафіку, безпеки та багато іншого. Вони призначені для візуалізації даних та аналізу параметрів системи у реальному часі або в зазначений період часу.

Інформативні панелі на цих дашбордах можуть надавати корисну статистику, включаючи графіки, діаграми, числові показники та інші візуальні елементи, що допомагають краще розуміти стан та продуктивність їх інфраструктури.

Узагальнюючи, використання дашбордів у системах моніторингу дозволяє забезпечити ефективне керування та нагляд за інфраструктурою, вчасно виявляти проблеми та приймати відповідні заходи для забезпечення найвищої доступності, продуктивності та безпеки системи.

Після успішного розгортання інфраструктури та додатків на кластері Kubernetes за допомогою коду TS та Pulumi було проведено ретельний аналіз для підтвердження коректної роботи системи.

Ключові моменти:

- Статус подів: Усі поди, що розгорнуті на кластері, перебувають у стані "Running", що свідчить про стабільну роботу всіх конейнеризованих додатків.
- Розгортання сервісів: Всі необхідні сервіси, включаючи Load Balancer, успішно розгорнуті та мають зовнішні IP-адреси, що забезпечує доступність додатків ззовні.
- Доступність інформаційної системи: Інформаційна система доступна за зовнішніми URL, отриманими від Pulumi.

- Функціональність адмін частини: Авторизація та доступ до головної сторінки адмін частини інформаційної системи підтверджують коректну роботу бекенд частини.

Моніторинг:

- Grafana успішно розгорнута та доступна за зовнішньою IP-адресою.
- Авторизація та доступ до головної сторінки Grafana підтверджують коректну роботу моніторингової системи.
- Наявні дашборди Grafana дозволяють моніторити різні аспекти роботи системи, включаючи використання ресурсів, продуктивність, навантаження, стан сервісів, мережевий трафік та багато іншого.

Отже, Результати аналізу підтверджують успішне розгортання інфраструктури та додатків на кластері Kubernetes. Система працює стабільно, всі необхідні сервіси та моніторинг доступні, що гарантує надійність, продуктивність та безпеку інформаційної системи.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи бакалавра було проведено аналіз сучасних підходів до управління контейнеризованими додатками в Kubernetes-кластері. Визначено важливість контейнеризації в сучасному інформаційному середовищі та розглянуто методи управління та оркестрації таких додатків.

У ході виконання кваліфікаційної роботи бакалавра було виконано наступні завдання:

1. Дослідження сучасних підходів та інструментів контейнеризації, оркестрації, моніторингу та автоматизованого розгортання.
2. Розроблено та налаштовано скрипти інфраструктури як коду для автоматизації процесів розгортання та конфігурації інфраструктури, що дозволяє забезпечити швидке відтворення середовищ у будь-який момент часу.
3. Проведено інтеграцію Docker для розгортання контейнеризованих додатків.
4. Налаштовано оркестратор Kubernetes для управління контейнеризованими додатків.
5. Налаштовано та інтегровано систему моніторингу та телеметрії з використанням Grafana та Prometheus для постійного моніторингу стану та використання ресурсів контейнеризованих середовищ.
6. Проведено аналіз результатів роботи для перевірки розгортання інфраструктури, її продуктивності та ефективності в умовах навантаження.

Отже, можна зробити висновок, що розроблена система автоматичного запуску та управління контейнеризованими додатками в Kubernetes-кластері є ефективним та перспективним рішенням для сучасних інформаційних середовищ.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What to choose? Virtualization or Containerization? [Електроний ресурс] - URL: https://dev.to/yogini16/what-to-choose-virtualization-or-containerization-1mjm (дата звернення: 09.04.2024).

2. Container Orchestration? [Електроний ресурс] - URL: https://www.wallarm.com/what/what-is-container-orchestration-7-benefits-and-4-best-tools (дата звернення: 12.04.2024).

3. What is Infrastructure as Code? [Електроний ресурс] - URL: https://learn.microsoft.com/en-us/devops/deliver/what-is-infrastructure-as-code (дата звернення: 15.04.2024).

4. Data Engineering with Infrastructure as Code. [Електроний ресурс] - URL: https://www.linkedin.com/posts/ankitrathi_data-engineering-with-infrastructure-as-code-activity-7155447053467148288-noDQ/ (дата звернення: 24.04.2024).

5. What is infrastructure monitoring? [Електроний ресурс] - URL: https://www.ibm.com/topics/infrastructure-monitoring (дата звернення: 18.04.2024).

6. What is Cloud Computing? Types of Cloud-based Technologies and Services. [Електроний ресурс] - URL: https://blog.servercore.com/what-is-cloud-computing-types-of-cloud-based-technologies-and-services-9e0d0a49ae3b (дата звернення: 21.04.2024).

7. Docker. [Електроний ресурс] - URL: https://www.docker.com/ (дата звернення: 27.04.2024).

8. What is Podman? [Електроний ресурс] - URL: https://podman.io/ (дата звернення: 30.04.2024).

9. Docker Hub. [Електроний ресурс] - URL: https://hub.docker.com/ (дата звернення: 03.05.2024).

10. Docker Workflow. [Електроний ресурс] - URL: https://medium.com/@augustineozor/docker-workflow-b9fe71d32184 (дата звернення: 05.05.2024).

11. Kubernetes. [Електроний ресурс] - URL: https://kubernetes.io/docs/concepts/overview/ (дата звернення: 09.04.2024).

12. OpenShift. [Електроний ресурс] - URL: https://www.redhat.com/en/technologies/cloud-computing/openshift (дата звернення: 12.04.2024).

13. Rancher. [Електроний ресурс] - URL: https://rancher.com/docs/rancher/latest/zh/ (дата звернення: 15.04.2024).

14. Amazon Web Services. [Електроний ресурс] - URL: https://aws.amazon.com/ (дата звернення: 18.04.2024).

15. Azure. [Електроний ресурс] - URL: https://azure.microsoft.com/ (дата звернення: 21.04.2024).

16. Google Cloud. [Електроний ресурс] - URL: https://cloud.google.com/ (дата звернення: 24.04.2024).

17. Architecture overview Kubernetes [Електроний ресурс] - URL: https://medium.com/@augustineozor/docker-workflow-b9fe71d32184 (дата звернення: 27.04.2024).

18. What is Jenkins? Architecture, Installation, and Setup Explained. [Електроний ресурс] - URL: https://www.lambdatest.com/blog/what-is-jenkins/ (дата звернення: 30.04.2024).

19. What is GitLab and How to Use It? [Електроний ресурс] - URL: https://www.simplilearn.com/tutorials/git-tutorial/what-is-gitlab (дата звернення: 03.05.2024).

20. What Is GitHub and Why Should You Use It? [Електроний ресурс] - URL: https://www.coursera.org/articles/what-is-git (дата звернення: 05.05.2024).

21. What is IBM Cloud? Services Offered, Features & Pricing. [Електроний ресурс] - URL: https://www.datamation.com/cloud/ibm-cloud/ (дата звернення: 09.04.2024).

22. Getting started with Red Hat CodeReady Containers. [Електроний ресурс] - URL: https://access.redhat.com/documentation/en-us/red_hat_codeready_containers/1.0/html/getting_started_guide/getting-started-with-codeready-containers_gsg (дата звернення: 12.04.2024).

23. Operator pattern. [Електроний ресурс] - URL: https://kubernetes.io/docs/concepts/extend-kubernetes/operator/ (дата звернення: 15.04.2024).

24. Kubernetes Service Mesh Definition. [Електроний ресурс] - URL: https://avinetworks.com/glossary/kubernetes-service-mesh/ (дата звернення: 18.04.2024).

25. Serverless on Kubernetes. [Електроний ресурс] - URL: https://medium.com/appvia/serverless-on-kubernetes-63b49aeaf4ef (дата звернення: 21.04.2024).

26. Kubernetes extension. [Електроний ресурс] - URL: https://quarkus.io/guides/deploying-to-kubernetes (дата звернення: 24.04.2024).

27. CNCF. [Електроний ресурс] - URL: https://www.cncf.io/ (дата звернення: 27.04.2024).

28. Terraform. [Електроний ресурс] - URL: https://www.terraform.io/ (дата звернення: 30.04.2024).

29. HashiCorp. [Електроний ресурс] - URL: https://developer.hashicorp.com/terraform/docs (дата звернення: 03.05.2024).

30. Creating a Highly Available Architecture with Terraform. [Електроний ресурс] - URL: https://www.hashicorp.com/ (дата звернення: 05.05.2024).

31. Mozilla Public License. [Електроний ресурс] - URL: https://www.mozilla.org/en-US/MPL/ (дата звернення: 09.04.2024).

32. BSL stands for Business Source License. [Електроний ресурс] - URL: https://fossa.com/blog/business-source-license-requirements-provisions-history/ (дата звернення: 12.04.2024).

33. Pulumi. [Електроний ресурс] - URL: https://byjus.com/chemistry/hydrochloric-acid/ (дата звернення: 15.04.2024).

34. Python. [Електроний ресурс] - URL: https://www.python.org/ (дата звернення: 18.04.2024).

35. JavaScript. [Електроний ресурс] - URL: https://uk.javascript.info/ (дата звернення: 21.04.2024).

36. TypeScript. [Електроний ресурс] - URL: https://www.typescripttutorial.net/ (дата звернення: 24.04.2024).

37. Go. [Електроний ресурс] - URL: https://go.dev/ (дата звернення: 27.04.2024).

38. C#. [Електроний ресурс] - URL: https://learn.microsoft.com/uk-ua/dotnet/csharp/ (дата звернення: 30.04.2024).

39. Java. [Електроний ресурс] - URL: https://www.java.com/en/ (дата звернення: 03.05.2024).

40. Kubernetes Pulumi. [Електроний ресурс] - URL: https://www.pulumi.com/registry/packages/kubernetes/ (дата звернення: 09.04.2024).

41. Monitoring Tools in Kubernetes: An Overview of Available Options and Integration. [Електроний ресурс] - URL: https://medium.com/@prateek.malhotra004/monitoring-tools-in-kubernetes-an-overview-of-available-options-and-integration-812c4c64b8b6 (дата звернення: 09.04.2024).

42. 11 Top Cloud Service Providers Globally In 2023. [Електроний ресурс] - URL: https://www.cloudzero.com/blog/cloud-service-providers (дата звернення: 12.04.2024).

Додаток А Код Dockerfile

Dockerfile

Dockerfile > ...

```
1   FROM debian:stable
2
3   RUN apt-get update && apt-get install -y \
4       apache2 \
5       php \
6       libapache2-mod-php \
7       php-mysql \
8       php-cli \
9       php-pdo
10
11  RUN a2enmod rewrite
12  RUN service apache2 restart
13
14  WORKDIR /var/www/html/
15  COPY . .
16
17  EXPOSE 80
18
19  CMD ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
20
```

Додаток Б Код Pulumi

index.ts Pulumi.yaml

index.ts > admission_campain_mfkDeployment

```ts
import * as k8s from "@pulumi/kubernetes";

const labels = { app: "admission-campain-mfk" };

const admission_campain_mfkDeployment = new k8s.apps.v1.Deployment("admission_campain_mfkDeployment", {
    apiVersion: "apps/v1",
    kind: "Deployment",
    metadata: { name: labels.app },
    spec: {
        replicas: 3,
        selector: { matchLabels: labels },
        template: {
            metadata: { labels },
            spec: {
                containers: [{
                    name: labels.app,
                    image: "helga09/admission-campain-mfk:0.0.1",
                    ports: [{ containerPort: 80 }],
                    env: ["DB_HOST", "DB_USER", "DB_PASSWORD", "DB_DATABASE"].map(name => ({
                        name,
                        valueFrom: { secretKeyRef: { name: "db-secret", key: name } },
                    })),
                }],
            },
        },
    },
});

const defaultAdmission_campain_mfkService = new k8s.core.v1.Service("defaultAdmission_campain_mfkService", {
    apiVersion: "v1",
    kind: "Service",
    metadata: { name: labels.app, namespace: "default" },
    spec: {
        allocateLoadBalancerNodePorts: true,
        ipFamilies: ["IPv4"],
        ipFamilyPolicy: "SingleStack",
        ports: [{ port: 80, protocol: "TCP", targetPort: 80 }],
        selector: labels,
        sessionAffinity: "None",
        type: "LoadBalancer",
```