НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут телекомунікаційних систем

Кафедра телекомунікацій

До захисту допущено:

Завідувач кафедри

_____ Сергій КРАВЧУК

«___»_____2023 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія та програмування інфокомунікацій»

спеціальності 172 Телекомунікації тарадіотехніка

на тему: « “Cloud native” сервіси та технології в мережах операторів мобільного зв’язку»

Виконав:
студент IV курсу, групи ТЗ-92
Ярошінський Дмитро Олександрович _____

Керівник:
Доцент кафедри ТК НН ІТС, к.т.н., с.н.с.,
Міночкін Дмитро Анатолійович _____

Рецензент:
Доцент кафедри ІКТС НН ІТС, к.т.н., доцент,
Новогрудська Ріна Леонідівна _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2023 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут телекомунікаційних систем
Кафедра телекомунікацій

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 172 Телекомунікації тарадіотехніка

Освітньо-професійна програма «Інженерія та програмування інфокомунікацій»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій КРАВЧУК
«___»_____2023 р.

ЗАВДАННЯ
на дипломну роботу студенту
Ярошінському Дмитру Олександровичу

1.Тема роботи «“Cloud native” сервіси та технології в мережах операторів мобільного зв’язку», керівник роботи Міночкін Дмитро Анатолійович, к.т.н., с.н.с., затверджені наказом по університету від «22» травня 2023 р. № 1884-с.

2. Термін подання студентом роботи 13 червня 2023 року.

3. Вихідні дані до роботи: інформаційні матеріали про технологію Cloud Native в мережах операторів мобільного зв’язку. Структурований план порядку розробки матеріалів дипломної роботи.

4. Зміст роботи : Здійснити огляд архітектури та принципів дії Cloud Native. Проаналізувати технології Cloud Native в мережах операторів мобільного зв’язку. Розглянути наявні використання технології Cloud Native мобільними операторами. Проаналізувати ключові переваги Cloud Native. Провести порівняльний аналіз існуючих технологій з Cloud Native. Розробити свою власну архітектуру на основі гібридної хмари.

5. Перелік ілюстративного матеріалу: слайди презентації за матеріалами проведеного дослідження технології Cloud Native в мережах операторів мобільного зв’язку.

6. Дата видачі завдання _____20 лютого 2023 р

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Розробка, оформлення, узгодження та затвердження технічного завдання на роботу. Аналітичний огляд інформаційних матеріалів. Підбір та опрацювання необхідної науковотехнічної літератури.	20.02.2023-07.03.2023	Виконано
2	Вивчення та розбір основних відомостей про хмарні технології: основні складові Cloud Native.	08.03.2023-24.03.2023	Виконано
3	Провести порівняльний аналіз існуючих технологій з Cloud Native. Розглянути наявні використання технології Cloud Native мобільними операторами.Розробити свою власну архітектуру на основі Cloud Native.	25.03.2023-23.04.2023	Виконано
4	Підготовка матеріалів до друку та оформлення пояснювальної записки	28.04.2023-20.05.2023	Виконано
5	Підготовка та оформлення презентації для доповіді	25.05.2023-09.06.2023	Виконано

Студент _____ Дмитро ЯРОШІНСЬКИЙ
(підпис)

Керівник _____ Дмитро МІНОЧКІН
(підпис)

РЕФЕРАТ

Дипломна робота містить 69 сторінок, 9 рисунків. Було використано 28 джерел інформації.

Актуальність роботи. «Cloud native» сервіси та технології в мережах операторів мобільного зв’язку – це новий спосіб розробки та розгортання мережевих функцій, які є більш гнучкими, масштабованими, стійкими та ефективними, ніж традиційні методи.

Cloud native використовує архітектуру мікросервісів, контейнеризацію, оркестрацію та автоматизацію для створення мережевих функцій, які можуть працювати на будь-якій хмарній платформі та адаптуватися до всіх вимог і середовищ.

Ця технологія може спростити архітектуру мережі та операції, зменшуючи комплексність та залежності від власного ПО та серверів. Вона дозволить мобільним операторам швидко розвиватись на ринку інновацій шляхом автоматизованого випуску оновлень мережевих функцій, покращить досвід використання послугами для користувачів, надаючи більш сучасні, безпечні та легкі для користування додатки та сервіси.

Очікується, що хмарні сервіси і технології матимуть значний вплив на мобільні мережі 5G, забезпечуючи швидші інновації, нижчі витрати, кращу продуктивність і нові бізнес-можливості для постачальників послуг зв'язку (CSP).

Мета підходу Cloud Native в мережах мобільних операторів мобільного зв’язку полягає в закладанні легкої можливості розширення потужностей та покращення додатків та сервісу в реальному часі завдяки використанню багатьох технологій Cloud Native.

Об’єкт дослідження – “Cloud native” сервіси та технології в мережах операторів мобільного зв’язку

Предмет дослідження – Cloud Native технології в мережах зв’язку

Методи дослідження. В роботі використовуються підходи критичного та порівняльного аналізу, протягом дослідження теорії та обладнання систем Cloud Native технологій.

Отримані результати. Результати отримані внаслідок досліджень використання Cloud Native підходу в телекомунікаціях дали змогу зрозуміти, що це покращить наявні мережі мобільних операторів , зробить легше розробку нових продуктів та покращить досвід використання продуктів користувачами.

Галузь застосування. Результати роботи можна використовувати під час створення мережі зв’язку з використанням підходу Cloud Native. Результати можуть допомогти зрозуміти які ключові принципи можуть бути використані при створенні та зрозуміти майбутній потенціал.

Ключові слова. Технології Cloud Native , архітектура мобільної мережі, телеком-хмара.

ABSTRACT

The thesis contains 69 pages and 9 pictures. 25 sources of information were used.

Relevance of work. "Cloud native" services and technologies in mobile operator networks are a new way to develop and deploy network functions that are more flexible, scalable, resilient and efficient than traditional methods.

Cloud native uses microservices architecture, containerization, orchestration, and automation to create network functions that can run on any cloud platform and adapt to all requirements and environments.

This technology can simplify network architecture and operations by reducing complexity and dependencies on proprietary software and servers. It will enable mobile operators to innovate rapidly by automating the release of network feature updates, and improve the user experience by providing more modern, secure and easy-to-use applications and services.

Cloud services and technologies are expected to have a significant impact on 5G mobile networks, providing faster innovation, lower costs, better performance and new business opportunities for communication service providers (CSPs).

The goal of the Cloud Native approach in mobile MNO networks is to enable easy capacity expansion and real-time application and service enhancements by leveraging multiple Cloud Native technologies.

Object of research - "Cloud native" services and technologies in the networks of mobile operators

Subject of research - Cloud Native technologies in communication networks

Research methods. The paper uses the approaches of critical and comparative analysis in the study of the theory and development of Cloud Native technologies.

Results obtained. The results of the research on the use of Cloud Native approach in telecommunications made it possible to understand that it will improve existing networks of mobile operators, make it easier to develop new products and improve the user experience.

Field of application. The results of the work can be used when creating a communication network using the Cloud Native approach. The results can help to understand what key principles can be used in the creation and understand the future potential.

Keywords. Cloud Native technologies, mobile network architecture, telco cloud.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

CI/CD	Continuous Integration and Continuous Delivery
CNCF	Cloud Native Computing Foundation
VM	Virtual Machine
CNF	Cloud-Native Network Function
NFV	Network Functions Virtualization Management And Orchestration
KVM	Kernel-Based Virtual Machine
NFVI	Network Functions Virtualization Infrastructure
API	Application Programming Interface

ВСТУП

«Cloud native» сервіси та технології в мережах операторів мобільного зв’язку – це новий спосіб розробки та розгортання мережевих функцій, які є більш гнучкими, масштабованими, стійкими та ефективними, ніж традиційні методи. Cloud native використовує архітектуру мікросервісів, контейнеризацію, оркестрацію та автоматизацію для створення мережевих функцій, які можуть працювати на будь-якій хмарній платформі та адаптуватися до всіх вимог і середовищ. Очікується, що хмарні сервіси і технології матимуть значний вплив на мобільні мережі 5G, забезпечуючи швидші інновації, нижчі витрати, кращу продуктивність і нові бізнес-можливості для постачальників послуг зв'язку (CSP).

Переваги "Cloud Native " послуг і технологій для мобільних операторів полягають у наступному:

- Скорочення капітальних інвестицій: за рахунок використання обладнання загального призначення і спільних ресурсів замість спеціального обладнання для різних цілей.
- Скорочення операційних витрат за рахунок автоматизації розгортання, масштабування та управління мережевими функціями. Сервіс-орієнтована архітектура дозволяє інтегруватися з Cloud Native адаптованою хмарою, щоб впоратися з високими навантаженнями під час пікових годин, пропонувати нові послуги з меншими витратами на розробку, а також використовувати переваги інших сторонніх сервісів, таких як аналітика, машинне навчання та штучний інтелект.
- Додаткові платні послуги: використання хмарних платформ для створення нових послуг і потоків доходів від їх використання
- Адаптація бізнесу: швидке впровадження нових додатків і запуск нових послуг

- Підвищення продуктивності: динамічне масштабування мережевих функцій для зростання мережі відповідно до попиту
- Підвищення стійкості: за рахунок використання вбудованих механізмів виявлення пошкоджених мікросервісів і можливостей самовідновлення.
- Швидкість обслуговування: Спільними службами можуть користуватися всі мережеві функції, розгорнуті в хмарному середовищі (CNE). Використання загальних служб забезпечує всі мережеві функції надають телеметрію в одній і тій же структурі, що в подальшому спрощує кореляцію та усунення несправностей.

РОЗДІЛ 1
ТЕХНОЛОГІЯ CLOUD NATIVE

1.1 Визначення поняття Cloud Native

Cloud Native - це підхід до створення та запуску додатків, які використовують переваги хмарних обчислень. Хмарні обчислення - це надання обчислювальних ресурсів через Інтернет, таких як сервери, сховища, бази даних, мережі, програмне забезпечення та аналітика . Cloud Native додатки створені для дослідження можливістей легкого розширення в майбутньому, покращення відмовостійкісті в стресових ситуаціях та покращення швидкості їх роботи[1].

Додатки Cloud Native складаються з декількох невеликих взаємозалежних сервісів, які називаються мікросервісами, що працюють незалежно і потребують мінімальних обчислювальних ресурсів для запуску. Мікросервіси часто упаковані в контейнери - легкі та портативні пристрої, які ізолюють код програми від базової інфраструктури [2]. Контейнери можна розгортати та оновлювати швидко і часто та в реальному часі, не впливаючи на надання послуг.

Також, додатки Cloud Native покладаються на інструменти та практики автоматизації, такі як безперервна інтеграція та безперервна доставка (CI/CD), щоб забезпечити швидкі та часті зміни в програмному забезпеченні без ручного втручання. Конвеєри CI/CD автоматизують процеси створення, тестування та розгортання програмного забезпечення в різних середовищах. Автоматизація також допомагає забезпечити якість, безпеку та надійність програмного забезпечення [3].

Додатки Cloud Native можна використовувати на будь-якій хмарній платформі, наприклад, публічній, приватній або комбінованій хмарі. Публічні хмари належать і управляються сторонніми провайдерами, які пропонують обчислювальні ресурси за рівнем потреби користувача. Приватні хмари належать і управляються однією організацією, яка має ексклюзивний доступ до

обчислювальних ресурсів. Гібридні хмари - це поєднання публічних і приватних хмар, які дозволяють переміщати дані та додатки між ними[4].

Хмарні технології мають різні переваги для бізнесу. Вони допомагають підвищити ефективність, знизити витрати, забезпечити доступність та сприяти інноваціям. Застосовуючи хмарний підхід, компанії можуть створювати високомасштабовані, гнучкі та відмовостійкі додатки, які вони можуть швидко оновлювати відповідно до вимог клієнтів.

1.2 Історія виникнення Cloud Native

Концепція архітектури Cloud Native зробила революцію в тому, як компанії розробляють, розгортають та керують своїми додатками та продуктами. Цей інноваційний підхід дозволив організаціям створювати масштабовані, стійкі та легкі до пристосування системи, які можуть процвітати в динамічному цифровому просторі. Щоб оцінити значення архітектури Cloud Native, важливо розуміти її історію та походження.

Поява хмарних обчислень

Історія архітектури Cloud Native починається з появою хмарних обчислень на початку 2000-х років. Коли інтернет став більш доступним і дешевим, компанії почали вивчати потенціал використання віддалених серверів для зберігання, управління та обробки даних. Цей зсув призвів до розвитку хмарних сервісів, які дозволили компаніям отримувати доступ до обчислювальних ресурсів за рівнем їх потреби та більш ефективно розвивати свої продукти[5].

Поява мікросервісів

По мірі того, як хмарні обчислення набирали обертів, розробники почали усвідомлювати обмеження традиційних додатків, створених цілісно і як єдиний продукт. Ці великі, складні системи було важко підтримувати, масштабувати та оновлювати, що призводило до повільних циклів розробки та зниження гнучкості. У відповідь на це з'явилася концепція мікросервісів, яка передбачала

розбиття додатків на менші незалежні компоненти, які можна було розробляти, розгортати незалежно один від одного.

Ідея мікросервісів не нова, але з часом вона еволюціонувала з попередніх шаблонів та технологій розробки програмного забезпечення. Одним з перших впливів була концепція віддаленого виклику процедур (RPC), яка з'явилася у 1980-х роках як спосіб зробити розподілені обчислення прозорими для розробників. RPC дозволяв розробникам викликати процедури або методи на різних машинах, не турбуючись про деталі мережі. Однак, RPC мав недоліки, такі як проблеми з продуктивністю через мережеві накладні витрати та чат-інтерфейси[6].

Розвиток хмарних обчислень і веб-технологій у 2000-х і 2010-х роках створив нові можливості і вимоги до розробки програмного забезпечення. Хмарні обчислення запропонували масштабовані, еластичні та доступні згідно потреби обчислювальні ресурси через Інтернет, а веб-технології уможливили багаті користувацькі інтерфейси та динамічну доставку контенту. Розробникам потрібно було створювати додатки, які могли б використовувати переваги хмарних обчислень і відповідати очікуванням користувачів щодо швидкості, доступності та інновацій. Це призвело до появи мікросервісів як архітектурного стилю, що відповідав хмарному середовищу.

Мікросервіси запропонували кілька переваг над монолітними архітектурами, включаючи покращену масштабованість, відмовостійкість та гнучкість. Такий підхід дозволив розробникам швидше проходити ітерації розробки, експериментувати з новими функціями та ефективніше реагувати на складні ринкові умови. В результаті мікросервіси стали ключовим компонентом руху в напрямку Cloud Native.

Поява контейнерів та контейнерної оркестрації

Впровадження мікросервісів призвело до необхідності більш ефективного способу пакування, розгортання та управління цими окремими компонентами. Ця потреба породила контейнеризацію - легку технологію віртуалізації, яка

дозволяє розробникам упаковувати додатки та їхні залежності в ізольовані, портативні одиниці, що називаються контейнерами.

Наприклад, Docker, випущений у 2013 році, популяризував контейнеризацію і швидко став галузевим стандартом для створення та управління контейнерами. Однак, оскільки кількість контейнерів зростала, організаціям потрібен був спосіб організувати їх та керувати ними в масштабі. Це призвело до розробки платформ оркестрування контейнерів, таких як Kubernetes, яка була випущена компанією Google у 2014 році. Kubernetes забезпечив потужну та гнучку основу для управління контейнерними додатками, що ще більше зміцнило важливість використання контейнерів в архітектурі Cloud Native.

Фонд Cloud Native Computing Foundation (CNCF)

Cloud Native Computing Foundation (CNCF) відіграє ключову роль у розвитку та стандартизації нативних хмарних технологій з моменту свого заснування. Як спільний проект під егідою Linux Foundation, CNCF сприяє інноваціям та співпраці між лідерами галузі, сприяючи впровадженню архітектури Cloud Native. Ця стаття розповідає про історію створення CNCF та його вплив на екосистему Cloud Native[9].

Створення CNCF

CNCF була заснована у 2015 році як дочірня компанія Linux Foundation. Його місія полягає в тому, щоб зробити Cloud Native обчислення повсюдними, надаючи організаціям можливість створювати і запускати масштабовані додатки в сучасних динамічних середовищах. Фонд прагне створити спільноту розробників, кінцевих користувачів, постачальників IT-технологій та послуг для спільної роботи над проектами з відкритим вихідним кодом та сприяння впровадженню технологій Cloud Native[7].

Одним з перших і найбільш значущих проектів CNCF було впровадження Kubernetes, платформи оркестрування контейнерів з відкритим вихідним кодом, спочатку розробленої компанією Google. Kubernetes швидко став наріжним каменем екосистеми Cloud Native, надаючи потужну та гнучку основу для управління контейнерними додатками.

Окрім Kubernetes, CNCF розмістив і підтримував численні інші проекти, зокрема Prometheus, інструментарій для моніторингу та оповіщення; Envoy, високопродуктивний проксі-сервер; і containerd, середовище виконання контейнерів. Ці проекти сприяли розвитку надійного і різноманітного ландшафту Cloud Native[10].

Щоб допомогти організаціям орієнтуватися в складній екосистемі Cloud Native, CNCF створив CNCF Landscape, всеосяжну інтерактивну карту різних технологій, інструментів і платформ, які складають простір Cloud Native. Ландшафт слугує цінним ресурсом для підприємств, які прагнуть впровадити технології Cloud Native та зрозуміти взаємозв'язок між різними проектами та рішеннями.

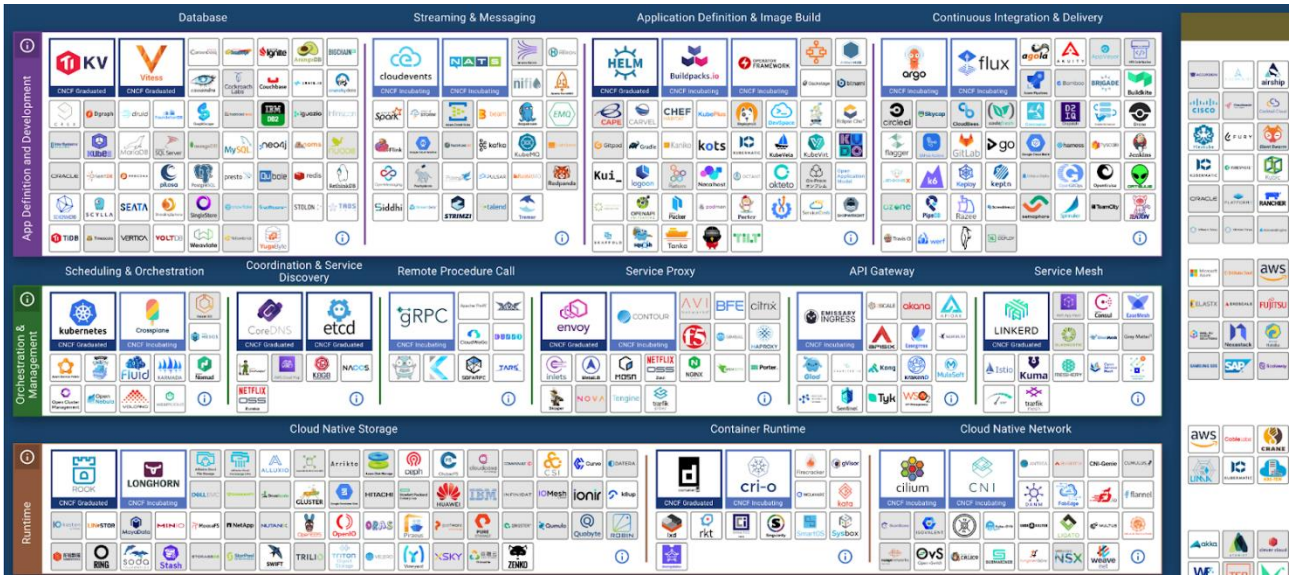

Рис.1.1 Ландшафт CNCF додатків

CNCF також розробив централізовану інформаційну панель безперервної інтеграції (CI) для спрощення тестування та управління кількома проектами на хмарних і "голих" платформах. Ця інформаційна панель, відома як CNCF Cloud Native Interactive Landscape, стала важливим інструментом для розробників та організацій, що працюють з технологіями Cloud Native.

CNCF відіграє важливу роль у формуванні сильної та активної спільноти Cloud Native. Завдяки таким заходам, як KubeCon + CloudNativeCon, фонд об'єднує розробників, кінцевих користувачів та лідерів галузі для обміну знаннями, обговорення ідей та співпраці над проектами. Ці заходи відіграють вирішальну роль у стимулюванні інновацій та сприянні впровадженню технологій Cloud Native у різних галузях.

Історія Cloud Native Computing Foundation відзначена її прагненням сприяти співпраці, інноваціям та стандартизації в екосистемі Cloud Native. Завдяки підтримці проектів з відкритим вихідним кодом, розробці цінних ресурсів та організації громадських заходів, CNCF відіграє ключову роль у формуванні майбутнього Cloud Native обчислень. Фонд продовжує рости і розвиватися, але залишається відданим своїй місії - зробити Cloud Native обчислення розповсюдженими і доступними для організацій по всьому світу.

1.2.1 Важливість та актуальність Cloud Native у сучасному IT-ландшафті

Важливість та актуальність Cloud Native у сучасному світі неможливо переоцінити. Впроваджуючи технології Cloud Native, організації пришвидшують розвиток своїх продуктів та покращують досвід користувачів з кожним днем, одночасно сприяючи розвитку культури співпраці та інновацій. Оскільки бізнес продовжує долати виклики цифрової епохи, прийняття принципів і практик Cloud Native буде мати вирішальне значення для збереження конкурентоспроможності та забезпечення довгострокового успіху.

У сучасному стрімкому цифровому світі компанії повинні швидко адаптуватися, щоб залишатися конкурентоспроможними та задовольняти потреби своїх клієнтів, які постійно змінюються[11]. Технології Cloud Native стали критично важливим компонентом сучасного IT-ландшафту, що дозволяє організаціям створювати та запускати додатки в динамічних середовищах.

Перечислимо ключові переваги технології Cloud Native:

Спритність і швидкість

Однією з головних переваг технологій Cloud Native є підвищена гнучкість та швидкість, яку вони пропонують. Використовуючи мікросервіси, контейнери та платформи оркестрування контейнерів, такі як Kubernetes, організації можуть швидше працювати на результат свого продукту. Це дозволяє компаніям швидко реагувати на зміни ринку, експериментувати з новими функціями та надавати їх своїм клієнтам швидше.

Масштабованість та відмовостійкість

Додатки Cloud Native розроблені таким чином, щоб бути легкими в розширенні та відмовостійкими за своєю суттю. Розбиваючи додатки на менші незалежні компоненти, організації можуть масштабувати окремі сервіси за потреби, забезпечуючи оптимальну продуктивність навіть при коливаннях попиту. Крім того, технології Cloud Native підвищують відмовостійкість і здатність до самовідновлення, мінімізуючи вплив збоїв і забезпечуючи безперервне надання послуг[12].

Економічна ефективність

Впровадження технологій Cloud Native може призвести до значного скорочення витрат для бізнесу. Використовуючи динамічні хмарні моделі, що надаються потужності за рівнем потреби, організації можуть досягти економічної ефективності та краще розподіляти ресурси відповідно до своїх потреб. Це дозволяє компаніям оптимізувати витрати на інфраструктуру та

уникнути надмірного або недостатнього використання ресурсів,коштів, сил працівників.

Співпраця та інновації

Технології Cloud Native сприяють співпраці та інноваціям у командах розробників. Розбиваючи додатки на менші, більш керовані компоненти, розробники можуть працювати над окремими сервісами незалежно, зменшуючи ризик конфліктів і сприяючи більш ефективному процесу розробки. Таке середовище співпраці заохочує експерименти та інновації, дозволяючи компаніям залишатися попереду на конкурентному ринку та користувачам отримувати кращі продукти через конкуренцію.

Цифрова трансформація та безперервність бізнесу

Після пандемії COVID-19 потреба в цифровій трансформації та безперервності бізнесу стала більш очевидною, ніж будь-коли. Технології Cloud Native відіграють вирішальну роль у наданні організаціям можливості адаптуватися до швидкозмінного середовища та підтримувати бізнес-операції в умовах непередбачуваних викликів. Використовуючи Cloud Native, компанії можуть забезпечити свій довгостроковий успіх і стійкість у світі, що стає все більш цифровим[12].

1.3 Роль контейнерів в архітектурі Cloud Native

Контейнери - це пакети програмного забезпечення, які містять усі необхідні елементи для роботи в будь-якому середовищі. Вони віртуалізують операційну систему і працюють будь-де, від приватного дата-центру до публічної хмари або навіть на особистому ноутбуці розробника[13].

Контейнери відіграють ключову роль у хмарних додатках, тобто додатках, які спроектовані, розроблені та розгорнуті для використання всіх переваг моделі хмарних обчислень. Вони використовують хмарні технології та сервіси для

створення масштабованих, надійних і безпечних додатків, які можуть працювати на будь-якій хмарній платформі[14].

Деякі з переваг використання контейнерів для хмарних додатків:

Мобільність*:* Контейнери можуть працювати на будь-якій платформі, що їх підтримує, наприклад, на операційних системах Linux, Windows або Mac; на віртуальних машинах або фізичних серверах; на машині розробника або в локальних центрах обробки даних; і, звичайно, в хмарі. Це дозволяє легко переміщати додатки між різними середовищами без зміни коду або конфігурації.

Ефективність: Контейнери легкі та використовують менше ресурсів, ніж віртуальні машини. Вони можуть працювати під управлінням спільної операційної системи та користуватися спільними бібліотеками і залежностями. Це зменшує накладні витрати та підвищує продуктивність додатків. Це також забезпечує більшу щільність та використання ресурсів, що знижує вартість хостингу та експлуатації додатків

Спритність: Контейнери пришвидшують розробку та розгортання додатків. Вони дозволяють розробникам зосередитись на логіці та залежностях додатків, тоді як IT-операційні команди можуть зосередитись на розгортанні та управлінні. Контейнери також підтримують CI/CD. Це дозволяє швидше надавати нові функції та виправлення. Однак використання контейнерів для хмарних додатків також пов'язане з певними проблемами, а саме:

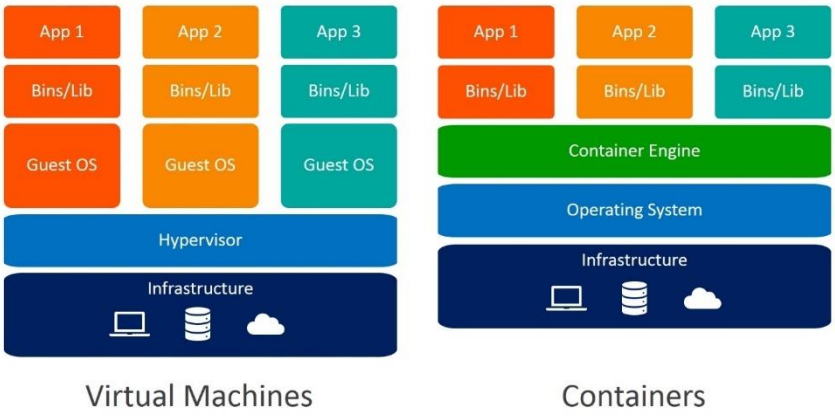

Рис.1.2. Порівняльна ілюстрація інфраструктури з контейнерами та VM

Складність: Контейнери збільшують складність додатків, оскільки вони передбачають розбиття програми на менші компоненти, які взаємодіють через стандартні протоколи та API. Кожен компонент повинен бути спроектований і розроблений незалежно і скоординований з іншими компонентами. Це вимагає іншого мислення і набору навичок від розробників і архітекторів, які повинні прийняти принципи і практики архітектури мікросервісів

Безпека: Контейнери створюють ризики для безпеки, оскільки вони працюють на спільній операційній системі, яка може мати вразливості або неправильну конфігурацію. Крім того, контейнери можуть містити конфіденційні дані або облікові дані, які необхідно захистити від несанкціонованого доступу або витоку[15]. Це вимагає від розробників та IT-операційних команд дотримуватися найкращих практик безпеки, таких як використання шифрування, автентифікації, авторизації, аудиту та безпечного кодування.

Управління: Контейнери потребують інструментів та процесів управління для організації їх розгортання, масштабування, моніторингу, реєстрації, усунення несправностей та оновлення. Ці інструменти та процеси повинні бути сумісними з хмарним середовищем, в якому працюють контейнери,

і підтримувати динамічну природу контейнерів. Це вимагає від IT-операційних команд впровадження нових інструментів і процесів, призначених для контейнерних додатків.

Для вирішення цих завдань розробники та IT-операційні команди можуть використовувати різні хмарні сервіси та платформи, які підтримують контейнерні додатки, такі як Kubernetes, Docker Swarm, Azure Container Apps, Google Cloud Run, Amazon Elastic Container Service (ECS) тощо. Ці сервіси та платформи надають такі функції, як оркестрування контейнерів, виявлення сервісів, балансування навантаження, перевірка працездатності, ведення журналів, моніторинг, масштабування, безпека тощо. Вони також інтегруються з іншими хмарними сервісами та платформами для підвищення функціональності та продуктивності контейнерних додатків.

1.3.1 Управління контейнерами – Docker

Що таке Docker?

Docker - це програмна платформа, яка дозволяє створювати, ділитися та запускати програми за допомогою контейнерів. Контейнери - це пакети програмного забезпечення, які містять всі необхідні елементи для запуску в будь-якому середовищі. Вони віртуалізують операційну систему і працюють будь-де, від приватного дата-центру до публічної хмари або навіть на особистому ноутбуці розробника[17].

Docker Inc. була заснована Камелем Фунаді, Соломоном Хайксом та Себастьяном Палем під час участі в групі стартап-інкубатора Y Combinator Summer 2010 і запущена в 2011 році. Стартап також був одним з 12 стартапів першої когорти Founder's Den. Хайкс розпочав проект Docker у Франції як внутрішній проект компанії dotCloud, що надає платформу як послугу.

Рис.1.3 Соломон Хайкс та Себастьян Пал в групі стартап-інкубатора Y Combinator

Docker дебютував для громадськості в Санта-Кларі на конференції PyCon у 2013 році. Він був випущений з відкритим вихідним кодом у березні 2013 року. На той час він використовував LXC як середовище виконання за замовчуванням. Через рік, з виходом версії 0.9, Docker замінив LXC на власний компонент libcontainer, написаний на мові програмування Go[17]. У 2017 році Докер створив проект Moby для відкритих досліджень і розробок.

Docker - це набір продуктів платформи як послуги (PaaS), які використовують віртуалізацію на рівні операційної системи для доставки програмного забезпечення в пакетах, що називаються контейнерами. Сервіс має як безкоштовний, так і преміум-рівень. Програмне забезпечення, на якому розміщуються контейнери, називається Docker Engine.Вперше воно було запущено в 2013 році і розробляється компанією Docker, Inc.

Контейнери ізольовані один від одного і містять власне програмне забезпечення, бібліотеки та конфігураційні файли; вони можуть спілкуватися один з одним через чітко визначені канали. Оскільки всі контейнери користуються послугами одного ядра операційної системи, вони використовують менше ресурсів, ніж віртуальні машини.

Docker може пакувати програму та її залежності у віртуальний контейнер, який можна запустити на будь-якому комп'ютері з Linux, Windows або macOS. Це дозволяє програмі працювати у різних місцях, наприклад, локально, у публічній або приватній хмарі. Під час роботи у Linux Docker використовує функції ізоляції ресурсів ядра Linux (наприклад, cgroups та простори імен ядра) та файлову систему, що підтримує об'єднання (наприклад, OverlayFS), щоб дозволити контейнерам працювати в межах одного екземпляра Linux, уникаючи накладних витрат на запуск та підтримку віртуальних машин. Docker на macOS використовує віртуальну машину Linux для запуску контейнерів[16].

Оскільки контейнери Docker легкі за розміром, на одному сервері або віртуальній машині можна запускати кілька контейнерів одночасно. Аналіз 2018 року показав, що типовий випадок використання Docker передбачає запуск восьми контейнерів на хост, а чверть проаналізованих організацій запускають 18 або більше контейнерів на хост. Він також може бути встановлений на одноплатному комп'ютері, наприклад Raspberry Pi.

Підтримка ядром Linux просторів імен здебільшого ізолює уявлення програми про операційне середовище, включаючи дерева процесів, мережу, ідентифікатори користувачів та змонтовані файлові системи, тоді як cgroups ядра забезпечують обмеження ресурсів пам'яті та процесора. Починаючи з версії 0.9, Docker включає власний компонент (який називається "libcontainer") для використання засобів віртуалізації, що надаються безпосередньо ядром Linux, на додаток до використання абстрактних інтерфейсів віртуалізації через libvirt, LXC і systemd-nspawn.

Docker пропонує багато переваг для розробників, а саме:

Переносимість: Контейнери Docker можуть працювати на будь-якій платформі, яка їх підтримує, наприклад, на операційних системах Linux, Windows або Mac; на віртуальних машинах або фізичних серверах; на машині розробника або в локальних дата-центрах; і, звичайно, в публічній хмарі. Це

дозволяє легко переміщати додатки між різними середовищами без зміни коду або конфігурації.

Ефективність: Контейнери Docker легкі та використовують менше ресурсів, ніж віртуальні машини. Вони можуть працювати на спільній операційній системі та мати спільні бібліотеки та залежності. Це зменшує накладні витрати і підвищує продуктивність додатків. Це також забезпечує більшу щільність та використання ресурсів, що знижує вартість хостингу та експлуатації додатків

Спритність: Контейнери Docker прискорюють розробку та розгортання додатків. Вони дозволяють розробникам зосередитися на логіці та залежностях додатків, тоді як IT-операційні команди можуть зосередитися на розгортанні та управлінні. Контейнери також підтримують безперервні конвеєри доставки, які автоматизують тестування, розгортання та оновлення додатків. Це дозволяє швидше надавати нові функції та виправлення.

Docker надає інструменти та сервіси, які допоможуть вам створювати, розгортати та керувати контейнерними програмами. Ось деякі з основних компонентів Docker:

Docker Engine*:* Головний компонент Docker, який дозволяє створювати, запускати та керувати контейнерами. Він складається з сервера (*dockerd*), який взаємодіє з клієнтом (docker) за допомогою API або команд CLI. Docker може працювати на різних платформах, таких як Linux, Mac, Windows та Cloud провайдери. Ви можете встановити докер-рушій різними способами, залежно від вашої операційної системи та вподобань. Docker також підтримує різні канали випуску, такі як стабільний та тестовий, які мають різні рівні стабільності та можливостей. Docker дозволяє створювати та використовувати об'єкти докерів, такі як зображення, контейнери, мережі та томи, які складають ваші програми. Ви також можете використовувати docker для оркестрування ваших додатків за

допомогою таких сервісів, як Kubernetes, AWS ECS, Azure ACI та Swarm mode. Движок докерів - це проект з відкритим вихідним кодом, ліцензований за ліцензією Apache License 2.0

Docker Desktop*:* Нативний додаток для Mac, Windows або Linux, який надає всі інструменти Docker на ваш комп'ютер.

Docker Desktop - це програма, яка допоможе вам створювати, ділитися та запускати контейнеризовані додатки на вашому комп'ютері Mac, Linux або Windows. Це найпростіший спосіб розпочати роботу з докером і використовувати його для розробки та тестування. Docker Desktop містить все необхідне для запуску Docker на вашому комп'ютері, наприклад, Docker engine, Docker CLI, Docker compose і вбудовані налаштування для використання Kubernetes. Ви також можете використовувати Docker Desktop для доступу до різних служб та інструментів докера, таких як Docker Hub, розширення Docker та сканування.

Docker Desktop надає зручний графічний інтерфейс користувача (GUI), який дозволяє керувати контейнерами, програмами та зображеннями безпосередньо з вашого комп'ютера. Ви можете використовувати інформаційну панель для перегляду стану ваших контейнерів і служб, моніторингу використання ресурсів, перегляду журналів і подій, а також для виконання різних дій, таких як запуск, зупинка, перезапуск, обрізання і видалення. Ви також можете використовувати інформаційну панель для перегляду томів, визначення того, що займає місце, і видалення непотрібних файлів і каталогів. Ви також можете отримати доступ до інтерфейсу командного рядка докера з інформаційної панелі для виконання команд на ваших контейнерах і образах.

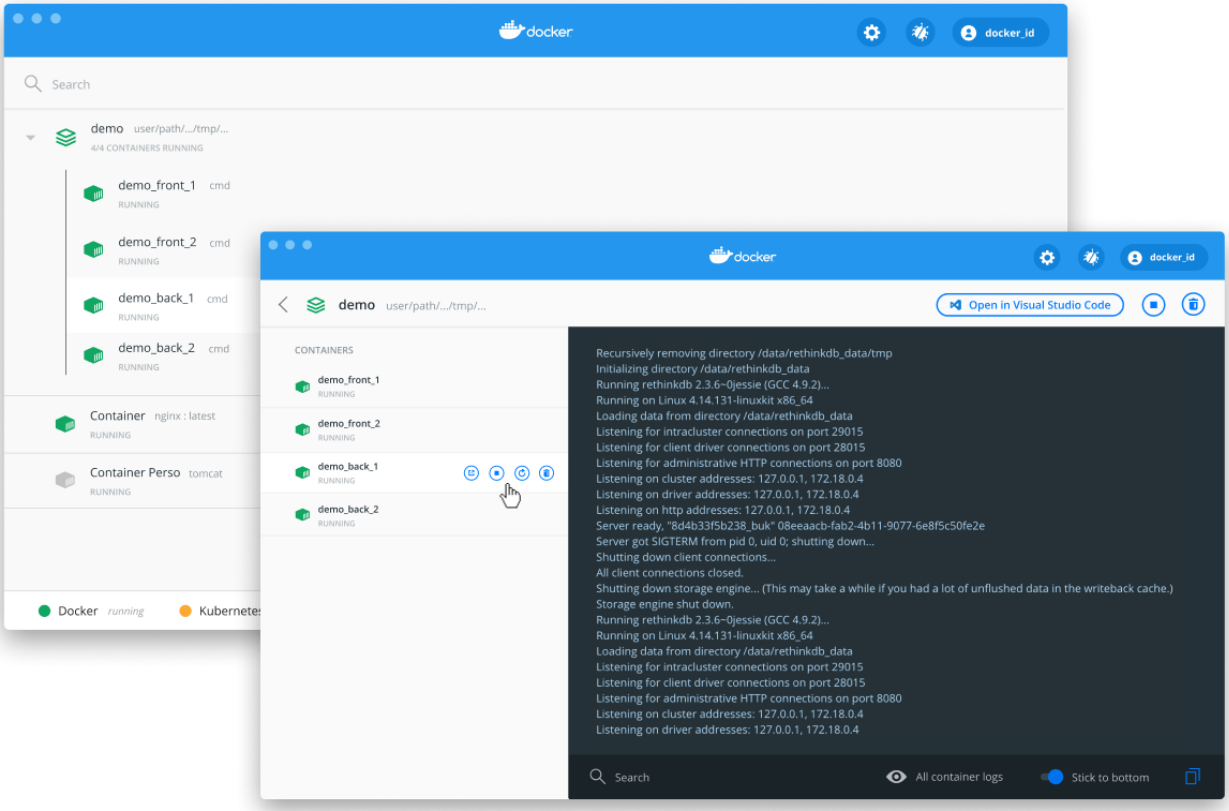

Рис 1.4 Інтерфейс програми Docker Desktop

Робочий стіл Docker дозволяє перемикатися між різними функціями і бекендами залежно від ваших потреб. Ви можете вибирати між контейнерами Linux і контейнерами Windows, якщо ви використовуєте Windows 10 або Windows 11 Professional або корпоративну версію. Ви також можете вибирати між бекендом WSL 2 і бекендом Hyper-V, якщо ви використовуєте Windows 10 або Windows 11 Home Edition. Docker Desktop має функціонал підключення до віддалених хостів Docker або кластерів Kubernetes і керувати ними зі свого комп'ютера.

Docker Desktop також дозволяє інтегруватися з різними інструментами та фреймворками для розробників, які покращують ваш робочий процес і продуктивність. Ви можете використовувати розширення докера для

підключення до готових інструментів для налагодження, тестування, роботи з мережею, безпеки тощо. Ви також можете створювати власні кастомні інструменти і ділитися ними зі своєю командою або з усім світом. Також , ви можете використовувати Docker Desktop для роботи з популярними мовами та фреймворками, такими як Node.js, Python, Java, .NET, Ruby, PHP, Go тощо.

Docker Hub: це сервіс, який дозволяє знаходити образи контейнерів, ділитися ними та керувати ними зі своєю командою або спільнотою Docker. Контейнерні образи - це пакунки, які містять код вашого додатку та всі його залежності, що робить їх готовими до запуску на будь-якій платформі, яка підтримує докер. Docker hub - це найбільша у світі бібліотека та спільнота контейнерних образів, що містить понад 100 000 образів від постачальників програмного забезпечення, проектів з відкритим вихідним кодом та індивідуальних розробників.

Docker hub надає різноманітні функції та переваги для користувачів:

Приватні репозиторії*:* Ви можете створювати і використовувати приватні сховища для зберігання і доступу до власних зображень контейнерів. Ви також можете надавати доступ іншим користувачам або командам для спільної роботи над вашими проектами. Приватні сховища доступні безкоштовно для 5 користувачів і 1 команди, або з платними планами підписки для більших команд і підприємств.

Автоматизовані збірки*:* Ви можете прив'язати свій обліковий запис GitHub або Bitbucket до докер-хабу і автоматично збирати та надсилати зображення контейнерів щоразу, коли ви редактуєте код у своїх сховищах. Ви також можете використовувати веб-хуки, щоб запускати збірки з інших сервісів або платформ. Автоматичні збірки допоможуть вам спростити робочий процес

розробки і гарантувати, що ваші образи завжди будуть актуальними і узгодженими.

Програма для видавців: Ви можете подати заявку на отримання статусу видавця на Docker Hub і пакувати та публікувати свої програми та плагіни у вигляді контейнерів для легкого завантаження та розгортання мільйонами користувачів Docker по всьому світу. Ви також можете використовувати перевірені значки видавця, щоб продемонструвати свою надійність та якість. Програма Publisher допоможе вам охопити ширшу аудиторію і розвивати свій бізнес за допомогою docker hub.

Офіційні образи Docker*:* Ви можете переглядати та використовувати офіційні зображення docker, які куруються та підтримуються співробітниками docker і забезпечують високоякісну базу для створення ваших власних додатків. Офіційні образи докера охоплюють широкий спектр популярних мов, фреймворків, баз даних, операційних систем та інструментів. Ви також можете долучитися до створення офіційних образів докерів, повідомляючи про проблеми або надсилаючи запити на GitHub.

Docker Compose: Docker Compose - це інструмент, який допомагає вам визначати і запускати багатоконтейнерні програми за допомогою Docker. За допомогою Docker Compose ви можете використовувати YAML-файл для налаштування служб вашої програми, таких як їхні образи, порти, томи, мережі, залежності тощо. Потім, за допомогою однієї команди, ви можете створити і запустити всі служби з вашого конфігураційного файлу. Docker Compose працює у всіх середовищах, таких як розробка, тестування, постановка та виробництво.

Docker Compose має кілька переваг для користувачів, а саме:

Простота*:* Ви можете використовувати один файл для опису всього стеку вашого додатку, замість того, щоб використовувати декілька команд або

скриптів Docker. Ви також можете використовувати змінні та шаблони для повторного використання та налаштування конфігурації для різних сценаріїв.

Переносимість: Ви можете легко перенести вашу програму з однієї машини на іншу або з однієї платформи на іншу, просто скопіювавши ваш конфігураційний файл і запустивши Docker Compose. Ви також можете поділитися файлом конфігурації з іншими користувачами і дозволити їм запускати вашу програму з мінімальними налаштуваннями.

Масштабованість: Ви можете розширити вашу програму, змінюючи кількість реплік для кожного сервісу та оновлюючи їх за допомогою Docker Compose. Ви також можете використовувати Docker Compose для інтеграції зі службами оркестрування, такими як Kubernetes або Swarm mode, і розгортати ваш додаток у розподіленому режимі.

Сумісність*:* Ви можете використовувати Docker Compose з будь-яким образом докера або контейнером, незалежно від їх походження або мови. Ви також можете використовувати його з різними інструментами та розширеннями докерів, такими як Docker hub, Docker scan, Docker desktop тощо.

Docker Swarm*:* це функція Docker, яка дозволяє створювати і керувати кластером Docker Engine, який називається роєм(Swarm). Рой складається з одного або декількох вузлів-менеджерів та одного або декількох робочих вузлів. Ноди-менеджери відповідають за організацію та планування завдань, які виконуються на робочих нодах. Робочі вузли - це машини, які виконують завдання як докер-контейнери.

Docker Swarm має кілька переваг для користувачів, а саме

Управління кластером інтегроване з Docker: Ви можете використовувати Docker CLI для створення та приєднання до рою, розгортання

та оновлення сервісів, а також моніторингу стану вашого кластеру. Вам не потрібно ніякого додаткового програмного забезпечення або конфігурації для використання режиму рою.

Децентралізований дизайн: Вузли-розпорядники рою використовують алгоритм консенсусу для підтримки узгодженого стану кластера. Ви можете мати кілька вузлів-розпорядників для забезпечення високої доступності та відмовостійкості. Робочі вузли можуть приєднуватися до рою або залишати його в будь-який час, не впливаючи на загальну функціональність.

Визначена сервісна модель: Ви можете використовувати YAML-файл або докерний компілятор для визначення сервісів вашого додатку та їх конфігурацій, таких як образи, порти, мережі, томи, залежності та політики масштабування. Ви також можете використовувати docker stack deploy для розгортання вашої програми як стека сервісів для рою.

Балансування навантаження і виявлення сервісів: Вузли менеджера рою призначають кожному сервісу унікальне DNS-ім'я і розподіляють навантаження між запущеними контейнерами. Ви також можете використовувати зовнішній балансувальник навантаження, щоб відкрити свої сервіси для зовнішнього світу. Вузли диспетчера рою також підтримують розподілене сховище ключів-значень, яке зберігає інформацію про сервіси та контейнери в рої.

Оновлення та відкат: Ви можете оновлювати свої сервіси з мінімальним часом простою, використовуючи циклічні оновлення. Ви можете вказати параметри оновлення, такі як паралельність, затримка, дії при збої та перевірка працездатності. Ви також можете відкотити свої сервіси до попередньої версії, якщо щось піде не так.

1.3.2 Автоматизація контейнерів – Kubernetes

Що таке Kubernetes ?

Kubernetes - це система з відкритим вихідним кодом для автоматизації розгортання, масштабування та управління контейнерними додатками. Спочатку вона була розроблена компанією Google, а зараз підтримується Фондом хмарних нативних обчислень (Cloud Native Computing Foundation). Kubernetes дозволяє розробникам запускати додатки на різних платформах, таких як публічні, приватні або гібридні хмари, використовуючи загальний набір інструментів та API. Kubernetes працює з контейнерними технологіями, такими як Docker та containerd, для пакування та ізоляції додатків у блоки, які можна розгортати та оновлювати незалежно. Kubernetes також надає такі функції, як виявлення сервісів, балансування навантаження, оркестрування сховища, самовідновлення та горизонтальне масштабування[8].

Назва Kubernetes походить з грецької, що означає "керманич" або "пілот". Kubernetes часто скорочують як K8s, рахуючи вісім літер між K і s (нумеронім).

Kubernetes працює з контейнерами та CRI-O. Її придатність для запуску та управління великими хмарними робочими навантаженнями призвела до широкого впровадження в центрах обробки даних. Існує безліч дистрибутивів цієї платформи - від постачальників, а також хмарні пропозиції від усіх основних постачальників публічних хмар.

Kubernetes надає інструменти та сервіси, які допоможуть вам створювати, розгортати та керувати контейнерними програмами. Для цього він надає інтерфейси API, які дозволяють контролювати та керувати розгортанням, масштабуванням та організацією ваших контейнерних додатків. Kubernetes можна використовувати локально або в хмарі, і наразі це найпопулярніша платформа для керування контейнерними програмами.

Kubernetes, з грецької "керманич, навігатор" або "провідник" та етимологічний корінь кібернетики) був оголошений Google в середині 2014 року. Проект був створений Джо Беда, Бренданом Бернсом і Крейгом Маклакі, до яких незабаром приєдналися інші інженери Google, в тому числі Брайан Грант і Тім Хокін.

На проектування та розробку Kubernetes вплинув менеджер кластерів Google Borg. Багато з його найкращих учасників раніше працювали над Borg; вони дали Kubernetes кодову назву "Проект 7" на честь персонажа "Зоряного шляху", колишнього борга Сьомого з Дев'яти, а його логотипом стало колесо з сімома спицями. На відміну від Borg, який був написаний на C++, вихідний код Kubernetes написаний на мові Go [8].

Kubernetes 1.0 був випущений 21 липня 2015 р. Google працював з Linux Foundation, щоб сформувати Cloud Native Computing Foundation (CNCF) і запропонував Kubernetes як посівну технологію. У лютому 2016 року був випущений менеджер пакетів Helm для Kubernetes.

Як це працює?

Kubernetes визначає набір будівельних блоків ("primitives"), які разом забезпечують механізми розгортання, підтримки та масштабування додатків на основі процесора, пам'яті або користувацьких показників. Kubernetes є слабопов’язаним і розширюваним для задоволення різних робочих навантажень. Внутрішні компоненти, а також розширення та контейнери, які працюють на Kubernetes, покладаються на Kubernetes API[18]. Платформа здійснює контроль над обчислювальними ресурсами та ресурсами зберігання, визначаючи ресурси як об'єкти, якими потім можна керувати як такими.

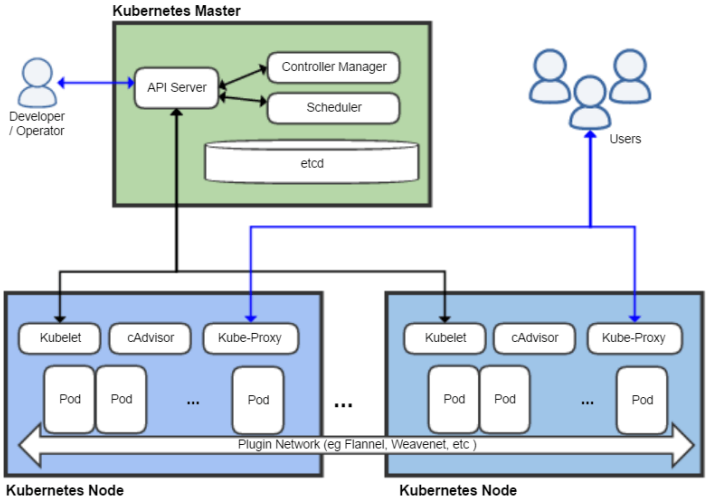

Рис.1.5 Діаграма архітектури Kubernetes

Kubernetes слідує архітектурі Master/slave. Компоненти Kubernetes можна розділити на ті, що керують окремими вузлами, і ті, що є частиною площини управління.

Головний вузол Kubernetes обробляє площину управління кластером Kubernetes, керуючи його робочим навантаженням та спрямовуючи зв'язок по всій системі. Площина керування Kubernetes складається з різних компонентів, кожен з яких є окремим процесом, що може працювати як на одному головному вузлі, так і на декількох головних вузлах, що підтримують кластери високої доступності.

Компоненти площини керування Kubernetes є наступними:

etcd - це постійне, легке, розподілене сховище даних на основі ключ-значення, розроблене CoreOS. Вона надійно зберігає дані конфігурації кластера, що представляють загальний стан кластера в будь-який момент часу. etcd надає перевагу узгодженості, а не доступності у випадку розділу мережі. Узгодженість має вирішальне значення для правильного планування та експлуатації сервісів.

Сервер API

Сервер API обслуговує API Kubernetes за допомогою JSON через HTTP, який забезпечує як внутрішній, так і зовнішній інтерфейс до Kubernetes. Сервер API обробляє та перевіряє REST-запити та оновлює стан об'єктів API в etcd, тим самим дозволяючи клієнтам налаштовувати робочі навантаження та контейнери на робочих вузлах. Сервер API використовує watch API etcd для моніторингу кластера, впровадження критичних змін конфігурації або відновлення будь-яких відхилень стану кластера від заявленого розгортальником. Наприклад, розгортальник може вказати, що мають бути запущені три екземпляри певного "pod". etcd запам'ятовує цей факт. Якщо контролер розгортання виявить, що запущено лише два екземпляри (що суперечить оголошенню etcd), він запланує створення додаткового екземпляра цього "pod"[8]

Kubernetes Scheduler

Kubernetes Scheduler - це розширюваний компонент, який вибирає, на якому вузлі запускати незапланований модуль, ”pod” (базовий об'єкт, яким керує планувальник), виходячи з доступності ресурсів. Планувальник відстежує використання ресурсів на кожному вузлі, щоб гарантувати, що робоче навантаження не перевищує наявних ресурсів. Для цього планувальник повинен знати вимоги до ресурсів, їхню доступність та інші обмеження або директиви політики, надані користувачем, такі як якість обслуговування, вимоги щодо спорідненості та неспорідненості, а також місцезнаходження даних. Роль планувальника полягає в тому, щоб узгодити "пропозицію" ресурсів з "попитом" робочого навантаження.

Контролер

Контролер - це цикл узгодження, який приводить фактичний стан кластера до бажаного, взаємодіючи з сервером API для створення, оновлення та видалення

ресурсів, якими він керує (наприклад, подів або кінцевих точок обслуговування). Одним з видів контролерів є контролер реплікації, який керує реплікацією та масштабуванням шляхом запуску заданої кількості копій одного з модулів у кластері. Інші контролери, які є частиною основної системи Kubernetes, включають контролер DaemonSet для запуску рівно одного модуля на кожній машині (або деякій підмножині машин) і контролер завдань для запуску модулів, які виконуються до завершення (наприклад, як частина пакетного завдання). Селектори міток, які є частиною визначення контролера, визначають набір контейнерів, якими керує контролер[18].

Nodes(вузли):

Вузол, також відомий як робочий або міньйон, - це машина, на якій розгортаються контейнери (робочі навантаження). На кожному вузлі кластера має бути запущено середовище виконання контейнерів, таке як containerd, а також згадані нижче компоненти для зв'язку з основним вузлом для мережевої конфігурації цих контейнерів.

Kubelet

Kubelet відповідає за стан виконання кожного вузла, гарантуючи, що всі контейнери на вузлі є здоровими. Він дбає про запуск, зупинку та підтримку контейнерів додатків, організованих у блоки відповідно до вказівок площини керування. Kubelet відстежує стан блоку, і якщо він не в потрібному стані, блок повторно розгортається на тому ж вузлі[8]. Стан вузла передається кожні кілька секунд за допомогою повідомлень на основну станцію. Як тільки основний вузол виявляє несправність вузла, контролер реплікації спостерігає за зміною стану і запускає подів на інших здорових вузлах.

Kube-proxy

Kube-proxy є реалізацією мережевого проксі і балансувальника навантаження, і підтримує абстракцію сервісу разом з іншими мережевими

операціями. Він відповідає за маршрутизацію трафіку до відповідного контейнера на основі IP і номера порту вхідного запиту.

Контейнер

Контейнер знаходиться всередині pod-у. Контейнер - це найнижчий рівень мікросервісу, який містить запущену програму, бібліотеки та їх залежності. Контейнери можуть бути доступними для світу через зовнішню IP-адресу. Kubernetes підтримує контейнери Docker з першої версії.

Простори імен

У Kubernetes простори імен використовуються для розділення ресурсів, якими він оперує, на окремі колекції, що не перетинаються. Вони призначені для використання у середовищах з великою кількістю користувачів, розподілених між кількома командами або проектами, або навіть для розділення середовищ, таких як розробка, тестування та виробництво.

Pods(контейнери)

Базовою одиницею планування у Kubernetes є блок, який складається з одного або декількох контейнерів, які гарантовано розміщуються на одному вузлі. Кожному блоку у Kubernetes присвоюється унікальна IP-адреса у кластері, що дозволяє програмам використовувати порти без ризику конфлікту. У межах блоку всі контейнери можуть посилатися один на одного.

DaemonSets

Зазвичай, завдання визначення вузла, на якому має бути запущено “pod”, делегується планувальнику Kubernetes Scheduler. Однак, у певних сценаріях може знадобитися розгортання “pod-ів”, на кожному вузлі кластера, що особливо корисно для випадків використання, пов'язаних зі збором журналів,

контролерами вхідних даних і службами зберігання даних. Цей специфічний тип планування роботи подів може бути досягнутий за допомогою DaemonSets.

ReplicaSets

Метою ReplicaSet є підтримка стабільного набору реплік “pod-ів”, запущених у будь-який момент часу. Таким чином, він часто використовується для гарантування доступності певної кількості однакових “pod”.

Також можна сказати, що ReplicaSets є механізмом групування, який дозволяє Kubernetes підтримувати кількість екземплярів, які було оголошено для певного пакунка[8]. У визначенні ReplicaSet використовується селектор, обчислення якого призведе до визначення усіх пов'язаних з ним pod'ів.

Сервіс Kubernetes - це набір блоків, які працюють разом, наприклад, один рівень багаторівневого додатку. Набір контейнерів, які складають сервіс, визначається селектором міток. Kubernetes надає два способи виявлення служби: за допомогою змінних середовища або за допомогою Kubernetes DNS(Domain Name Service). Служба виявлення призначає службі стабільну IP-адресу та DNS-ім'я, а також циклічно розподіляє трафік між мережевими з'єднаннями з цією IP-адресою, що відповідають селектору (навіть якщо збої спричиняють переміщення служби з машини на машину). За замовчуванням сервіс розміщується всередині кластера (наприклад, внутрішні вузли можуть бути згруповані в сервіс, а запити від зовнішніх вузлів розподіляються між ними), але сервіс також може бути розміщений за межами кластера (наприклад, щоб клієнти могли отримати доступ до зовнішніх вузлів)[8].

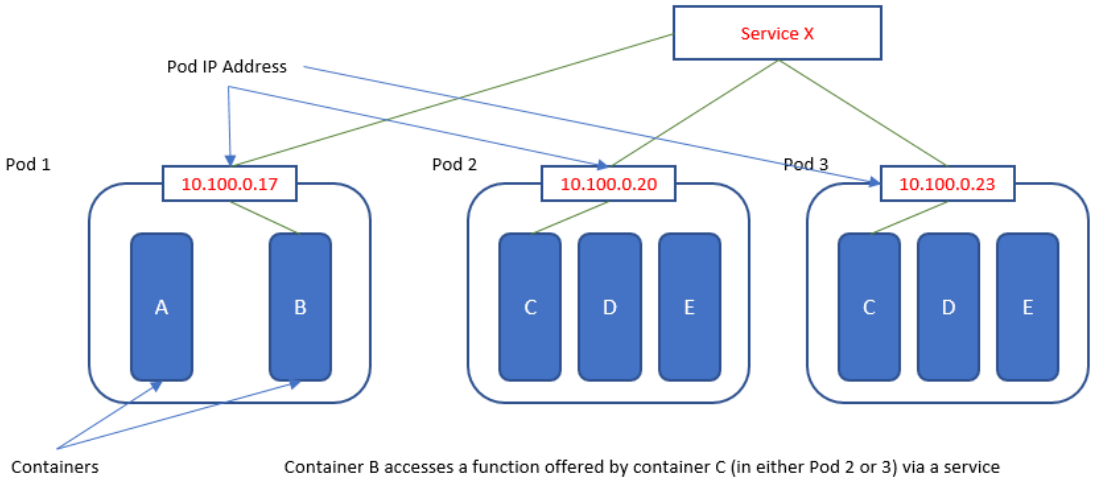

Рис.1.6 Спрощена схема, що показує, як служби взаємодіють з мережею Pod у кластері Kubernetes

Томи (файлова система)

Файлові системи в контейнері Kubernetes за замовчуванням забезпечують ефемерне зберігання даних. Це означає, що перезапуск поду знищить всі дані на таких контейнерах, а отже, ця форма зберігання даних є досить обмеженою у всьому, що не стосується тривіальних додатків. Том Kubernetes забезпечує постійне сховище, яке існує протягом усього життя самого тома. Це сховище також може використовуватися як спільний дисковий простір для контейнерів в межах тома. Томи монтуються у певні точки монтування всередині контейнера, які визначаються конфігурацією тома, і не можуть монтуватися на інші томи або посилатися на інші томи. Один і той самий том може бути змонтовано у різних точках дерева файлової системи різними контейнерами.

ConfigMaps and secrets

Поширеною проблемою при застосуванні є рішення про те, де зберігати та керувати інформацією про конфігурацію, деякі з яких можуть містити конфіденційні дані. Дані конфігурації можуть бути як дрібнозернистими, наприклад, окремі властивості, так і крупнозернистими, наприклад, цілі конфігураційні файли або документи JSON/XML[18]. Kubernetes надає два тісно

пов'язаних механізми для вирішення цієї проблеми: "configmaps" і "secrets", обидва з яких дозволяють вносити зміни до конфігурації без необхідності збирання програми. Дані з конфігураційних карт і секретів будуть доступні кожному екземпляру програми, до якого ці об'єкти було прив'язано під час розгортання. Секрет та/або конфігураційна карта надсилається на вузол лише у тому випадку, якщо це потрібно для кадру на цьому вузлі. Kubernetes зберігатиме їх у пам'яті цього вузла. Як тільки pod, який залежить від секрету або конфігураційної карти, видаляється, копія в пам'яті всіх пов'язаних секретів і конфігураційних карт також видаляється. Доступ до даних у під можна отримати одним із двох способів:

- як змінні оточення (які будуть створені Kubernetes під час запуску подів);
- доступні у файловій системі контейнера, яку можна побачити лише зсередини подів.

Самі дані зберігаються на головному комп'ютері, який є високо захищеною машиною, до якої ніхто не повинен мати доступ через логін і пароль. Найбільша різниця між секретом і конфігураційною картою полягає в тому, що вміст даних у секреті кодується в base64. В останніх версіях Kubernetes також з'явилася підтримка шифрування. Секрети часто використовуються для зберігання таких даних, як сертифікати, облікові дані для роботи з реєстрами образів, паролі та ключі ssh.

StatefulSets

Масштабування додатків - це лише питання додавання більшої кількості запущених модулів. З робочими навантаженнями зі станом складніше, тому що стан повинен зберігатися при перезавантаженні модуля. Якщо додаток розширяється в більший або менший розмір, може знадобитися його перерозподілення. Бази даних є прикладом робочих навантажень. При запуску в режимі високої доступності багато баз даних мають поняття первинного

екземпляру та вторинних екземплярів. У цьому випадку важливим є поняття впорядкування екземплярів. Інші програми, такі як Apache Kafka, розподіляють дані між своїми брокерами; отже, один брокер не є тим самим, що й інший. У цьому випадку важливим є поняття унікальності екземплярів.

StatefulSets - це контролери, які забезпечують такі властивості як унікальність та впорядкованість серед екземплярів pod-ів і можуть бути використані для запуску додатків, що працюють у стані.

Replication controllers and deployments

ReplicaSet декларує кількість екземплярів потрібного пакета, а контролер реплікації керує системою таким чином, щоб кількість працюючих здорових пакунків відповідала кількості пакунків, декларованих у ReplicaSet (визначеній шляхом оцінки його селектора).

Розгортання - це механізм керування наборами реплік вищого рівня ReplicaSet . У той час як контролер реплікації керує масштабуванням набору ReplicaSet, розгортання керує тим, що відбувається з набором ReplicaSet - чи потрібно розгортати оновлення, чи відкочувати його назад тощо. Коли розгортання йде вгору або вниз, це призводить до зміни декларації ReplicaSet - і цією зміною декларованого стану керує Replication Controller.

Labels and selectors

Kubernetes дозволяє клієнтам (користувачам або внутрішнім компонентам) прикріплювати ключі, які називаються "мітками", до будь-якого об'єкту API у системі, наприклад, pod-ів та вузлів(nodes). Відповідно, "селектори міток" - це запити до міток, які перетворюються на відповідні об'єкти. Коли визначено сервіс, можна визначити селектори міток, які будуть використовуватися маршрутизатором/балансировщиком навантаження сервісу для вибору екземплярів подів, до яких буде скеровано трафік[18]. Таким чином, простою

зміною міток подів або зміною селекторів міток на службі можна керувати тим, які поди отримують трафік, а які ні, що може бути використано для підтримки різних моделей розгортання, таких як синьо-зелене розгортання або A-B тестування. Ця можливість динамічно керувати тим, як сервіси використовують ресурси для реалізації, забезпечує вільний зв'язок в інфраструктурі.

Сховище

Контейнери з'явилися як спосіб зробити програмне забезпечення портативним. Контейнер містить усі пакунки, необхідні для запуску сервісу. Надана файлова система робить контейнери надзвичайно портативними і простими у використанні під час розробки. Контейнер можна переносити зі стадії розробки на стадію тестування або виробництва без змін конфігурації або з відносно невеликими змінами.

Історично Kubernetes підходив лише для “stateless” сервісів. Однак багато додатків мають базу даних, яка вимагає постійності, що призводить до створення постійного сховища для Kubernetes. Реалізація постійного сховища для контейнерів є одним з головних завдань адміністраторів Kubernetes, DevOps та хмарних інженерів. Контейнери можуть бути ефемерними, але все більше і більше даних в них не є такими, тому потрібно забезпечити виживання даних у разі завершення роботи контейнера або відмови обладнання. Розгортаючи контейнери з Kubernetes або контейнерними додатками, компанії часто усвідомлюють, що їм потрібне постійне сховище. Їм потрібно забезпечити швидке і надійне сховище для баз даних, кореневих образів та інших даних, що використовуються контейнерами.

На додаток до ландшафту, Cloud Native Computing Foundation (CNCF) опублікував іншу інформацію про Kubernetes Persistent Storage, включаючи блог, який допомагає визначити шаблон контейнерного сховища. Цю модель можна

розглядати як таку, що використовує сам Kubernetes як компонент системи зберігання або сервісу.

Більше інформації про відносну популярність цих та інших підходів можна знайти в ландшафтному дослідженні CNCF, яке показало, що OpenEBS від MayaData та Rook - проект оркестрування сховищ - були двома проектами, які, найімовірніше, оцінювалися станом на осінь 2019 року.

Container Attached Storage

Container Attached Storage - це тип сховища даних, який з'явився після того, як Kubernetes набув популярності. Підхід або шаблон Container Attached Storage покладається на сам Kubernetes для отримання певних можливостей, надаючи в основному блоки, файли, об'єкти та інтерфейси для робочих навантажень, що працюють на Kubernetes.

Загальні атрибути Container Attached Storage включають використання розширень для Kubernetes, таких як користувацькі визначення ресурсів, та використання самого Kubernetes для функцій, які в іншому випадку були б окремо розроблені та розгорнуті для зберігання або управління даними. Прикладами функціональності, яку надають користувацькі визначення ресурсів або сама Kubernetes, є логіка повторних спроб, яку надає сама Kubernetes, а також створення та підтримка інвентаризації доступних носіїв та обсягів зберігання, що зазвичай надається за допомогою користувацького визначення ресурсу.

API

Ключовим компонентом площини керування Kubernetes є сервер API, який надає HTTP API, що може бути викликаний іншими частинами кластера, а також кінцевими користувачами та зовнішніми компонентами. Цей API є

Representational state transfer (REST) API і є декларативним за своєю природою. Існує два типи ресурсів API. Більшість ресурсів API у Kubernetes API є об'єктами. Вони представляють конкретний екземпляр концепції на кластері, наприклад, блок або простір імен. Невелика кількість типів ресурсів API є "віртуальними". Вони представляють операції, а не об'єкти, такі як перевірка дозволів за допомогою ресурсу "subjectaccessreviews". Ресурси API, які відповідають об'єктам, будуть представлені у кластері з унікальними ідентифікаторами об'єктів. Віртуальні ресурси не мають унікальних ідентифікаторів[18].

Operators

Kubernetes можна розширити за допомогою користувацьких ресурсів. Ці ресурси API представляють об'єкти, які не є частиною стандартного продукту Kubernetes. Ці ресурси можуть з'являтися і зникати у працюючому кластері за допомогою динамічної реєстрації. Адміністратори кластера можуть оновлювати користувацькі ресурси незалежно від кластера.

Користувацькі контролери - це ще один механізм розширення. Вони взаємодіють з користувацькими ресурсами і надають справжній декларативний API, який дозволяє керувати життєвим циклом користувацького ресурсу відповідно до того, як розроблено сам Kubernetes. Поєднання користувацьких ресурсів та користувацьких контролерів часто називають оператором (Kubernetes). Основним варіантом використання операторів є фіксація мети людини-оператора, яка керує сервісом або набором сервісів, і реалізація їх за допомогою автоматизації та декларативного API, що підтримує цю автоматизацію. Люди-оператори, які опікуються конкретними додатками і сервісами, мають глибокі знання про те, як має поводитися система, як її розгортати і як реагувати у разі виникнення проблем. Приклади проблем, які вирішують оператори, включають створення та відновлення резервних копій

стану програми, а також оновлення коду програми разом із супутніми змінами, такими як схеми баз даних або додаткові налаштування конфігурації[18].

Cluster API

Ті ж принципи розробки API були використані для визначення API для програмного створення, конфігурації та управління кластерами Kubernetes. Ключовою концепцією, втіленою в API, є використання інфраструктури як програмного забезпечення, або уявлення про те, що інфраструктура кластера Kubernetes сама по собі є ресурсом/об'єктом, яким можна керувати так само, як і будь-якими іншими ресурсами Kubernetes. Аналогічно, машини, які складають кластер, також розглядаються як ресурс Kubernetes. API складається з двох частин - основного API та реалізації провайдера. Реалізація провайдера складається зі специфічних для хмарного провайдера функцій, які дозволяють Kubernetes надавати кластерний API у спосіб, який добре інтегрований зі службами та ресурсами хмарного провайдера.

1.3.4 Практика CI/CD

CI/CD, що розшифровується як “continuous integration (CI) and continuous delivery (CD)” або безперервна інтеграція та безперервне розгортання/поставка, - це практика розробки програмного забезпечення, яка революціонізує спосіб створення, тестування та розгортання додатків та продуктів. Це невід'ємна частина сучасних процесів розробки програмного забезпечення, що дозволяє командам створювати високоякісне програмне забезпечення швидше та ефективніше[19].

Безперервна інтеграція (Continuous Integration, CI) - це процес інтеграції змін коду від декількох розробників у спільний репозиторій. За допомогою CI розробники інтегрують свої зміни коду в центральну кодову базу, часто по кілька разів на день. Такий підхід допомагає виявити проблеми інтеграції нового коду на ранніх стадіях і сприяє співпраці між членами команди розробників.

Автоматизовані інструменти збірки використовуються для компіляції коду, запуску тестів і перевірки змін. Якщо виявляються якісь проблеми, розробники можуть швидко їх вирішити, що призводить до пришвидшення зворотного зв'язку та покращення якості коду[20].

Безперервне розгортання (CD) або безперервна доставка (CD) - це наступний крок у процесі CI/CD. Він передбачає автоматизацію розгортання додатків у різних середовищах, таких як розробка, тестування та готовий продукт. CD гарантує, що випуски програмного забезпечення будуть надійними, частими в випуску нових оновлень і можуть бути розгорнуті в будь-який час з мінімальним ручним втручанням. Автоматизувавши процес розгортання, команди можуть уникати людські помилки і скоротити час, необхідний для надання нових функцій або виправлення помилок у кінцевому продукті. CD також дозволяє організаціям впроваджувати такі практики, як синьо-зелене розгортання(blue-green deployment) або канаркові збірки ПЗ (canary), які ще більше підвищують надійність і стабільність їхніх додатків [20].

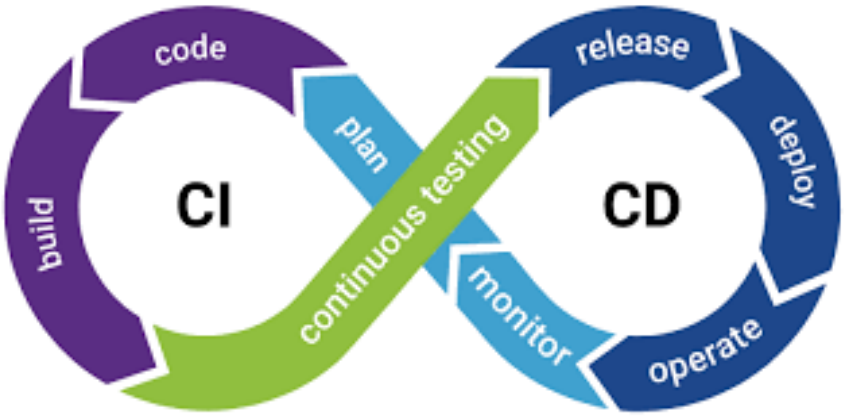

Рис.1.7 Схема циклу CI/CD

CI/CD в більшості використовує автоматизацію та принципи «інфраструктури як коду» (IaC). Інфраструктура та конфігурація управляються програмно, що дозволяє командам визначати свої вимоги до інфраструктури за

допомогою коду. Такий підхід забезпечує послідовне та відтворюване розгортання в різних середовищах, зменшує кількість проблем, пов'язаних з розгортанням, та покращує можливості розвертання додатків.

Автоматизоване тестування є важливим аспектом CI/CD. Тестування виконується на різних рівнях, включаючи модульні, інтеграційні та наскрізні тести, щоб забезпечити якість і надійність програмного забезпечення. Тестування інтегровано в конвеєр CI/CD, який запускається автоматично з кожною зміною коду. Це допомагає виявляти помилки та проблеми на ранніх стадіях процесу розробки, дозволяючи розробникам вирішувати їх до того, як вони потраплять у готовий продукт[21].

Переваги CI/CD численні. Вони сприяють співпраці між членами команди, покращують якість коду, знижують ризик внесення помилок і забезпечують швидкий зворотній зв'язок. Автоматизуючи процеси збірки, тестування та розгортання, CI/CD збільшує швидкість доставки програмного забезпечення, дозволяючи організаціям швидко реагувати на потреби клієнтів та вимоги ринку. Це також підвищує загальну стабільність і надійність додатків, оскільки проблеми виявляються на ранніх стадіях і можуть бути оперативно вирішені.

Ну і звісно ж, CI/CD відіграє вирішальну роль в архітектурі Cloud Native. Cloud Native відноситься до принципів проектування та розробки, які дозволяють цим високорозвиненим додаткам ефективно працювати в хмарних середовищах.

CI/CD є важливою практикою в екосистемі Cloud Native, оскільки вона зосереджена на автоматизації процесів створення, тестування та розгортання додатків. Це забезпечує оптимізацію життєвого циклу розробки програмного забезпечення та сприяє частішим і надійнішим випускам [20].

Ось як CI/CD сприяє архітектурі Cloud Native:

Безперервна інтеграція: CI передбачає регулярну інтеграцію змін коду від декількох розробників у спільний репозиторій проекту. Автоматизовані інструменти збірки компілюють код, запускають модульні тести та перевіряють наявність проблем з інтеграцією. В архітектурі Cloud Native CI гарантує, що окремі компоненти, розроблені різними командами, можуть бути безперешкодно інтегровані в додаток.

Безперервне розгортання/доставка: CD - це процес автоматизації розгортання додатків у виробничих середовищах. Він включає такі завдання, як пакування програми, налаштування інфраструктури та розгортання в різних середовищах, таких як розробка, стадія та виробництво. В архітектурі Cloud Native CD забезпечує часте та поетапне розгортання, гарантуючи швидке впровадження нових функцій та виправлення помилок, мінімізуючи час простою [19].

Мікросервіси та контейнери*:* Додатки Cloud Native зазвичай будуються з використанням архітектури мікросервісів, де додаток розбивається на менші, слабко пов'язані між собою сервіси. CI/CD забезпечує незалежну розробку та розгортання цих мікросервісів. Контейнери, такі як Docker, часто використовуються для пакування сервісів, а конвеєри CI/CD автоматизують створення та розгортання контейнерних додатків.

Інфраструктура як код (IaC): Нативна хмарна архітектура значною мірою покладається на автоматизацію інфраструктури. Конвеєри CI/CD часто використовують інструменти надання інфраструктури, такі як Terraform або AWS CloudFormation, для визначення та управління хмарною інфраструктурою,

необхідною для додатку. Такий підхід забезпечує узгодженість, масштабованість і відтворюваність інфраструктури в різних середовищах.

Автоматизоване тестування та забезпечення якості: Конвеєри CI/CD включають автоматизоване тестування на різних рівнях, включаючи модульні, інтеграційні та наскрізні тести. Ці тести допомагають забезпечити якість і стабільність роботи програми після внесення змін. В архітектурі Cloud Native процес CI/CD також включає механізми моніторингу та спостереження для виявлення та вирішення проблем в режимі реального часу[19].

Загалом, практики CI/CD в архітектурі Cloud Native спрощують процес розробки програмного забезпечення, уможливлюють швидке і часте розгортання, забезпечують масштабованість і відмовостійкість, а також сприяють співпраці між командами розробників і операторів. Автоматизуючи різні етапи життєвого циклу програми, CI/CD підвищує ефективність і надійність додатків Cloud Native в динамічних і розподілених хмарних середовищах.

Висновки

Отже, Cloud Native - це підхід до створення та запуску масштабованих додатків, які використовують переваги хмари. Додатки Cloud Native складаються з слабко пов'язаних мікросервісів, які працюють на контейнерах і організовуються за допомогою Kubernetes. Хмарні додатки є більш стійкими, гнучкими та економічно ефективними, ніж традиційні. Cloud native може допомогти різним галузям, наприклад телекомунікації, трансформувати надання послуг та впроваджувати інновації. Cloud native - це не лише технічний вибір, але й культурний та організаційний. Він вимагає прийняття нових практик, таких як DevOps та безперервна доставка. Cloud native - це шлях вперед для сучасної розробки та розгортання програмного забезпечення.

РОЗДІЛ 2
CLOUD NATIVE В МЕРЕЖАХ МОБІЛЬНОГО ЗВ’ЯЗКУ

2.1 Існуючі хмарні технології мобільних операторів

Протягом тривалого часу, телекомунікаційні системи та мобільні оператори зв'язку покладалися в основному на фізичні сервера для запуску своїх функцій і додатків. Програмне забезпечення в цих системах було більше схоже на мікропрограму, і його не можна було легко відокремити або доповнити для розширення функцій.

Це призвело до того, що розвиток був або мізерний, або його взагалі не було. Такий підхід був дорогим і ресурсномістким, а наявність резервних систем на випадок розширення інфраструктури означала б купівлю додадткових, часто лишніх фізичних копій тих самих систем, які в подальшому часі майже не використовуються або до моменту використання просто стають застарілими. Такий архітектурний підхід змушував сервіси і компанії об'єднувати і інтегруватися в єдині фізичні системих[24]. Результатом такого підходу стали неефективна надмірність системної інженерії для дублювання можливостей а також високі витрати на інфраструктуру та забезпечення наявних потужностей і персоналу.

Віртуалізація для корпоративних компаній стала популярною наприкінці 1990-х років, коли підприємства були вимушені заощаджувати на потужних і численних серверах. У той же час, почала набувати широкого розповсюдження операційна система з відкритим вихідним кодом Linux, яка стала початком сучасної ери. У 1999 році компанія VMware випустила свій перший продукт віртуалізації для пристроїв архітектури x86. На початку 2010-х років була представлена віртуалізація мережевих функцій (NFV) як спосіб віртуалізації мережевих функцій, підвищення відмовостійкості та використання серверного обладнання телекомунікаційними компаніями та іншими великими мережевими користувачами[23]. Маючи в використанні такі технології, як віртуальні мережеві функції (VNF), інфраструктура мережевих функцій (NFVI) та

компонент оркестрування (MANO), весь набір апаратного і програмного забезпечення NFV став можливий для інтеграції та розповсюдження між центрами обробки даних у всьому світі. У 2017 році Open Networking Automation Platform (ONAP) стала базовою платформою з відкритим готовим кодом для оркестрування та автоматизації фізичних і віртуальних елементів мережі з їх повним управлінням та можливістю розширення. NFV і ONAP в основному були зосереджені на створенні та організації віртуальних одиниць обланднання, яке б замінили фізичне - віртуальних боксів. Так само було під час початку віртуалізації серверів, коли початкові зусилля були зосереджені на створенні віртуальних копій фізичних серверів[24].

NFV - це технологія, яка дозволяє мережевим або мобільним операторам запускати мережеві функції як програмні додатки на стандартному обладнанні замість використання свого, спеціального навченого і налаштованого обладнання.

Мережеві функції - це такі сервіси, як маршрутизатори, брандмауери та балансувальники навантаження, які традиційно працювали на пропрієтарному обладнанні. NFV дозволяє мережевим операторам економити витрати, простір, електроенергію та обслуговування, а також легше модернізувати свої мережі.

NFV базується на модульній архітектурі, яка складається з чотирьох основних компонентів:

Віртуалізовані мережеві функції (Virtualized network functions) (VNF) - це програмні додатки, які надають мережеві функції, такі як спільний доступ до файлів, служби каталогів і конфігурація IP.

Інфраструктура віртуалізації мережевих функцій (Network functions virtualization infrastructure) (NFVI) складається з компонентів інфраструктури -

комп'ютерів, сховищ, мереж - на платформі для підтримки програмного забезпечення, такого як hypervisor, наприклад, KVM, або платформа управління контейнерами, необхідного для запуску мережевих додатків[24].

Управління, автоматизація та мережева оркестрація (Management, automation and network orchestration) (MANO) забезпечує основу для управління інфраструктурою NFV і створення нових VNF.

2.2 Перехід з VNF до CNF та злиття з існуючими технологіями

Почнемо з поняття CNF - це сервіс, який виконує мережеві функції за допомогою програмного забезпечення, на відміну від спеціально створеного апаратного забезпечення[25]. Управління мережевими функціями за допомогою програмного забезпечення стало можливим завдяки великим і недорогим ресурсам сучасних центрального процесора і пам'яті, доступним в серверних платформах сьогодення.

Раніше такий тип обчислювальних потужностей можна було реалізувати лише за допомогою спеціалізованих інтегральних схем, які свого часу були обов'язковими у фізичних серверах,тобто апаратний підхід. Оскільки CNF повністю засновані на програмному забезпеченні, вони використовують віртуальні інтерфейси замість фізичних. Прикладами CNF, які використовуються сьогодні, є маршрутизатори, брандмауери, віртуальні комутатори та шлюзи віртуальних приватних мереж[25].

Kubernetes, найпоширеніший рівень контейнерної оркестрації, дозволяє телекомунікаційним операторам перевести свою інфраструктуру з більш складних, дорогих і складних технологій на одну платформу, що підвищує продуктивність і відмовостійкість. Розширюваність Kubernetes повинна

дозволити поступово переходити від застарілих віртуальних мереж, створених на OpenStack або віртуальних машинах VMWare, до телекомунікаційного стека Kubernetes. Це дозволяє керувати VNF за допомогою абстрактного рівня Kubernetes (KubeVirt, Virtlet, Openstack).

Офісні додатки, такі як функції BSS та OSS (Operation Support System/Business Support System), можуть працювати безпосередньо на Kubernetes і користуватися всіма тими ж перевагами, що й корпоративні додатки, які отримують від Kubernetes. Оператори також повинні мати можливість запускати критичні функції в режимі реального часу на Kubernetes[23]. CNF повинні мати можливість працювати на Kubernetes і отримувати прямі та значні переваги від принципів хмарних технологій. Важливо зазначити, що теоретично Kubernetes і контейнеризація повинні забезпечувати справжню інфраструктурну незмінність і переносимість[23]. Це буде так, якщо оператори використовуватимуть і підтримуватимуть відповідні версії та конфігурації Kubernetes. Тим не менш, створення моделі відповідності Kubernetes для телекомунікацій є нетривіальним завданням, що вимагає перевірки та сертифікації ключових елементів, таких як мережеве програмне забезпечення (плагін CNI, мережева сервісна сітка і т.д.) та інших аспектів.

У чому ж різниця між VNF і CNF?

CNF - це не єдиний спосіб перенесення мережевих сервісів з апаратних пристроїв на програмне забезпечення. Інший спосіб - використання віртуальних мережевих функцій (VNF).

За допомогою VNF те саме програмне забезпечення, яке працює на апаратному мережевому пристрої, переноситься на віртуальну машину (VM). Єдина відмінність полягає в тому, що обробка та використання портів відбувається в програмному забезпеченні, а не в апаратному. Це дуже

відрізняється від CNF, які вибирають конкретні необхідні мережеві сервіси і запускають їх у контейнерному середовищі, такому як Kubernetes, а не у VM [25].

Перевага CNF полягає в тому, що обробка та виділення пам'яті потрібні лише для конкретних сервісів, а самі сервіси можуть бути розподілені по мережі залежно від того, де вони потрібні. Це означає, що CNF можуть досягти кращої ефективності, покращення можливостей розширення та продуктивності, ніж VNF.[23]

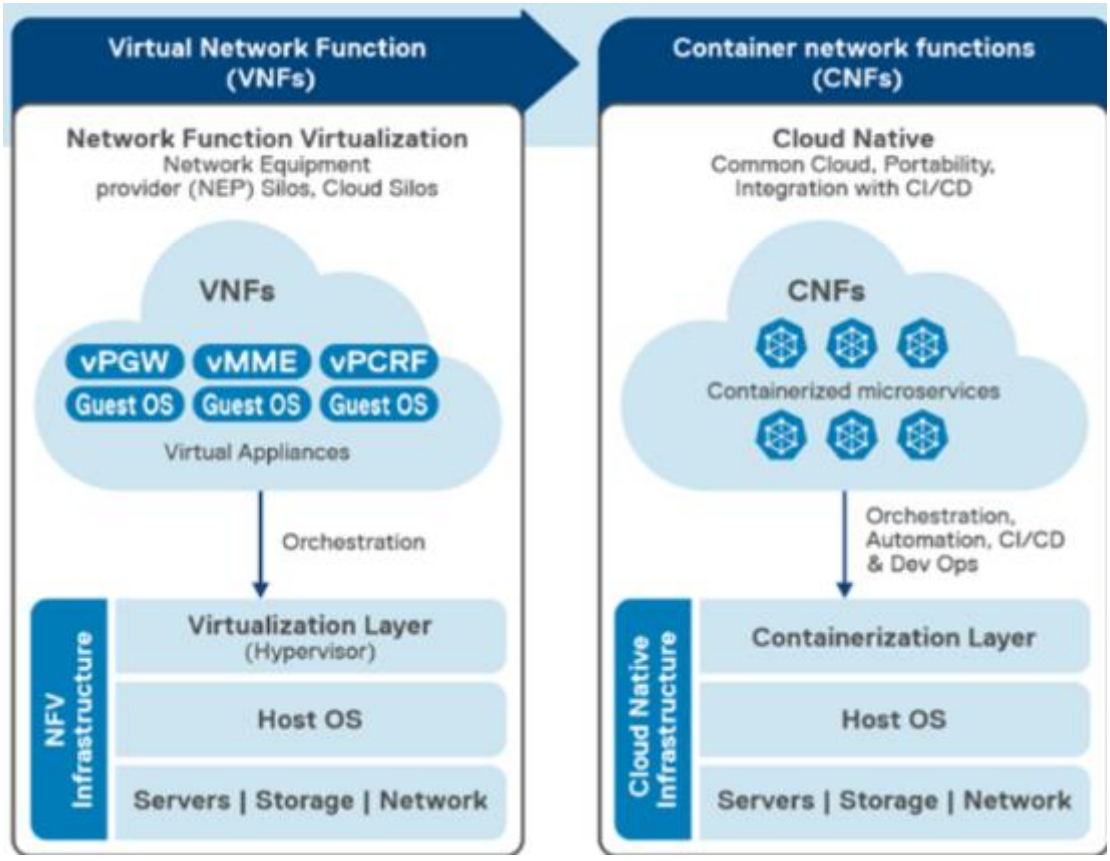

Рис.2.1 Порівняння VNF та CNF функцій

Переваги CNF над VNF:

- CNF пропонує кращу ефективність використання ресурсів за рахунок запуску більшої кількості сервісів на одному сервері (використовуючи власну структуру мікросервісів та концепцію контейнеризації).

- Краща відмовостійкість і вища доступність, оскільки мікросервіси розподілені на декількох серверах і машинах, а навантаження на обробку розподіляється спільно між потужностями.
- Cloud Native забезпечує вищу швидкість розробки для розширення мережі за допомогою оркестратора Kubernetes.
- CNF дозволяє зменшити час простою в мережі за рахунок оновлення мікросервісів (оскільки послуги обробляються, наприклад, 100 мікросервісами, для оновлення або зміни в мережі, завдання може бути виконано для 10 мікросервісів одночасно, а решта 90 мікросервісів будуть виконувати цю роботу замість них).

Незважаючи на це, вони також створюють перешкоди, які необхідно обговорювати в майбутньому переході. Враховуючи очевидні переваги CNF у ролі забезпечення економії капітальних і операційних витрат компаній і прискорення темпів впровадження інновацій, постачальники серверів мають великий спокуса перепозиціонувати віртуалізовані мережеві функції як «хмарні».

Локальне залізо нерозривно пов'язане з кодом.

Обробка без локальних серверів є найважливішою особливістю хмарної архітектури. Але в традиційних архітектурах додатків всі процеси зберігаться локально, а доступ до цих елементів або їх оновлення здійснюється окремими інструкціями по всьому коду[23]. Підсумуючи, в більшості випадків це являтиме собою майже повне переписування коду, що буде дуже затратним по всім аспектам, що майже поставить питання переходу на паузу. Щоб створити справжній CNF, розробникам дійсно потрібно починати з нуля, а не зі старої кодової бази.

Монолітні додатки важко декомпонувати

Хоча застарілий код додатків зазвичай демонструює високий ступінь модульності у вигляді викликів функцій та підпрограм, ці модулі рідко пропонують можливість Reverse Engeneering-гу коду програми на слабко пов'язані мікросервіси, які потрібні при переході. Частіше всього це відбувається через взаємну залежність від спільних даних в структурах додатку. Архітектура мікросервісів вимагає усунення таких залежностей, а це зазвичай вимагає повного переосмислення архітектури додатку.

Висновки

Підводячи підсумки, надзвичайно складно розбирати монолітне, застаріле програмне забезпечення для мережевих функцій, щоб втілити ключові архітектурні аспекти технологій Cloud Native, які є важливими для отримання їх переваг.

Для того, щоб ці висновки були враховані, CNF повинні бути впроваджені на рівні VNF і вписатися в операційні моделі, які керують цими попередніми типами мережевих функцій. Успіх великих телекомунікаційних компаній у впровадженні технологій Cloud Native залежатиме від пошуку точок і методів втілення старих і нових cloud технологій. Технічні та архітектурні відділи повинні визначити, як впровадити CNF в існуючі домени рівнів 2 і 3, інтегрувати CNF в існуючі системи управління, які зараз присутні в мережах постачальників послуг. Вони також повинні визначити, як впровадити практики DevOps, такі як безперервна інтеграція (CI) і безперервне розгортання/доставка (CD).

РОЗДІЛ 3
ЗАПРОПОНОВАНІ CLOUD NATIVE РІШЕННЯ В ГАЛУЗІ МОБІЛЬНИХ МЕРЕЖ

У міру розвитку хмарної трансформації провідні оператори по всьому світу використовують телекомунікаційні хмари на основі NFV, щоб побудувати ефективні, гнучкі базові мережі. Телекомунікаційна галузь розглядає телекомунікаційні хмари наступного покоління на основі Cloud Native як рішення для пом'якшення потенційного впливу на непередбачувані ситуації на ринку і постійно вивчає потенціал цієї технології, зіткнувшись з складними вимогами 5G у різних галузях.

Удосконалений на основі NFV, архітектура Cloud Native була запозичена з IT сфери. Це системна парадигма для розробки, запуску та управління програмним забезпеченням у хмарному середовищі, яка включає основні технології та інноваційний досвід[26]. Впровадження в галузі телекомунікацій Cloud Native полегшує створення мереж наступного покоління, які відрізняються еластичністю ресурсів, надійністю та гнучкістю на основі віртуалізації.

З великою ймовірністю моделі розгортання Cloud Native додатків у телекомунікаційних компаніях будуть переходити від функцій, які є більш відмовостійкими або менш чутливими до затримок (голосова пошта, OSS/BSS тощо), до більш важливих функцій. Телекомунікаційні компанії отримують багато переваг від запуску хмарних систем. Операційну узгодженість, відмовостійкість додатків, спрощене і швидке збільшення можливостей на рівні мікросервісів, спрощена інтеграція з корпоративними Cloud Native додатками та покращена переносимість між публічними, приватними та гібридними хмарними середовищами є одними з цих переваг [27]. Завдяки використанню мережевих функцій під управлінням Kubernetes, а не спеціалізованих серверів, прозорість,

спостережливість і контроль покращаться. Телекомунікаційні оператори будуть отримувати вигоду від вбудованих або природно інтегрованих інструментів і метрик, таких як Prometheus, Jaeger і OpenTracing, оскільки їх простіше вимірювати та спостерігати на рівні послуг в мікросервісах Kubernetes, що працюють під управлінням Kubernetes.

Оскільки Cloud Native передбачає розбір тісно пов'язаних процесів на слабко пов'язані мікросервіси, реархітектура додатків як CNF часто передбачає розбір функціональних аспектів на сервіси, що працюють в дискретних контейнерах або як окремі мікросервіси.

Операторам мобільного звязку, які збираються використовувати ці практики в широкому масштабі, буде надзвичайно важливо навчитися працювати з просторами імен і мультиорендування на рівні оркестрування. Розгортання розумної інфраструктури буде настільки ж важливим, як і розгортання CNF, якщо передбачається отримання прибутку від інвестицій.

Наприклад, якщо оператор повинен запустити кілька типів Kubernetes для різних CNF, то інфраструктура та необхідне програмне забезпечення для оркестрування більше не є вибором. За такого сценарію кожен набір CNF може потребувати більше головних серверів та ключ-значення etcd для екземплярів сховища, ніж це економічно доцільно, наприклад. Крім того, різні різновиди Kubernetes вимагатимуть різних операційних сценаріїв і потенційно ускладнять аналітику, моніторинг продуктивності додатків та управління інфраструктурою. Дивлячись на перспективу постачальників послуг, неможливо недооцінити, що центр обробки даних розміром з диван буде економічно ефективним, якщо очікується, що в цьому мініатюрному центрі обробки даних буде розміщено один екземпляр Kubernetes для кожного додатку. Оригінальна версія Kubernetes для телекомунікаційних компаній в ідеалі повинна працювати на всіх різних

компонентах інфраструктури, включаючи X86, ARM і RISC, через різний склад інфраструктури централізованих центрів обробки даних та електронних сервісів[28].

Стандартні організаці а також open-source організації та компанії, які представлені 3rd Generation Partnership Project (3GPP), European Telecommunications Standards Institute (ETSI), та Cloud Native Computing Foundation (CNCF) були визначені як основні компанії для просування Cloud Native в сфері телекомунікацій. Для незалежного масштабування та еволюції мережеві функції можна відокремити та абстрагувати за допомогою сервіс-орієнтованої архітектури (SBA) у 15-й версії 3GPP. У березні 2018 року CNCF офіційно випустила Kubernetes, систему управління контейнерами, яка була розроблена для автоматичного розгортання додатків, а також для кращого управління ресурсами та міжхостовими контейнерними кластерами. На основі існуючих стандартів і архітектур NFV ETSI представила Cloud Native для хмарних технологій телекомунікаційних компаній, щоб зробити можливим розгортання на основі контейнерів[22].

3.1 Telecom-хмара на основі Cloud Native від Huawei:

Huawei запропонували власне рішення - Telco Converged Cloud (TCC), щоб допомогти операторам адаптуватися до бізнес-вимог 5G, які постійно змінюються[26]. Це перше в галузі хмарне рішення наступного покоління на базі Cloud Native, продукт безперервних досліджень і розробок у сфері Cloud Native та великого досвіду трансформації хмарних технологій Huawei.

Як провідний постачальник ІКТ-інфраструктур у світі, Huawei активно сприяла розвитку екосистеми Cloud Native. Вона покращила паралельне

планування та об’єднання кластерів і можливості спільноти Docker, включаючи покращений контроль над процесором і пам’яттю. Дотепер Huawei вважалася найактивнішою в організаціях і спільнотах зі стандартизації серед усіх постачальників телекомунікаційного обладнання.

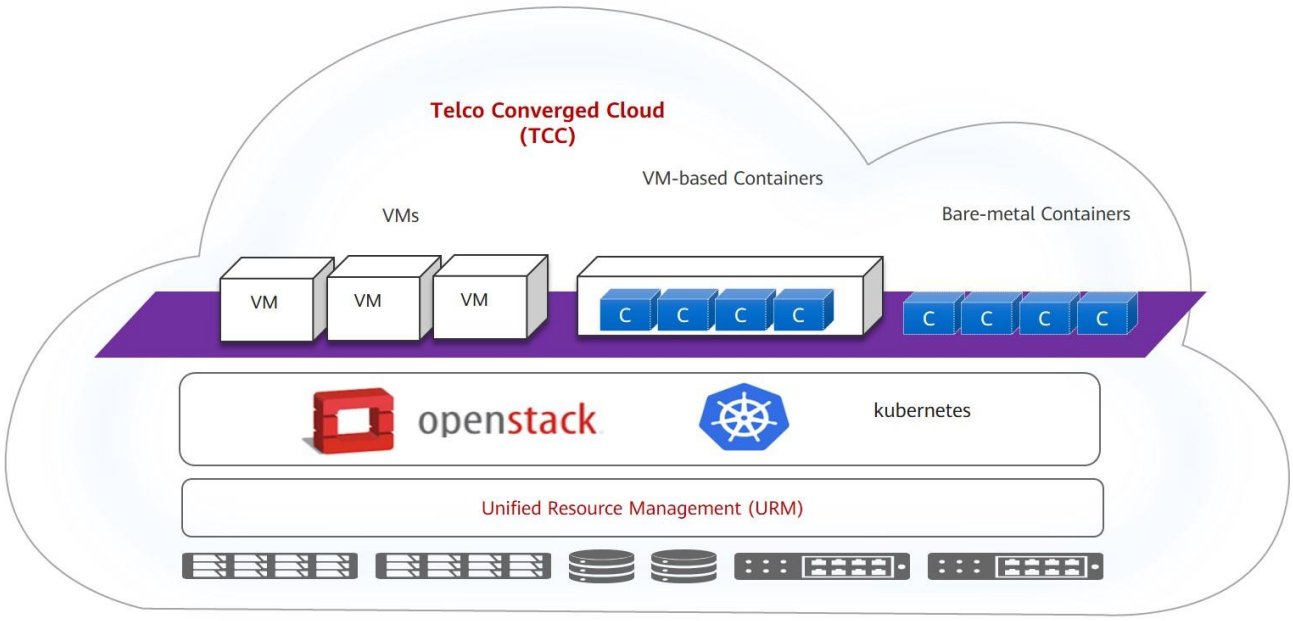

Рис.3.1. Архітектура Telco Converged Cloud

За допомогою Telco Converged Cloud (TCC) телеком-оператори можуть створити гнучку, швидку та надійну хмарну платформу для своїх мереж 5G. TCC використовує гіперконвергентну інфраструктуру, яка підтримує віртуальні машини та контейнери. Це дозволяє перейти від 4G до 5G і забезпечити уніфіковане управління ресурсами. Для підвищення ефективності, безпеки та продуктивності мережі TCC також забезпечує вдосконалення операторського класу, різноманітні обчислювальні можливості та автоматизацію експлуатації та обслуговування.

TCC було розроблено з урахуванням викликів та можливостей епохи 5G, такі як швидке впровадження інноваційних послуг, розгалуження мережі, периферійні обчислення та хмарні програми. Оператори можуть

використовувати TCC для прискорення хмарнізації своїх мереж, зниження загальної вартості володіння та підвищення конкурентоспроможності на ринку.

TCC - лідер у сфері рішень, який отримав багато нагород і визнання. На 5G World 2020 Summit12 TCC отримала нагороду «Краща ініціатива з віртуалізації мережі» за інноваційну дворівневу конвергенцію та легке розгортання. Багато операторів по всьому світу також використовують TCC для підтримки розгортання мереж 5G і надання послуг

Перечислимо переваги такого рішення:

Подвійна архітектура: Платформа TCC підтримує контейнери Kubernetes і віртуальні машини (BM) OpenStack. Крім того, TCC розробляє контейнерні додатки, щоб підтримувати інновації в епоху 5G на підприємствах B2B, B2C і B2H. Це також дозволяє існуючому VNF, розгорнутим на віртуальних машинах, продовжувати надавати послуги. TCC допомагає перевізникам відповідати вимогам ринку та досягати комерційного успіху завдяки великій гнучкості контейнерів.

Unified resource management (URM) об'єднує базові обчислювальні, мережеві та зберігальні ресурси та робить їх доступними для додатків верхнього рівня, таких як віртуальні машини та контейнери, на вимогу. Крім того, розгортання на основі URM потребує менших ресурсів, ніж незалежне розгортання пулу віртуальних машин або контейнерів.

Cloud Native забезпечує надійність і продуктивність операторського класу, що робить його в майбутній перспективі основною хмарою для телекомунікаційних компаній. Тим не менш, нативні додатки Kubernetes не відповідають стандартам продуктивності та надійності телекомунікаційної галузі. Щоб досягти надійності операторського класу, Huawei надає функції, такі як stateless design, N-шляхове резервування та A/B-тестування, щоб збільшити швидкість роботи додатків на базі Kubernetes. Ізоляція процесора, антиафінність

(Non-uniform memory access) NUMA та величезна пам'ять сторінок є кількома з цих технологій.

3.2 Google Cloud

Набір продуктів і рішень Google Cloud для телекомунікаційної сфери допомагає постачальникам послуг зв’язку (CSP) змінити свої IT, мережеві та периферійні системи за допомогою хмарних технологій. Оператори можуть створювати збільшити свої доходи, покращувати якість обслуговування клієнтів і оптимізувати операції за допомогою даних, штучного інтелекту (ШІ), 5G та периферійних обчислень, які доступні завдяки Google Cloud[27].

Ось деякі з основних телекомунікаційних рішень, які може запропонувати Google Cloud:

Google Distributed Cloud: це гібридна та мультихмарна платформа, яка розширює інфраструктуру та сервіси Google Cloud до периферії та центрів обробки даних CSP. Вона дозволяє CSP запускати Cloud-Native додатки та мережеві функції ближче до своїх клієнтів і пристроїв, залишаючись під ретельним наглядом і безпекою.

Google Workspace — це набір хмарних інструментів, які допомагають CSP розвивати та вести бізнес більш ефективним шляхом методом підвищення продуктивності та співпраці. Він містить програми, такі як Gmail, Документи, Диск, Зустрічі та Чат, серед інших.

Anthos for Telecom — це платформа, яка використовує контейнери та мікросервіси, щоб дозволити CSP модернізувати свої мережеві функції та додатки. Вона сприяє розгортанню, масштабуванню та автоматизації в гібридних і мультихмарних середовищах CSP.

BigQuery — це безсерверне сховище даних, яке дозволяє CSP швидко та легко аналізувати великі дані. Це може допомогти CSP знайти інформацію про поведінку клієнтів, продуктивність мережі, якість послуг тощо.

Для полеглшення труднощів CSP на шляху до хмарної трансформації, Google Cloud також співпрацює з різними постачальниками технологій, системними інтеграторами та постачальниками мережевого обладнання. У числі відомих партнерів є Nokia, Ericsson, AT&T, Vodafone, Orange і Telefonica. Крім того, Google Cloud надає сертифікацію та навчання фахівцям телекомунікаційного сектору, щоб вони могли навчитися та підтвердити свої навички роботи з хмарними технологіями[28].

3.3 Cloud Native Cloud рішення від Ericsson

Ericsson cloud native - це стратегія та набір рішень, які допомагають телекомунікаційним операторам впроваджувати хмарні технології для своїх мережевих додатків та послуг.

Ось деякі з переваг хмарних рішень Ericsson:

- Більш швидкі та автоматизовані оновлення та випуски мережевих додатків, що дозволяє підвищити операційну ефективність та скоротити час виходу на ринок.
- Більша масштабованість та інтероперабельність мережевих додатків, що дозволяє задовольнити зростаючий попит та різноманітність варіантів використання 5G.
- Підвищення рівня автоматизації та оркестрування мережевих функцій і послуг, що може зменшити складність і витрати.

Власна хмарна інфраструктура Ericsson: Це перевірена системою інфраструктура “Containers as a service” CaaS оптимізована для розміщення хмарних 5G-додатків. Вона усуває необхідність у рівні віртуалізації, що спрощує архітектуру та підвищує продуктивність.

Ericsson Cloud Native IMS: це рішення для IP-мультимедійної підсистеми (IMS), яке дозволяє розгортати функції IMS в контейнерному середовищі за

допомогою Kubernetes CaaS. Воно підтримує передачу голосу через LTE (VoLTE), передачу голосу через WiFi (VoWiFi), розширені комунікаційні послуги (RCS) тощо.

Ericsson Cloud Native Orchestration Center: Це новий центр у Північній Америці, який надає експертизу та підтримку телекомунікаційним операторам на шляху трансформації хмарних технологій. Він пропонує такі послуги, як консалтинг, проектування, тестування, інтеграція, розгортання та експлуатація хмарних рішень.

Висновки

Наразі існує декілька Cloud Native рішень для телеком операторів від великих гравців в IT індустрії. Але ще повинно пройти багато часу, щоб абсолютна більшість операторів по всьому світу перейшла на власно втілені технології Cloud Native в зв’язку з непродуманістю переходу, через який виникають проблеми. Через уже наявну структуру мережі з використанням VNF, які мають майже ті ж переваги, що і CNF, оператори не сильно квапляться з переходом через сумнівний процент переваг. Але в майбутньому, умови на ринку можуть змінитись і тоді перехід на технології Cloud Native буде обов'язковим для підтримання конкурентоспроміжності між гравцями на ринку.

Телекомунікаційні послуги мають високі стандарти продуктивності, безпеки та стійкості. Вони надають основні послуги, які забезпечують нашу безпеку та дозволяють нам виконувати повсякденну діяльність. Однак, ці стандарти мають високу ціну. Телекомунікаційна інфраструктура та додатки традиційно базувалися на тісно пов'язаних фізичних або віртуальних компонентах, які було важко модифікувати та розгортати. Це робило розвиток телекомунікацій дуже дорогим і повільним. Це також вимагало спеціальних знань про індивідуальні платформи.

Технології Cloud Native можуть прискорити розробку та вдосконалення телекомунікаційних додатків, дозволяючи їм працювати на товарній інфраструктурі та використовувати Kubernetes для модульності, легкої масштабованості, спостережуваності та сумісності. Це допоможе телекомунікаційним компаніям стати більш гнучкими, оперативними та адаптивними. Це також зменшить витрати та покращить якість послуг. Хмарні технології можуть дати телекомунікаційним компаніям перевагу над технологічними компаніями-початківцями, які використовують новіші технології, що базуються на програмному забезпеченні.

ЗАГАЛЬНІ ВИСНОВКИ ПО РОБОТІ

У дипломній роботі було досліджено існуючу архітектуру Cloud Native. Було описано історію створення, проведено детальний розбір складових цієї архітектури з підведенням їх переваг над класичними системами. Також було розглянуто сучасні мережі мобільних операторів, їх технологій та обговорено проблеми плавного переходу та теоретичні методи вирішення. Приведені приклади існуючих Cloud Native архітектур, запропонованих великими IT компаніями.

В результаті дослідження було з’ясовано, що технологія Cloud Native може значно покращити існуючі мережі мобільних операторів, полегшити побудову нових додатків для розробників, отримати можливість швидкого росту потужностей в разі росту потреб користувачів або додавання нових технологій та підвищити відмовостійкість своїх сервісів.

Телекомунікаційні компанії можуть використовувати технології Cloud Native для швидшого та якіснішого тестування, проектування, розгортання, надання, перегляду та впровадження своїх нових додатків та інновацій. Вони також можуть створювати більш стійку інфраструктуру та додатки, швидше та безпечніше впроваджувати нові функції та можливості.

Технології Cloud Native також можуть допомогти телекомунікаційним компаніям відповідати нормативним вимогам до критично важливих систем, таких як обробка екстрених викликів. Це пов'язано з тим, що додатки Cloud Native є більш відмовостійкими, швидкими та стійкими, ніж традиційні системи мобільних операторів.

Отже, підводячи підсумки можна сказати, що технології Cloud Native - це майбутнє телекомунікацій, і телекомунікаційні компанії повинні прийняти їх, щоб залишатися конкурентоспроможними та актуальними в майбутньому.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is Cloud Native? - Cloud Native Architecture Explained - AWS [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/what-is/cloud-native/

2. What is Cloud Native? | Microsoft Learn [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/architecture/cloud-native/definition

3. What is Cloud Native? | Oracle [Електронний ресурс] – Режим доступу до ресурсу: https://www.oracle.com/cloud/cloud-native/what-is-cloud-native/

4. Cloud Native | IBM [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/topics/cloud-native

5. Cloud Native Computing Foundation - Wikipedia [Електронний ресурс] – Режим доступу до ресурсу: [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Cloud_Native_Computing_Foundation

6. What is cloud native and why does it exist? | Cloud Native Computing Foundation [Електронний ресурс] – Режим доступу до ресурсу: https://www.cncf.io/online-programs/what-is-cloud-native-and-why-does-it-exist/

7. Getting Started with Kubernetes | A Brief History of Cloud-native Technologies - Alibaba Cloud Community [Електронний ресурс] – Режим доступу до ресурсу: https://www.alibabacloud.com/blog/getting-started-with-kubernetes-%7C-a-brief-history-of-cloud-native-technologies_595894

8. Kubernetes - Wikipedia [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Kubernetes

9. Who We Are | Cloud Native Computing Foundation [Електронний ресурс] – Режим доступу до ресурсу: https://www.cncf.io/about/who-we-are/

10. What is Cloud Native Computing Foundation (CNCF) - Definition from SearchITOperations [Електронний ресурс] – Режим доступу до ресурсу:

https://www.techtarget.com/searchitoperations/definition/Cloud-Native-Computing-Foundation-CNCF

11. Cloud-Native Computing: What It Is and Why Businesses Need It [Електронний ресурс] – Режим доступу до ресурсу: https://www.nutanix.com/theforecastbynutanix/technology/cloud-native-computing-what-it-is-and-why-businesses-need-it

12. 10 Key Benefits Of Cloud Native Application [2022 Update] [Електронний ресурс] – Режим доступу до ресурсу: https://www.toobler.com/blog/benefits-of-cloud-native-application

13. What are containers? | Google Cloud [Електронний ресурс] – Режим доступу до ресурсу: https://cloud.google.com/learn/what-are-containers

14. What is a container? | Microsoft Azure [Електронний ресурс] – Режим доступу до ресурсу: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-a-container/

15. Go Cloud Native with Azure Container Apps - Microsoft Community Hub [Електронний ресурс] – Режим доступу до ресурсу: https://techcommunity.microsoft.com/t5/apps-on-azure-blog/go-cloud-native-with-azure-container-apps/ba-p/3616407

16. Docker Docs: How to build, share, and run applications | Docker Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.docker.com/

17. Docker (software) - Wikipedia [Електронний ресурс] – Режим доступу до ресурсу: [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Docker_(software)

18. Concepts | Kubernetes [Електронний ресурс] – Режим доступу до ресурсу: https://kubernetes.io/docs/concepts/

19. What is CI/CD? [Електронний ресурс] – Режим доступу до ресурсу: [Електронний ресурс] – Режим доступу до ресурсу: https://www.redhat.com/en/topics/devops/what-is-ci-cd

20. Continuous Integration and Continuous Delivery (CI/CD) Fundamentals | GitHub Resources [Електронний ресурс] – Режим доступу до ресурсу: https://resources.github.com/ci-cd/

21. CI/CD - Wikipedia [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/CI/CD

22. The telecommunications industry embraces cloud-native engineering | Research & insight | Capgemini [Електронний ресурс] – Режим доступу до ресурсу: https://www.capgemini.com/insights/research-library/the-telecommunications-industry-embraces-cloud-native-engineering/

23. There's No Easy Evolution Path from VNF to CNF [Електронний ресурс] – Режим доступу до ресурсу: https://www.metaswitch.com/blog/theres-no-easy-evolution-path-from-vnf-to-cnf

24. Cloud native thinking for telecommunications GitHub [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/cncf/telecom-user-group/blob/master/whitepaper/cloud_native_thinking_for_telecommunications.md#1.5

25. What is a cloud-native network function (CNF)? [Електронний ресурс] – Режим доступу до ресурсу: [Електронний ресурс] – Режим доступу до ресурсу: https://www.techtarget.com/searchnetworking/definition/cloud-native-network-function-CNF

26. Next generation telco cloud based on cloud native [Електронний ресурс] – Режим доступу до ресурсу: https://carrier.huawei.com/en/industry-perspective/5g-core-network/next-generation-telco-cloud-based-on-cloud-native

27. What Is Cloud Native | Google Cloud [Електронний ресурс] – Режим доступу до ресурсу: https://cloud.google.com/learn/what-is-cloud-native

28. 5 principles for cloud-native architecture—what it is and how to master it | Google Cloud Blog [Електронний ресурс] – Режим доступу до ресурсу: https://cloud.google.com/blog/products/application-development/5-principles-for-cloud-native-architecture-what-it-is-and-how-to-master-it