Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)
Факультет інформаційних технологій
(факультет)
Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня ____магістра___
(бакалавра, спеціаліста, магістра)

студента _____Гулько Іллі Ігоровича_____
(ПІБ)
академічної групи _____123м-20-1_____
(шифр)
спеціальності _____123 «Комп'ютерна інженерія»_____
(код і назва спеціальності)
за освітньо-професійною програмою _____«Комп'ютерна інженерія»_____
(офіційна назва)
на тему «Адаптація комп'ютерної системи під технології контейнеризації на основі хмарної інфраструктури»_____
(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	проф. Гнатушенко В.В.			
розділів:				
технічні вимоги до системи	проф. Гнатушенко В.В.			
розроблення програмного забезпечення	проф. Гнатушенко В.В.			

Рецензент				

Нормоконтролер	проф. Цвіркун Л.І.			

Дніпро
2022

ЗАТВЕРДЖЕНО:

завідувач кафедри
інформаційних технологій
та комп'ютерної інженерії
(повна назва)

_____ Гнатушенко В.В.
(підпис) (прізвище, ініціали)
«___»_____ 202__ року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня магістра
(бакалавра, спеціаліста, магістра)

студенту Гулько І. І. **академічної групи** 123м-20-1
(прізвище та ініціали) (шифр)

спеціальності 123 «Комп'ютерна інженерія»

за освітньою-професійною програмою 123 «Комп'ютерна інженерія»
(офіційна назва)

на тему «Адаптація комп'ютерної системи під технології контейнеризації на основі хмарної інфраструктури»,

затверджену наказом ректора НТУ «Дніпровська політехніка» від 10.12.2021 р. №1036

Розділ	Зміст	Термін виконання
Стан питання та постановка завдання	На основі матеріалів виробничих практик, інших науково-технічних джерел сформулювати наукове завдання, конкретизувати предмет та мету досліджень	20.09.2021
Теоретичний розділ	Обґрунтувати теоретичну базу розв'язання наукового завдання, якому присвячено роботу	25.10.2021
Синтез системи	Розробка комп'ютерної системи	20.11.2021
Експериментальний розділ	Проведення експериментів і аналіз висновків	15.12.2022
Графічна частина	Графічні результати роботи подати у вигляді рисунків схем таблиць на 10 арк. формату А4.	10.01.2022

Завдання видано _____ проф. Гнатушенко В. В.
(підпис керівника) (прізвище, ініціали)

Дата видачі _____

Дата подання до екзаменаційної комісії _____

Прийнято до виконання _____ Гулько І. І.
(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 60 с., 28 рис., 9 пос., 1 додаток.

Мета: адаптація комп'ютерної системи під технології контейнеризації на основі хмарної інфраструктури.

Розробниками був створений додаток, який надає можливість бізнесу вести облік робочого часу і завдань співробітників. Для того, щоб цей додаток був безпечний і відмовостійкій для користувача, був використовувати трирівневий стандарт деплоя.

Використовуючи цю архітектуру можна здійснити найкращі практики безпеки та відмовостійкості для всієї інфраструктури.

Але з часом стало зрозуміло, що програма перестала відповідати жорстким вимогам продуктивності та економії фінансів та ресурсів у хмарному рішенні ABC. У зв'язку з чим було ухвалено рішення про перегляд архітектурного рішення у бік існуючих систем контейнеризації та оркестрації додатків.

Результати перевірки у вигляді таблиць, графіків описані і наводяться у пояснювальній записці або додатках.

Container, Docker, Kubernetes, Openshift, CRI, CNI, CoreOS, DevOps, Git, EKS, ECS, Fargate, Image, ContainerD, Namespace, Serverless, PaaS, IaC, Terraform, AWS, EC2, VPC.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Kubernetes - це система оркестрування контейнерів з відкритим вихідним кодом для автоматизації розгортання комп'ютерних програм, масштабування та управління. Спочатку він був розроблений Google, а зараз підтримується Cloud Native Computing Foundation (CNCF). Його мета – надати «платформу для автоматизації розгортання, масштабування та операцій контейнерних робочих навантажень». Він працює з рядом інструментів для роботи з контейнерами та запускає контейнери у кластері, часто з образами, створеними за допомогою Docker.

Docker - це набір продуктів "платформа як послуга" (PaaS), які використовують віртуалізацію на рівні ОС для доставки програмного забезпечення в пакетах, які називаються контейнерами. Контейнери ізольовані один від одного та поєднують власне програмне забезпечення, бібліотеки та файли конфігурації; вони можуть спілкуватись один з одним через чітко визначені канали. Оскільки всі контейнери спільно використовують служби одного ядра операційної системи, вони використовують менше ресурсів, аніж віртуальні машини.

Платформа як послуга (PaaS) - модель хмарних обчислень, за якої споживач отримує доступ до використання платформ інформаційних технологій: операційних систем, систем управління базами даних, відповідного програмного забезпечення, інструментів для розробки та тестування розміщених у хмарних провайдерів.

Інфраструктура як код (IaC) - це спосіб забезпечення та управління обчислювальними та мережевими ресурсами, описуючи їх у вигляді програмного

коду, на відміну від налаштування необхідного обладнання самостійно або використання інтерактивних інструментів. Прикладом IaC може бути Terraform.

Amazon Web Services (AWS) - дочірня компанією Amazon, яка надає платформу хмарних обчислень і API-інтерфейси на вимогу для приватних осіб, компаній і урядів на основі принципу «оплата за фактом».

ВСТУП

Останнім часом виник підвищений попит на оптимізацію обчислювальних потужностей. У зв'язку з цим ведеться активна розробка та вивчення альтернативних методів агрегації та оркестрування комп'ютерних систем.

Паралельно з програмним рівнем IT сфери розвиваються і перетворюються обчислювальне обладнання. З даним фактом збільшується як потужності, так і витрати на їх утримання.

Поступово, індустрія почала зсуватися від on-premise рішень в сторону централізованих обчислювальних платформ, в яких відповідальність за обладнання і безпеку несе власник дата центру. Така модель розподіленої відповідальності дозволяє знизити кількість людино-годин і фінансові витрати, які йдуть на утримання обчислювальної інфраструктури.

Також, разом з розвитком хмарних інструментів почали з'являтися окремі підходи до розробки інфраструктури: тепер інструменти автоматизації та оркестрування стали більш затребувані.

На даний момент основними типами обчислювальних систем є: Bare Metal, Віртуалізація, Контейнеризація та Serverless.

1 СТАН ПИТАННЯ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Стислий опис системи, яка потребує оптимізації

Компанія «ModernDesign» має проблеми з надмірним навантаженням на працівників, хаосу серед поставлених задач та відсутній можливості слідкувати за прогресом їх виконання.

Для основних обчислювальних потужностей використовується сервіс EC2 (Elastic Compute Cloud) - це веб-сервіс, який забезпечує безпечну обчислювальну ємність із змінним розміром в хмарі. Amazon EC2 пропонує обчислювальну платформу з можливістю вибору процесора, сховища, мережі, операційної системи та моделі покупки.

Основною базою даних для додатка є PostgreSQL. Для того, щоб розмістити цю базу даних в AWS можна використовувати RDS (Relational Database Service). Це спрощує налаштування, експлуатацію і масштабування реляційної бази даних. RDS забезпечує економічну і змінну ємність при автоматизації трудомістких адміністративних завдань, таких як підготовка обладнання, настройка бази даних, виправлення і резервне копіювання.

Віртуальні машини EC2 запущені в ізольованій VPC (Virtual Private Cloud). В VPC інстанси розподілені по підмережам, що дає можливість налаштувати роутинг між усіма компонентами.

1.1.1 Опис архітектури системи

Хмарна архітектура визначає технологічні компоненти, які поєднуються для створення хмари, в якій ресурси об'єднуються за допомогою технології віртуалізації та спільно використовуються в мережі. Компонентами хмарної архітектури є:

1. Зовнішня платформа (клієнт або пристрій, що використовується для доступу до хмари)
2. Одна або кілька серверних платформ (сервери та сховище)
3. Хмарна методологія доставки
4. Мережа для підключення хмарних клієнтів, серверів та сховища

Разом ці технології створюють архітектуру хмарних обчислень, на якій можуть працювати програми, надаючи кінцевим користувачам можливість використовувати потужність хмарних ресурсів. Хоча термін «хмарні обчислення» є відносно новим (21 століття), концепція хмарних обчислень дуже схожа на обчислення на мейнфреймах, популярні з 1960-х років, коли централізовані сервери запускали додатки, які використовували термінали, підключені до приватної мережі.

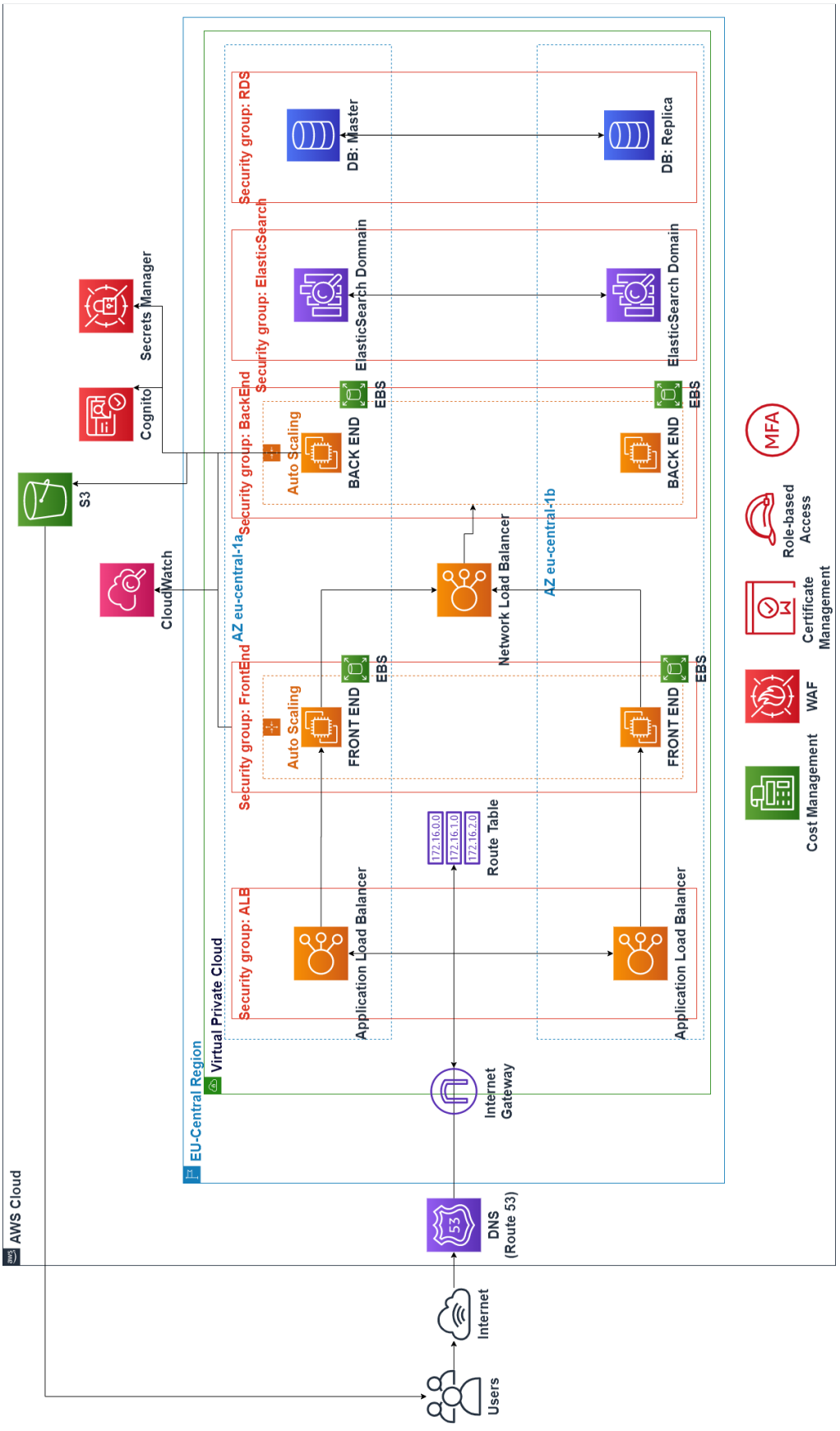

Рисунок 1.1 – Схема апаратного налаштування інфраструктури

1.1.2 Мережа комп’ютерної системи

Грунтом для мережевого налаштування в Amazon Web Services є Virtual Private Cloud. VPC веде себе як традиційна мережа TCP/IP, яку можна розширювати та масштабувати в міру необхідності. Однак компоненти DC, такі як маршрутизатори, комутатори, VLANS і т. д., явно не існують у VPC. Вони абстраговані та перетворені на хмарне програмне забезпечення.

Використовуючи VPC, можна швидко розгорнути віртуальну мережну інфраструктуру, в яку можуть бути запущені ресурси AWS. Кожен VPC визначає, що потрібно вашим ресурсам AWS, зокрема: IP-адреси, Підмережі, Маршрутизація, Безпека мережі.

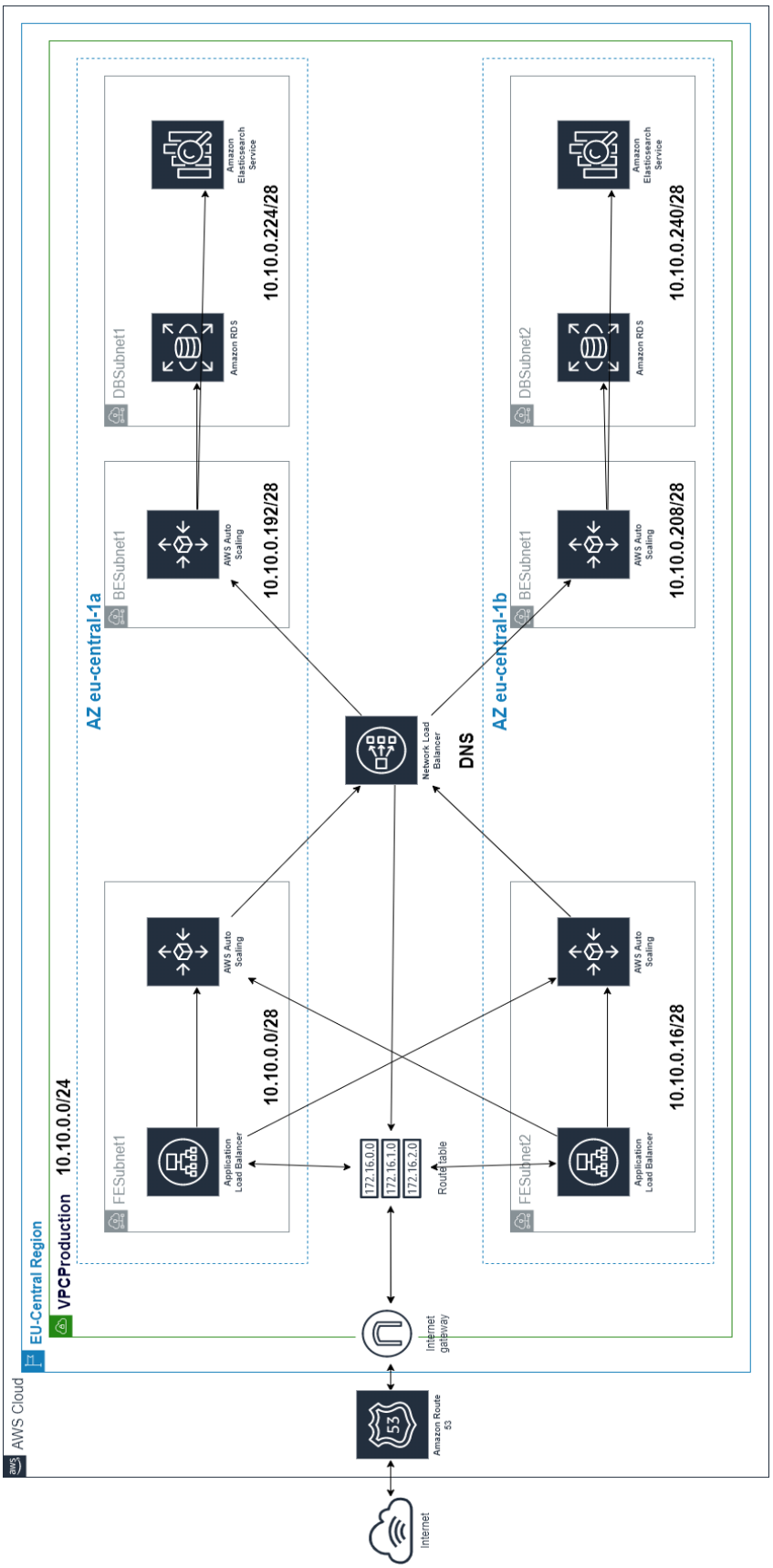

Рисунок 1.2 – Схема мережевого налаштування інфраструктури

2 АНАЛІЗ ОБЧИСЛЮВАЛЬНИХ РЕСУРСІВ ДЛЯ КОМП'ЮТЕРНОЇ СИСТЕМИ

Для подальшого аналізу методів оптимізації системи розглянемо типи обчислювальних ресурсів для комп'ютерної системи.

2.1 Bare Metal

Bare metal сервер – це фізичний комп'ютерний сервер, який використовується лише одним споживачем чи орендарем. Кожен сервер, пропонований в оренду, є окремим фізичним елементом обладнання, який сам по собі є функціональним сервером. Це не віртуальні сервери, які працюють на кількох загальних апаратних засобах.

Цей термін використовується для розрізнення серверів, на яких може розміщуватись декілька клієнтів і які використовують віртуалізацію та хмарний хостинг. Кожен bare metal сервер може виконувати будь-який обсяг роботи для користувача або мати кілька одночасних користувачів, але вони повністю присвячені тому, хто їх орендує.

Колись усі сервери були Bare Metal серверами. Сервери розташовувалися локально і часто належали організації, яка їх використовувала та експлуатувала. Операційні системи були розроблені дуже рано (початок 1960-х років), щоб забезпечити розподіл часу. Окремі великі комп'ютери, мейнфрейми чи міні-комп'ютери зазвичай розміщувалися в централізованих місцях. Перехід на дешеві ПК у 1980-х роках змінив ситуацію, оскільки ринок розширився, і більшість організацій почали купувати або здавати в оренду свої власні комп'ютери. Зростання Інтернету у 1990-х роках стимулювало практику розміщення в центрах обробки даних, де багато клієнтів спільно використовували можливості окремих серверів.

В даний час невеликі веб-сервери часто коштують дорожче за підключення, ніж вартість обладнання, що сприяє централізації. Можливість віртуального хостингу також спростила спільне розміщення багатьох веб-сайтів на одному сервері.

2.2 Віртуалізація

Віртуалізація - це процес створення віртуальної (а не фактичної) версії будь-чого, включаючи апаратні платформи віртуальних комп'ютерів, пристрої зберігання та ресурси комп'ютерної мережі.

Віртуалізація почалася у 1960-х роках як метод логічного поділу системних ресурсів, що надаються мейнфреймами, між різними програмами. З того часу значення цього терміна розширилося.

Віртуалізація устаткування - це віртуалізація комп'ютерів як повних апаратних платформ, певних логічних абстракцій їх компонентів чи лише функцій, які необхідні для запуску операційних систем. Віртуалізація приховує фізичні характеристики обчислювальної платформи від користувачів, представляючи натомість абстрактну обчислювальну платформу. Спочатку програмне забезпечення, що управляє віртуалізацією, називалося «керуючою програмою», але згодом перевагу стали віддавати термінам «гіпервізор» або «монітор віртуальних машин».

2.3 Контейнеризація

Контейнеризація - це упаковка програмного коду лише з бібліотеками операційної системи (ОС) та залежностями, необхідними для запуску коду для створення єдиного полегшеного файлу, що називається контейнером, який

узгоджено працює в будь-якій інфраструктурі. Контейнери, більш портативні та ресурсоефективні, ніж віртуальні машини, стали фактично обчислювальними одиницями сучасних хмарних додатків.

Контейнеризація дозволяє розробникам створювати та розгортати програми швидше та безпечніше. При використанні традиційних методів код розробляється у певному обчислювальному середовищі, яке при перенесенні в нове місце часто призводить до помилок. Наприклад, коли розробник переносить код з настільного комп'ютера на віртуальну машину. Контейнеризація усуває цю проблему, поєднуючи код програми разом із пов'язаними файлами конфігурації, бібліотеками та залежностями, необхідними для його роботи. Цей єдиний пакет програмного забезпечення або «контейнер» абстрагується від операційної системи хоста і, отже, стає автономним і переносимим – здатним без проблем працювати на будь-якій платформі чи хмарі.

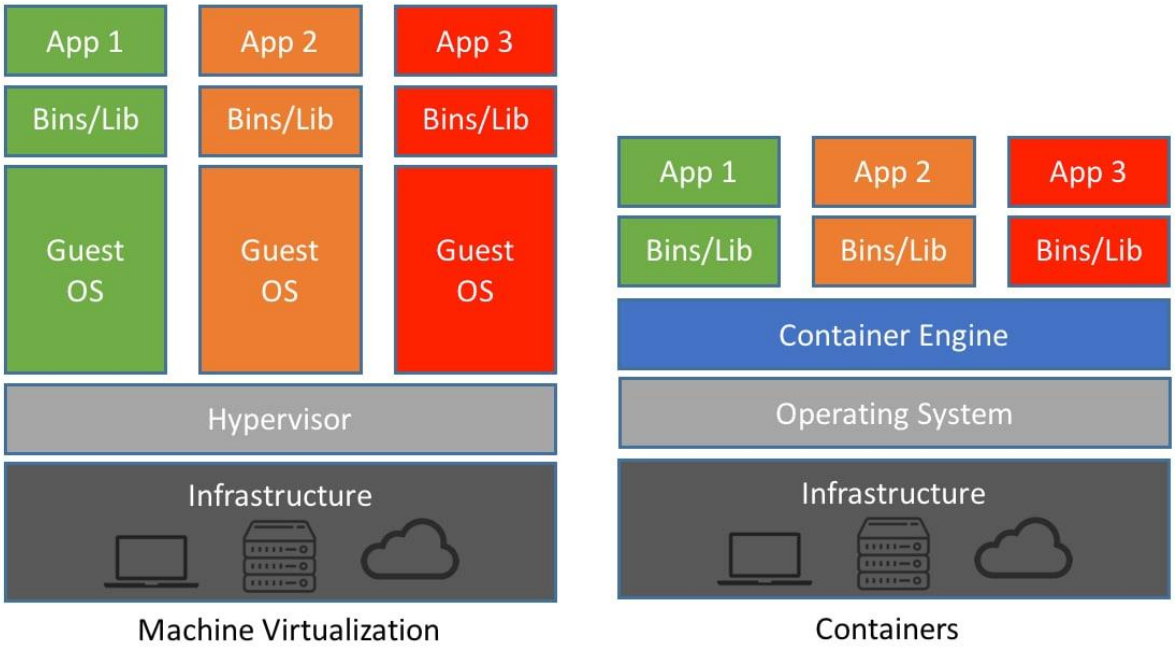

Рисунок 2.1 - Порівняння архітектури віртуальної машини та контейнеру

2.3.1 Одиниця контейнеризації

Контейнер - це стандартна одиниця програмного забезпечення, яка упаковує код і всі його залежності. Образ контейнера Docker - це автономний виконуваний пакет програмного забезпечення, який включає необхідне для запуску програми: код, середовище виконання, системні інструменти, системні бібліотеки та налаштування.

Образи контейнерів стають контейнерами під час виконання. Контейнерне програмне забезпечення, доступне як для додатків Linux, так і для Windows, завжди працюватиме однаково, незалежно від інфраструктури.

Контейнери не є об'єктами першого класу в ядрі Linux. Контейнери здебільшого складаються з кількох базових примітивів ядра:

1. Namespace (з ким дозволено комунікувати)
2. cgroups (кількість ресурсів, які дозволено використовувати)
3. LSM (Linux Security Modules – що саме дозволено робити)

Також, контейнери потребують Container Runtime, або Engine.

Container Runtime - це частина програмного забезпечення, яке приймає запити користувачів, включаючи параметри командного рядка, витягує image і з погляду кінцевого користувача запускає контейнер. Існує безліч Container Runtime, наприклад: docker, RKT, CRI-O та LXD. Крім того, багато хмарних провайдерів, платформи як послуга (PaaS) і платформи контейнерів мають власні вбудовані Container Runtime, які використовують образи контейнерів сумісні з Docker або OCI. Наявність стандартного галузевого формату образу контейнера забезпечує взаємодію між усіма різними платформами.

OCI (Open Container Initiative) розробляє runc, середовище виконання контейнерів, яке реалізує їх специфікацію і є основою інших інструментів вищого рівня.

2.3.2 Оркестратори

Контейнерне оркестрування - це автоматизація більшості операційних зусиль, необхідні запуску контейнерних робочих навантажень і сервісів. Це включає широкий спектр речей, які необхідні програмним командам для управління життєвим циклом контейнера, включаючи підготовку, розгортання, масштабування (вгору і вниз), мережеву взаємодію, балансування навантаження та багато іншого.

Оркестрування контейнерів є ключем до роботи з контейнерами, та також пропонує власні переваги для контейнерного середовища, в тому числі:

- Спрощені операції - це найважливіша перевага оркестрування контейнерів та основна причина її прийняття. Контейнери створюють велику складність, яка може швидко вийти з-під контролю без оркестрування контейнерів для керування ними.
- Стійкість - інструменти оркестрації контейнерів можуть автоматично перезапускати або масштабувати контейнер або кластер, збільшуючи стійкість.
- Додаткова безпека - автоматизований підхід до оркестрування контейнерів допомагає забезпечувати безпеку контейнерних програм за рахунок зменшення або усунення ймовірності людської помилки.

На ринку існує достатньо оркестраторів, наприклад OpenShift, Kubernetes, Docker Compose, Minikube, Rancher та інші.

2.4 Serverless

Serverless, або безсерверні обчислення - це модель виконання хмарних обчислень, в якій постачальник хмарних послуг виділяє машинні ресурси за запитом. Термін "безсерверний" невірний у тому сенсі, що сервери, як і раніше, використовуються постачальниками хмарних послуг для виконання коду для розробників. Однак планування ємності, конфігурація, керування, обслуговування, стійкість до відмов або масштабування контейнерів, віртуальних машин або фізичних серверів не стосується розробників програмного забезпечення.

Безсерверні обчислення не утримують ресурси в енергозалежній пам'яті; обчислення виконуються короткими пакетами зі збереженням результатів до сховища. Коли програма не використовується, їй не виділяються обчислювальні ресурси. Ціни ґрунтуються на фактичній кількості ресурсів, що споживаються додатком.

Serverless постачальники пропонують обчислювальні середовища виконання, також відомі як платформи «функція як послуга» (FaaS), які виконують логіку програми, але не зберігають дані. Такими середовищами виконання підтримуються загальні мови програмування, таки як Java, NodeJS, Python, Golang. Як правило, функції виконуються в межах ізоляції, наприклад, у контейнерах Linux.

2.5 Порівняння типів обчислювальних ресурсів для комп'ютерної системи

На підставі проведених досліджень було визначено, що використання Bare Metal серверів є необґрунтованим, оскільки інфраструктура вже знаходиться у хмарному середовищі та віртуалізоване за допомогою сервісу EC2.

На даний момент програма вимагає оптимізації ресурсів з боку апаратної частини, що не дозволяє нам продовжувати використовувати сервіси віртуалізації обчислювальних систем.

Також, використання серверлес платформ неможливе через тип програми, яка вимагає постійного та стабільного існування.

Таким чином, найоптимальнішим вибором є контейнеризація програми під певний контейнер runtime з подальшим використанням оркестратора.

3 ПОРІВНЯННЯ МЕТОДІВ КОНТЕЙНЕРИЗАЦІЇ

Контейнерна технологія народилася у 1979 році з версією 7 Unix та системою chroot. Chroot ізолює процес, обмежуючи доступ програми до певного каталогу, цей каталог складається з кореневого та дочірнього каталогів. Ця система була першим проблиском ізольованого процесу, і незабаром вона була прийнята і додана до ОС BSD у 1982 році; проте контейнерні технології, на жаль, не будуть розвиватися протягом наступних двох десятиліть і залишаться бездіяльними.

Технологія контейнерів нарешті набрала обертів у 2000-х з появою Free BSD Jails. «Jails» - це розділи комп'ютера, на яких в одній системі може бути кілька осередків/розділів. Ця система була додатково удосконалена в 2001 році за допомогою Linux VServer з поділом ресурсів, який пізніше був пов'язаний з ядром Linux за допомогою OpenVZ в 2005 році, а в 2004 були об'єднані з поділом кордонів

для створення в контейнерах Solaris. подальший прогрес було досягнуто із запровадженням контрольних груп у 2006 році.

Групи управління або контрольні групи були реалізовані для обліку та ізоляції використання таких ресурсів, як ЦП та пам'ять. Незабаром вони були використані та побудовані в LXC (контейнери LinuX) у 2008 році, і це була найбільш повна та стабільна версія контейнерної технології в той час, оскільки вона працювала на ядрі Linux без будь-яких виправлень. Завдяки надійності та стабільності LXC, багато інших технологій, засновані на LXC, першою з яких стала Warden у 2011 році та, що більш важливо, Docker у 2013 році.

Незважаючи на те, що в період з 2000 по 2011 рік контейнерна технологія значно покращилася, впровадження Docker змінило все і поодинці призвело до відродження контейнерної технології. Docker побудував свою основу на двох системах: LXC і libcontainers. Libcontainers походять від LMCTFY, який був стек контейнерів з відкритим вихідним кодом, в якому програми створювали свої власні підконтейнери і керували ними. На додаток до попереднього програмного забезпечення, Docker мав простий у використанні графічний інтерфейс і дозволяв запускати кілька програм з різними вимогами до ОС на одній ОС. Через ці феноменальні якості Docker вибухнув, що призвело до 100 мільйонів завантажень протягом року, а після успіху Docker інша технологія, rkt (вимовляється як Rocket), спробувала вирішити деякі проблеми Docker, включивши суворіші вимоги до безпеки та виробництва.

Вплив та популярність обох цих технологій зростали в міру їхньої взаємодії з Cloud Native Computing Foundation, глобальним центром розробників, фондів та постачальників. Це стимулювало участь спільноти та співробітництво, яке продовжувало зростати після того, як Microsoft дозволила запускати контейнери Linux (включаючи rkt та Docker) на комп'ютерах з Windows у 2016 році. Раніше контейнери були в основному технологією на основі Linux, але це рішення призвело

до того, що Microsoft стала основним впливом на технологію контейнерів, що спостерігається як у контейнерах Process, так і в контейнерах Hyper-V на комп'ютерах з Windows.

Розвиток контейнерних технологій продовжився у 2017 році завдяки впровадженню Kubernetes, високоефективної технології оркестрування контейнерів. Після включення Kubernetes у хмарних провайдерів, таких як CNCF, та його підтримки з боку Docker, він став галузевим стандартом, а поєднання Kubernetes та інших контейнерних інструментів, таких як Flannel, призвело до подальших покращень у контейнерних технологіях. У 2019 році все ще спостерігається золотий вік контейнерних технологій, і він продовжує досягати мас через Containerd, Containership та багато інших, що призводить до постійної популярності та прогресу в індустрії контейнерних технологій.

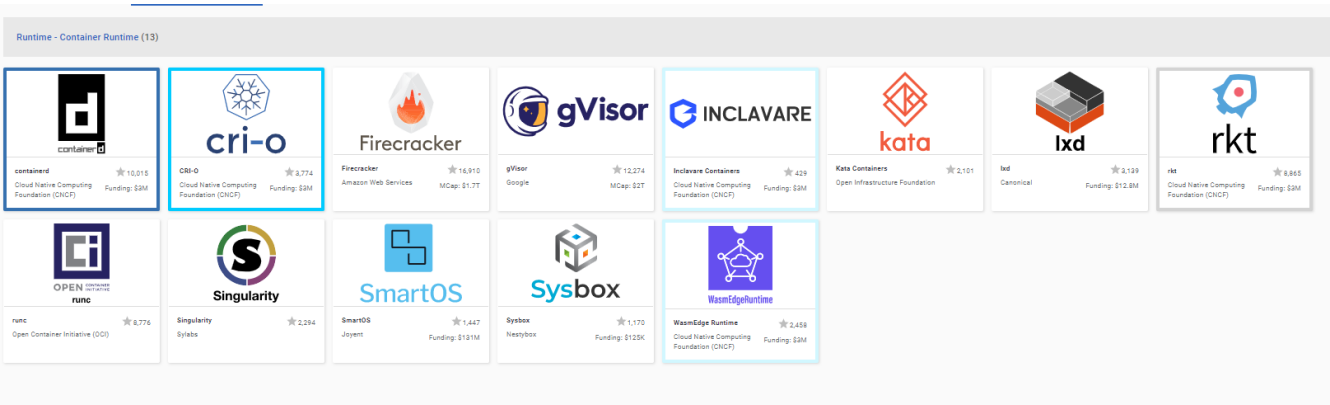

Рисунок 3.1 - Список CNCF Container Runtimes

3.1 OCI vs CRI

Open Container Initiative (OCI) - це проект Linux Foundation, започаткований Docker у червні 2015 року з метою розробки відкритих стандартів для віртуалізації на рівні операційної системи (контейнерів), насамперед контейнерів Linux. В даний час розробляються і використовуються дві специфікації: runtime-spec та image-spec. OCI розробляє runc, середовище виконання контейнерів, яке реалізує їх

специфікацію і є основою інших інструментів вищого рівня. runC був вперше випущений у липні 2015 як версія 0.0.1.

CRI (Container Runtime Interface) складається зі специфікацій / вимог (які будуть додані), protobuf API та бібліотек для середовищ виконання контейнерів для інтеграції з kubelet на вузлі. CRI API зараз знаходиться в альфа-версії, а інтеграція CRI-Docker використовується за замовчуванням, починаючи з Kubernetes 1.7+.

До появи CRI середовища виконання контейнерів (наприклад, docker, rkt) були інтегровані з kubelet за допомогою реалізації внутрішнього високорівневого інтерфейсу kubelet. Вхідний бар'єр для середовища виконання був високий, тому що для інтеграції потрібно було розуміння внутрішнього пристрою kubelet і участь переважно репозиторії Kubernetes. Що ще важливіше, це не буде масштабуватися, тому що кожне нове додавання спричиняє значні накладні витрати на обслуговування переважно репозиторії Kubernetes.

Kubernetes прагне бути розширюваним. CRI - це невеликий, але важливий крок до включення середовища виконання контейнерів, що підключаються, і створення більш здорової екосистеми.

3.2 Container Engine

Container Engine - це частина програмного забезпечення, яке приймає запити користувача, включаючи параметри командного рядка, витягує зображення і з точки зору кінцевого користувача запускає контейнер. Існує безліч контейнерних двигунів, включаючи docker, RKT, CRI-O та LXD. Крім того, багато хмарних провайдерів, платформи як послуга (PaaS) і платформи контейнерів мають власні вбудовані двигуни контейнерів, які використовують образи контейнерів, сумісні з Docker або OCI. Наявність стандартного галузевого формату образу контейнера забезпечує можливість взаємодії між усіма різними платформами.

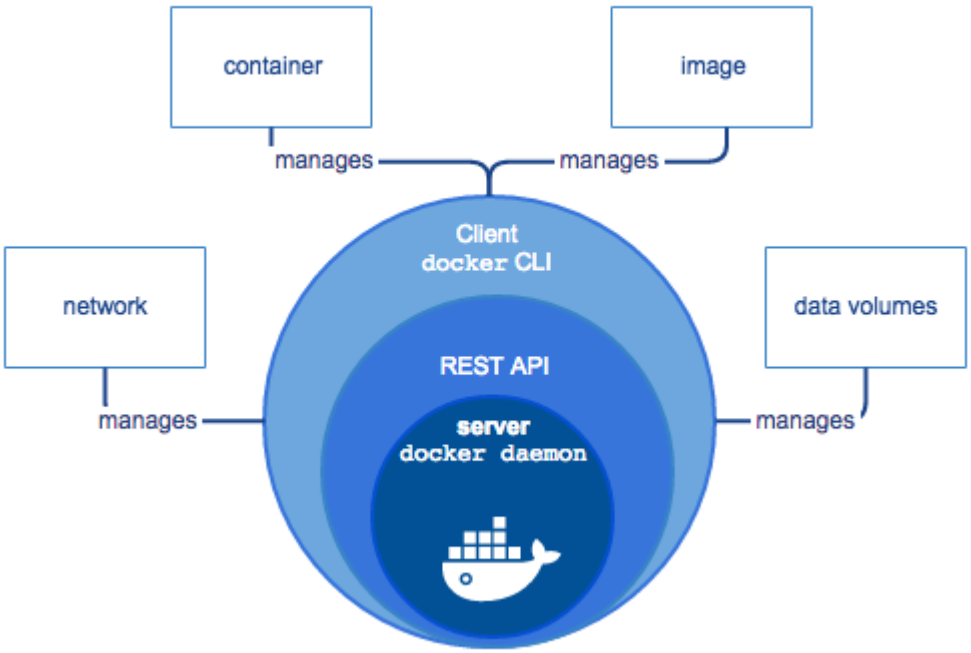

Рисунок 3.2 - Архітектура Docker Engine

Якщо піти на один рівень глибше, то більшість Container Engine фактично не запускають контейнери, вони покладаються на середовище виконання, сумісне з OCI, таку як runc. Зазвичай двигун контейнера відповідає за:

- Обробка user input
- Обробка введення через API з оркестратора контейнерів
- Отримання образів контейнерів із сервера реєстру
- Розпакування та розширення образу контейнера на диску за допомогою граф драйвера (блоку або файлу в залежності від драйвера)
- Підготовка точки монтування контейнера, зазвичай у сховище копіювання під час запису (знову ж таки, блок або файл залежно від драйвера)
- Підготовка метаданих, які будуть передані у середу виконання контейнера для правильного запуску контейнера.

- Використання деяких значень за замовчуванням з контейнера (наприклад, ArchX86)
- Використання введення користувача для перевизначення значень за умовчанням в образі контейнера (наприклад, CMD, ENTRYPOINT)
- Використання стандартних значень, вказаних в образі контейнера (наприклад, правила SECCOM)
- Виклик середовища виконання контейнера

3.3 Порівняння Container Runtimes

Container Runtime - компонент нижнього рівня, який зазвичай використовується в Container Engine, але також може використовуватися вручну для тестування. Еталонна реалізація стандарту середовища виконання Open Containers Initiative (OCI) – runc. Але є й інші середовища виконання, сумісні з OCI, такі як crun, railcar та katacontainers. Docker, CRI-O та багато інших двигунів контейнерів покладаються на runc.

Середовище виконання контейнера відповідає за:

- Використання точки монтування контейнера, наданої Container Engine (також може бути простим каталогом для тестування)
- Використання метаданих контейнера, наданих Container Engine (також може бути вручну створеним config.json для тестування)
- Взаємодія із ядром для запуску контейнерних процесів (системний виклик clone)
- Налаштування контрольних груп
- Налаштування політики SELinux
- Налаштування правил App Armor
- Щоб розповісти трохи про історію, коли двигун Docker

3.3.1 Docker

Docker - це набір продуктів "платформа як послуга" (PaaS), які використовують віртуалізацію на рівні ОС для доставки програмного забезпечення в пакетах, які називаються контейнерами. Контейнери ізольовані один від одного та поєднують власне програмне забезпечення, бібліотеки та файли конфігурації; вони можуть спілкуватися один з одним через чітко визначені канали. Оскільки всі контейнери спільно використовують служби одного ядра операційної системи, використовують менше ресурсів, ніж віртуальні машини.

Сервіс має як безкоштовні, так і преміальні рівні. Програмне забезпечення, на якому розміщуються контейнери, називається Docker Engine. Вперше він був запущений у 2013 році та розроблений Docker, Inc.

Docker може запакувати програму та її залежності у віртуальний контейнер, який може працювати на будь-якому комп'ютері під керуванням Linux, Windows або macOS. Це дозволяє програмі працювати в різних місцях, наприклад локально, у загальнодоступній або приватній хмарі. При роботі в Linux Docker використовує функції ізоляції ресурсів ядра Linux (такі як cgroups та простори імен ядра) та файлову систему з можливістю об'єднання (наприклад, OverlayFS), щоб дозволити контейнерам працювати в одному екземплярі Linux, уникаючи накладних витрат на запуск та обслуговування віртуальних машин . Docker у macOS використовує віртуальну машину Linux для запуску контейнерів.

Оскільки контейнери Docker мають мінімальну вагу, на одному сервері або віртуальній машині може одночасно працювати кілька контейнерів. Аналіз 2018 показав, що типовий варіант використання Docker включає запуск восьми контейнерів на хост, і що чверть проаналізованих організацій використовують 18 або більше контейнерів на хост.

Підтримка просторів імен ядром Linux в основному ізолює уявлення про операційне середовище, включаючи дерева процесів, мережу, ідентифікатори користувачів та змонтовані файлові системи, в той час як контрольні групи ядра забезпечують обмеження ресурсів для пам'яті та ЦП. Починаючи з версії 0.9, Docker включає власний компонент (званий "libcontainer") для використання засобів віртуалізації, що надаються безпосередньо ядром Linux, на додаток до використання абстрактних інтерфейсів віртуалізації через libvirt, LXC та systemd-nspawn.

Docker реалізує високорівневий API для надання полегшених контейнерів, які запускають процеси ізольовано. Контейнери Docker - це стандартні процеси, тому можна використовувати функції ядра для відстеження їх виконання, включаючи, наприклад, використання таких інструментів, як strace, спостереження та втручання у системні виклики.

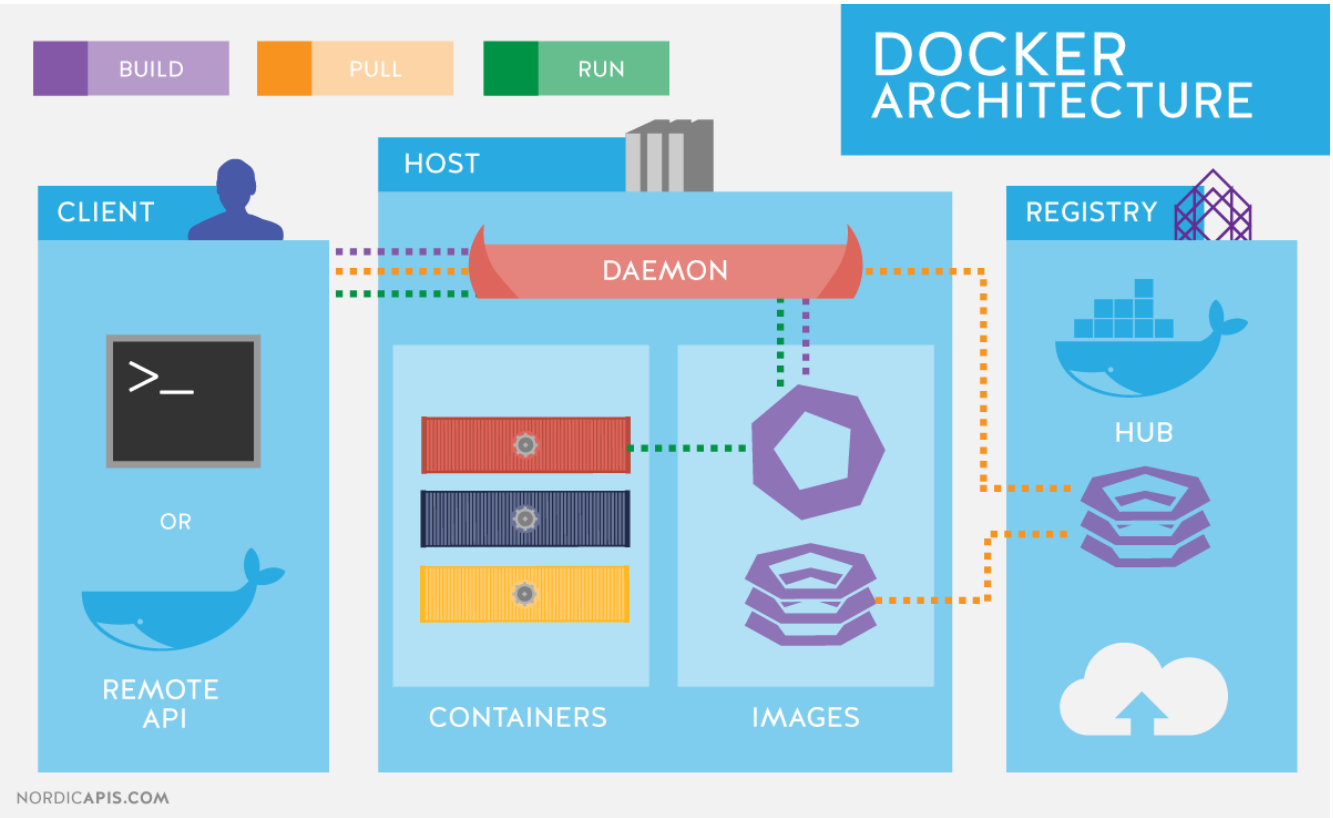

Рисунок 3.3 - Архітектура Docker

Програма Docker як послуга складається із трьох компонентів:

Програмне забезпечення: демон Docker, так званий dockerd, є постійним процесом, який управляє контейнерами Docker і обробляє об'єкти-контейнери. Демон прослуховує запити надіслані через Docker Engine API. Клієнтська програма Docker, яка називається docker, надає інтерфейс командного рядка (CLI), який дозволяє користувачам взаємодіяти з демонами Docker.

Об'єкти Docker: це різні об'єкти, які використовуються для складання програми в Docker. Основні класи об'єктів Docker – це зображення, контейнери та сервіси.

Контейнер Docker - це стандартизоване інкапсульоване середовище, в якому запускаються програми. Контейнер можна керувати за допомогою Docker API або інтерфейсу командного рядка.

Образ Docker - це шаблон для читання, який використовується для створення контейнерів. Зображення використовуються для зберігання та доставки програм.

Служба Docker дозволяє масштабувати контейнери між кількома демонами Docker. Результат відомий як Swarm, набір демонів, що взаємодіють, які спілкуються через Docker API.

Реєстри Docker: це репозиторій для образів Docker. Клієнти Docker підключаються до реєстрів для завантаження («витягування») образів для використання або завантаження («проштовхування») образів, які вони створили. Реєстри можуть бути публічними чи приватними. Два основні загальнодоступні реєстри - це Docker Hub і Docker Cloud. Docker Hub - це стандартний реєстр, в якому Docker шукає образи. Реєстри Docker також дозволяють створювати сповіщення на основі подій.

3.3.2 containerd

containerd - це колишня частина Docker, а нині самостійне рішення, що реалізує середовище для запуску контейнерів. При його створенні, як стверджують розробники, вони прагнули простоти, надійності та портованості.

Це демон на хост-системі, який керує всім життєвим циклом контейнера: від отримання та зберігання образу до запуску контейнера (через runC – докладніше див. нижче) та контролю його роботи. З демоном containerd можна взаємодіяти по низькорівневому gRPC API через локальний UNIX-сокет, а для експериментів та налагодження також доступна консольна утиліта ctr (вона також використовує gRPC API). Вихідний код написаний на Go та доступний на GitHub під ліцензією Apache License 2.0.

Основні примітиви, з якими працює containerd, - це bundles ("комплекти") та контейнери. Запуском контейнерів займається runC - утиліта, написана на Go, що використовує libcontainer і відокремлена від Docker у 2015 році. Вона працює відповідно до специфікації OCI Runtime Specification та запускає контейнери як свої дочірні процеси. Для запуску вимагає лише кореневу файлову систему та конфігурацію (решта: отримання образу, його розпакування тощо – залишається для неї «за кадром»).

Bundles містять конфігурацію, метадані та дані кореневої файлової системи. Вони є дисковим представленням запущеного контейнера (у найпростішому випадку — це звичайний каталог у ФС), яке можна переносити інші системи, упаковувати і поширювати. По суті, весь пристрій containerd полягає в тому, щоб координувати створення і запуск bundles.

Компоненти containerd утворюють такі підсистеми:

- Distribution - сервіс, що забезпечує отримання образів контейнерів.
- Bundle - сервіс, що дозволяє витягувати (і пакувати) bundles з дискових образів.
- Runtime - сервіс для запуску bundles (викликає runC для запуску контейнерів з переданими їм параметрами).

Кожна підсистема має чи кілька компонентів, реалізують її поведінку. Саме до сервісів, що надаються підсистемами, звертаються користувачі containerd через gRPC API.

Компоненти containerd, що працюють одночасно з різними підсистемами, називаються модулями:

- Executor реалізує безпосередній запуск контейнера.
- Supervisor, що контролює та відображає статус контейнера.
- Metadata зберігає метадані в графовій базі даних.
- Content, що надає доступ до адресованого сховища контенту (постійних даних).
- Snapshot, який управляє снапшотами файлової системи для образів контейнера. Аналог graphdriver у сьогоднішньому Docker. Шари розпаковуються у снапшоти.
- Events, що реалізує подієву поведінку та можливість аудиту.
- Metrics, забезпечує доступність (по API) метрик різних компонентів.

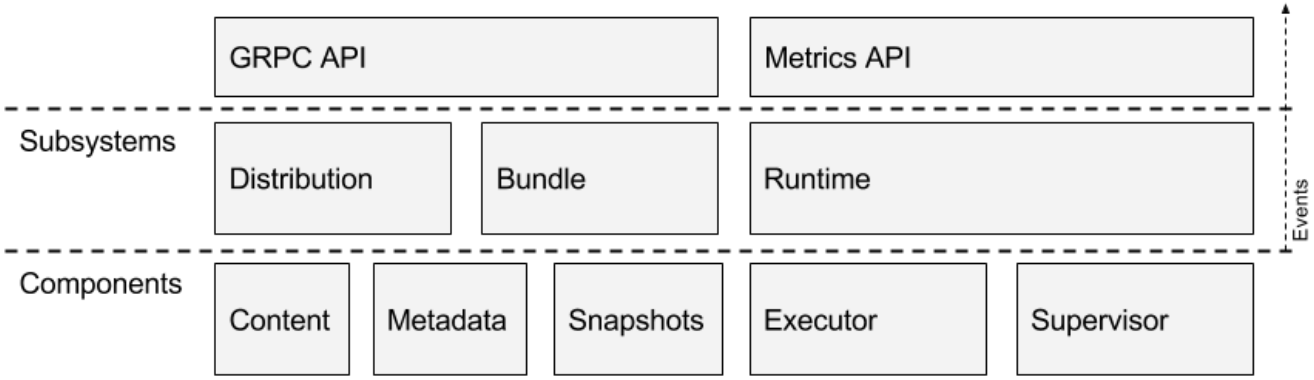

Рисунок 3.4 - Архітектура containerd

3.3.3 CRI-O

CRI-O - це реалізація Kubernetes CRI (Container Runtime Interface), що дозволяє використовувати середовища виконання, сумісні з OCI (Open Container Initiative). Це легка альтернатива використанню Docker як середовище виконання для кластерів. Це дозволяє Kubernetes використовувати будь-яке OCI-сумісне середовище виконання як середовище виконання контейнера для запуску модулів. Сьогодні він підтримує контейнери runc та Kata як середовища виконання контейнерів, але можна підключити будь-яке OCI-сумісне середовище виконання.

CRI-O підтримує образи контейнерів OCI та може вилучати з будь-якого реєстру контейнерів. Це легка альтернатива використанню Docker, Moby або rkt як середовище виконання для Kubernetes.

Архітектурні компоненти CRI-O:

- Kubernetes зв'язується з kubelet для запуску поду.
- Поди - це концепція, що складається з одного або декількох контейнерів, що використовують одні й ті самі простори імен IPC, NET і PID, що знаходяться в одній контрольній групі.

- Kubelet надсилає запит демону CRI-O через kubernetes CRI для запуску нового POD.
- CRI-O використовує бібліотеку контейнерів/зображень для отримання зображення з реєстру контейнерів.
- Завантажений образ розпаковується у кореневі файлові системи контейнера, зберігається у файлових системах COW, з використанням бібліотеки контейнерів/сховища.
- Після створення rootfs для контейнера CRI-O генерує json-файл специфікації виконання OCI, який описує, як запускати контейнер за допомогою інструментів OCI Generate.
- Потім CRI-O запускає OCI-сумісне середовище виконання, використовуючи специфікацію виконання процедур контейнера. Середовище виконання OCI за промовчанням – runc.
- Кожен контейнер контролюється окремим процесом conmon. Conmon-процес містить ідентифікатор PID1 контейнерного процесу. Він обробляє ведення журналу контейнера і записує код виходу для процесу контейнера.
- Мережа для модуля налаштовується за допомогою CNI, тому з CRI-O можна використовувати будь-який модуль CNI, що підключається.

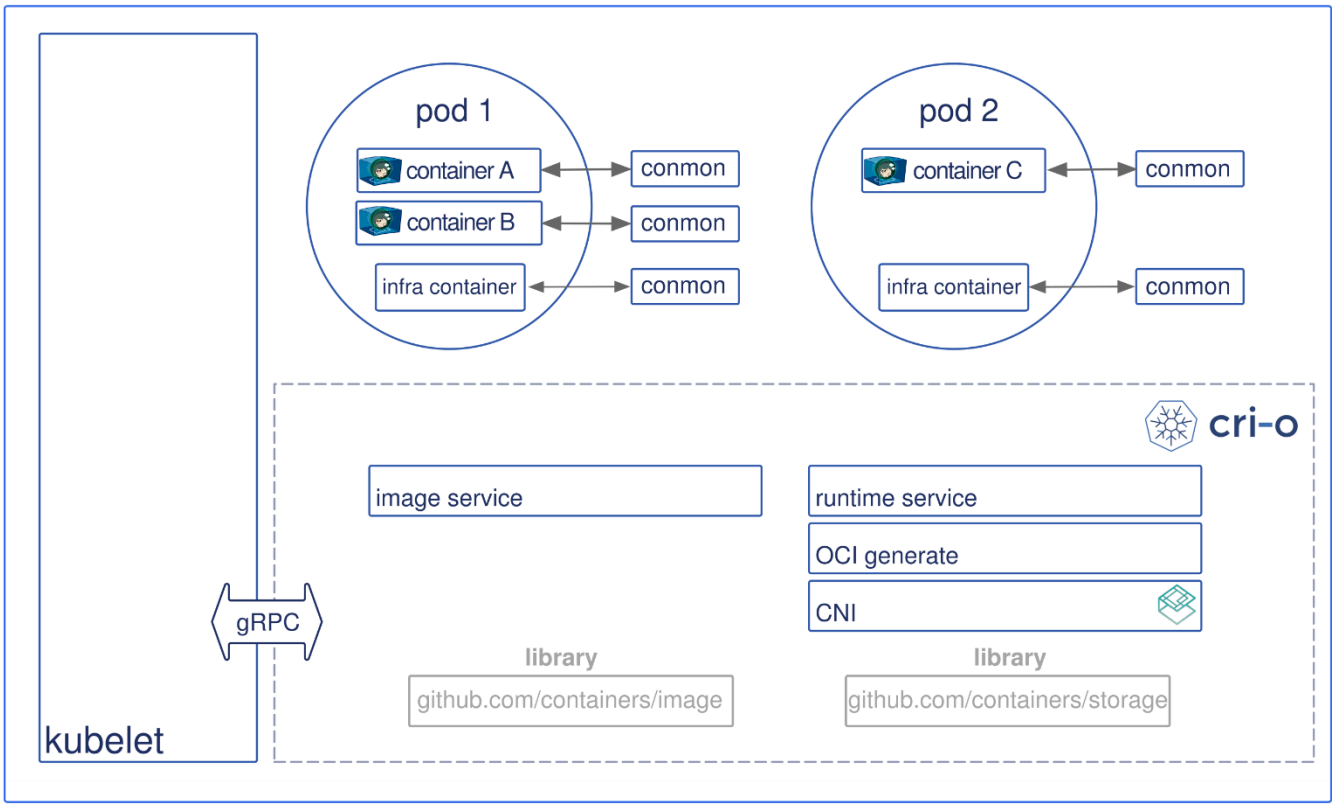

Рисунок 3.5 - Архітектура CRI-O

3.3.4 rkt

rkt – це механізм контейнеризації додатків, розроблений для сучасних виробничих хмарних середовищ. Він має власний підхід, середовище виконання і чітко визначену область, що робить його ідеальним для інтеграції з іншими системами.

Основною виконавчою одиницею rkt є модуль, набір з одного або кількох додатків, що виконуються у загальному контексті (модулі rkt є синонімами концепції у системі оркестрування Kubernetes). rkt дозволяє користувачам застосовувати різні конфігурації (наприклад, параметри ізоляції) як на рівні подів, так і більш детальному рівні додатків. Архітектура rkt означає, що кожен модуль виконується безпосередньо в класичній моделі процесу Unix (тобто відсутня центральний демон) в автономному ізольованому середовищі. rkt реалізує сучасний

відкритий стандартний формат контейнера, специфікацію контейнера додатків (appc), але може виконувати інші образи контейнерів, наприклад створені за допомогою Docker.

З моменту свого впровадження CoreOS у грудні 2014 року проект rkt значно розвинувся і набув широкого поширення. Він доступний для більшості основних Linux-дистрибутивів, і кожен випуск rkt створює автономні пакети rpm/deb, які користувачі можуть встановити. Ці пакети також доступні як частина репозиторію Kubernetes для тестування інтеграції rkt + Kubernetes. rkt також відіграє центральну роль у тому, як Google Container Image та CoreOS Container Linux запускають Kubernetes.

Цей проект завершено, і всі заходи щодо розробки/обслуговування зупинені.

3.3.5 LXC

Контейнери Linux (LXC) – це метод віртуалізації на рівні операційної системи для запуску кількох ізольованих систем Linux на керуючому хості з використанням одного ядра Linux.

Ядро Linux надає функції cgroups, які дозволяють обмежувати та визначати пріоритети ресурсів (ЦП, пам'ять, блокове введення-виведення, мережа тощо) без необхідності запуску будь-яких віртуальних машин, а також функціональність ізоляції простору імен, що дозволяє повністю ізолювати представлення програми про операційне середовище, включаючи дерева процесів, мережі, ідентифікатори користувачів та змонтовані файлові системи.

LXC поєднує в собі контрольні групи ядра та підтримку ізольованих просторів імен, щоб забезпечити ізольоване середовище для додатків. Ранні версії Docker використовували LXC як драйвер виконання контейнера, хоча LXC був

зроблений необов'язковим в v0.9, а підтримка була припинена в Docker v1.10. Посилання на контейнери Linux зазвичай відносяться до контейнерів Docker, що працюють у Linux.

LXD – це системний контейнер та менеджер віртуальних машин нового покоління. Він пропонує уніфікований інтерфейс для повних систем Linux, що працюють всередині контейнерів або віртуальних машин.

LXD заснований на образах та надає образи для великої кількості дистрибутивів Linux. Він забезпечує гнучкість та масштабованість для різних сценаріїв використання, з підтримкою різних серверних модулів зберігання та типів мереж, а також можливістю встановлення на обладнання, починаючи від окремого ноутбука або хмарного екземпляра та закінчуючи повною серверною стійкою.

При використанні LXD ви можете керувати своїми екземплярами (контейнерами та віртуальними машинами) за допомогою простого інструменту командного рядка, безпосередньо через REST API або за допомогою сторонніх інструментів та інтеграцій. LXD реалізує єдиний REST API як локального, так віддаленого доступу.

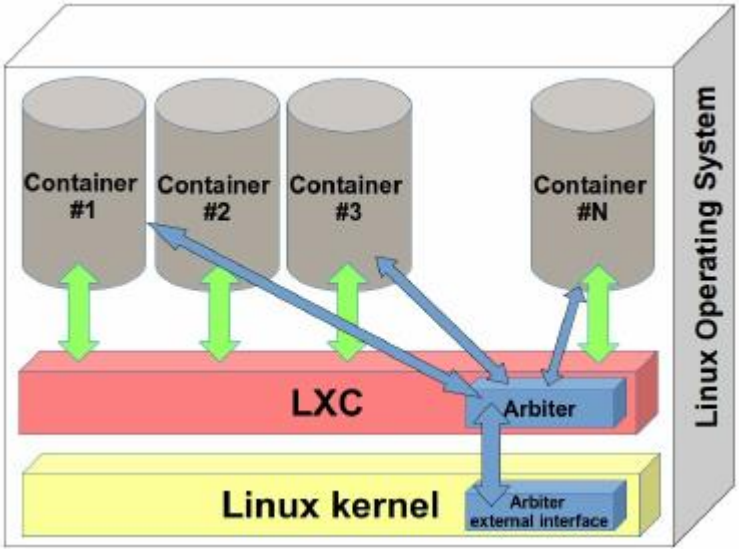

Рисунок 3.6 - Архітектура LXC

3.3.6 Інші Container Runtimes

Firecracker - це технологія віртуалізації з відкритим вихідним кодом, спеціально розроблена для створення та управління безпечними багатокористувацькими контейнерами та сервісами на основі функцій. Firecracker дозволяє розгортати робочі навантаження на легких віртуальних машинах, які називаються microVM, які забезпечують підвищену безпеку та ізоляцію робочих навантажень у порівнянні з традиційними віртуальними машинами, забезпечуючи при цьому швидкість та ефективність використання ресурсів контейнерів. Firecracker був розроблений у Amazon Web Services для підвищення якості обслуговування клієнтів таких сервісів, як AWS Lambda та AWS Fargate. Він виключає непотрібні пристрої та гостьові функції, щоб зменшити обсяг пам'яті та площу атаки кожної мікровіртуальної машини. Це підвищує безпеку, скорочує час запуску та збільшує використання обладнання. Firecracker зазвичай доступний на 64-бітних процесорах Intel, AMD та Arm з підтримкою апаратної віртуалізації. Firecracker використовується/інтегрується (в алфавітному порядку): appfleet, containerd via firecracker-containerd, Fly.io, Kata Containers, Koyeb, Northflank, OpenNebula, Qovery, UniK та Weave FireKube (через Weave Ignite).

Kata Containers - це спільнота з відкритим вихідним кодом, що працює над створенням безпечного середовища виконання контейнерів з легковагими віртуальними машинами, які працюють і працюють як контейнери, але забезпечують більш надійну ізоляцію робочих навантажень з використанням технології апаратної віртуалізації як другий рівень захисту. З моменту запуску в грудні 2017 року співтовариство успішно об'єднало найкращі частини Intel Clear Containers з Hyper.sh RunV і масштабувалося, щоб включити підтримку основних архітектур, включаючи AMD64, ARM, IBM p-series та IBM z-series на додаток до x86_64. Kata Containers також підтримує кілька гіпервізорів, включаючи QEMU,

Cloud-Hypervisor та Firecracker, та, серед іншого, інтегрується з проектом containerd. Спільнота Kata Containers знаходиться під контролем Open Infrastructure Foundation, який підтримує розробку та впровадження відкритої інфраструктури у всьому світі. Код розміщено на GitHub під ліцензією Apache 2.

gVisor – це контейнерна пісочниця, розроблена Google, орієнтована на безпеку, ефективність та простоту використання, випущена у травні 2018 року. gVisor реалізує близько 200 системних викликів Linux в просторі користувача для додаткової безпеки в порівнянні з контейнерами Docker, які працюють безпосередньо поверх Linux. ядро та ізольовані просторами імен. На відміну від ядра Linux, проект написаний безпечною для пам'яті мовою програмування Go, щоб запобігти поширеним помилкам, які часто виникають з програмним забезпеченням, написаним на C. За словами Google і Бреда Фітцпатріка, gVisor використовується у виробничому середовищі Google, такому як стандартне середовище App Engine, хмарні функції, Cloud ML Engine та Google Cloud Run. Нещодавно gVisor був інтегрований з Google Kubernetes Engine і дозволяє користувачам ізолювати свої модулі Kubernetes для таких випадків використання, як SaaS і мультиарендність.

4 АНАЛІЗ ОРКЕСТРАТОРІВ

Оскільки контейнери легкі та недовговічні за своєю природою, запуск їх у виробництво може швидко коштувати величезних зусиль. Зокрема, у поєднанні з мікросервісами, кожен з яких зазвичай запускається у своїх контейнерах, контейнерний додаток може переводитися в роботу з сотнями або тисячами контейнерів, особливо при створенні та експлуатації будь-якої великомасштабної системи.

Це може значно ускладнити роботу, якщо керувати нею вручну. Контейнерне оркестрування - це те, що робить цю операційну складність керованою для розробки

та експлуатації, або DevOps, тому що вона забезпечує декларативний спосіб автоматизації більшої частини роботи. Це робить його придатним для команд і культури DevOps, які зазвичай прагнуть працювати зі значно більшою швидкістю та гнучкістю, ніж традиційні команди розробників програмного забезпечення.

4.1 Контейнерні рішення з опцією self-hosting

4.1.1 OpenShift

Redhat пропонує платформу контейнерів OpenShift як послугу (PaaS). Він допомагає в автоматизації додатків на безпечних та масштабованих ресурсах у гібридних хмарних середовищах. Він надає платформи корпоративного рівня для створення, розгортання та управління контейнерними програмами.

Він побудований на Redhat Enterprise Linux та движку Kubernetes. Openshift має різні функції для управління кластерами через інтерфейс користувача і інтерфейс командного рядка. Redhat надає Openshift у двох варіантах:

- Openshift Online - пропонується як програмне забезпечення як послуга (SaaS)
- OpenShift Dedicated - пропонується як керовані послуги
- Openshift Origin (Origin Community Distribution) - це проект спільноти з відкритим вихідним кодом, який використовується в OpenShift Container Platform, Openshift Online та OpenShift Dedicated.

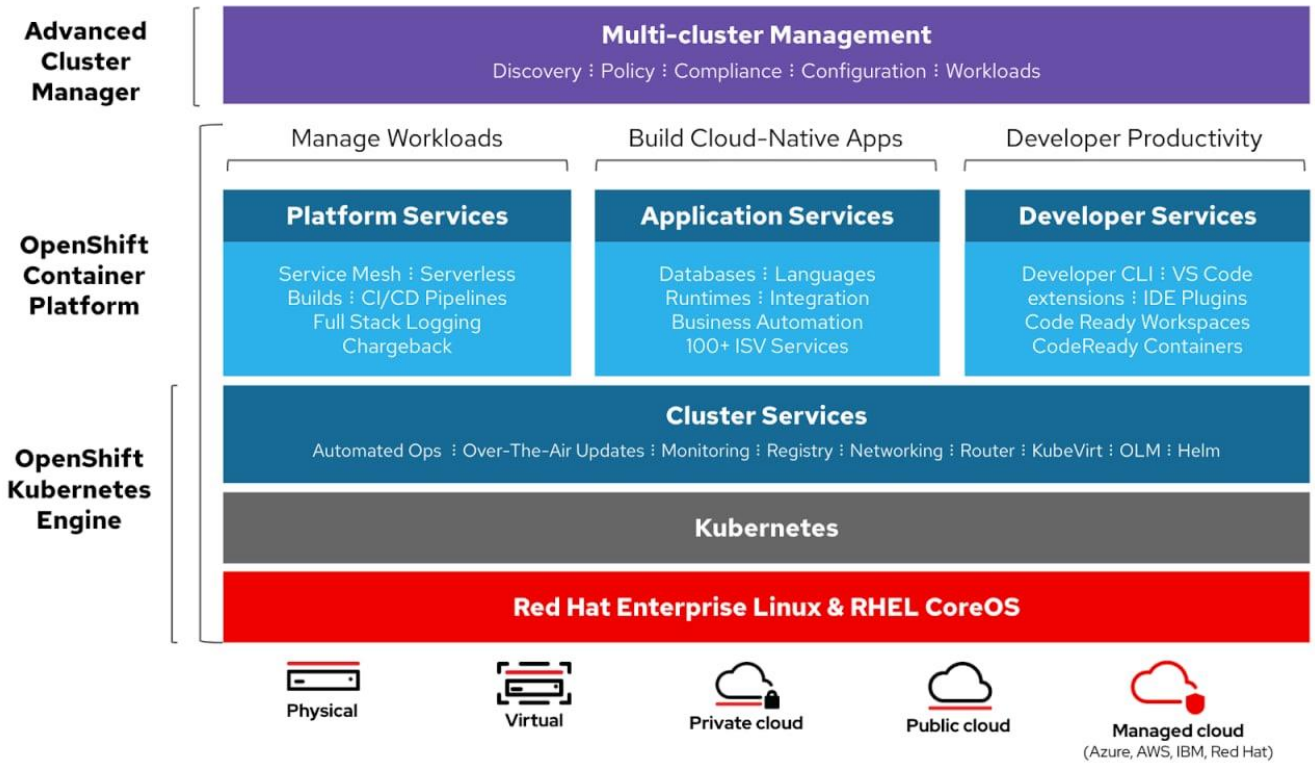

Рисунок 4.1 - Архітектура RedHat OpenShift

4.1.2 Nomad

Nomad - це гнучкий і легкий у використанні оркестратор робочих навантажень для розгортання та управління контейнерами та неконтейнерними програмами у локальному середовищі та у хмарах у будь-якому масштабі. Nomad працює як єдиний двійковий файл із невеликим обсягом ресурсів (35 МБ) і підтримується в macOS, Windows, Linux. Розробники використовують декларативну інфраструктуру як код (IaC) для розгортання своїх додатків та визначають спосіб розгортання програми. Nomad автоматично відновлює програми після збоїв.

Nomad оркеструє програми будь-якого типу (не тільки контейнери). Він забезпечує першокласну підтримку Docker, Windows, Java, віртуальних машин та ін.

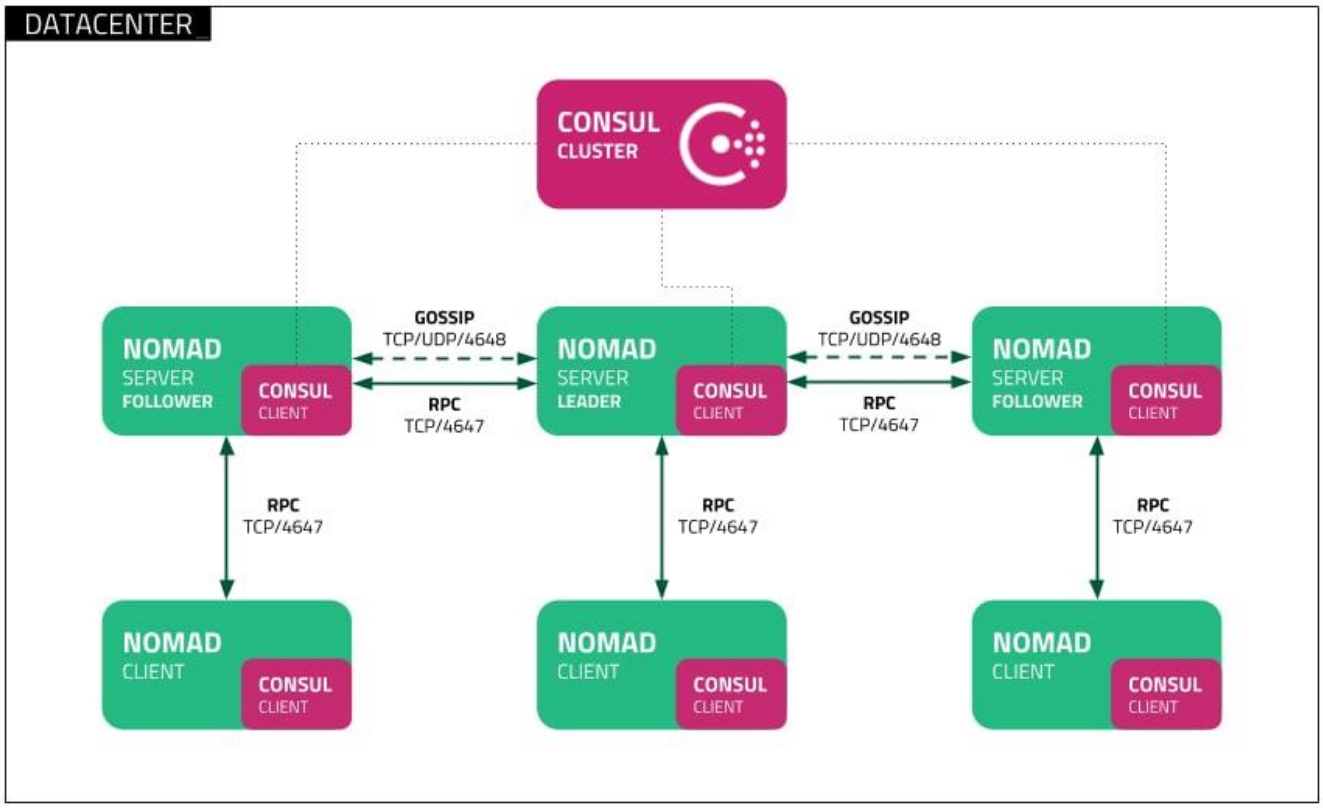

Рисунок 4.2 - Архітектура Hashicorp Nomad

4.1.3 Rancher

Rancher - це платформа з відкритим вихідним кодом, яка використовує оркестрування контейнерів, відому як cattle. Він дозволяє використовувати такі служби оркестрування, як Kubernetes, Swarm, Mesos. Rancher надає програмне забезпечення, необхідне управління контейнерами, тому організаціям не потрібно створювати платформи контейнерних сервісів з нуля, використовуючи окремий набір технологій з відкритим вихідним кодом.

Rancher 2.x дозволяє управляти кластерами Kubernetes, які працюють на вказаних клієнтом провайдерах.

4.2 Хмарні рішення

Розглянемо імплементацію Kubernetes у хмарах Amazon Web Services, Microsoft Azure та Google Cloud Platform.

4.2.1 Azure AKS

AKS - це повністю керована служба Kubernetes, пропонована Azure, яка пропонує безсерверні Kubernetes, безпеку та керування додатками. AKS керує кластером Kubernetes і дозволяє легко розгортати контейнерні програми. AKS автоматично налаштовує всі майстри та вузли Kubernetes, потрібно лише керувати вузлами агента та обслуговувати їх.

AKS – це безкоштовний сервіс; ви платите лише за вузли агентів у своєму кластері, а не за майстри. Можливо створити кластер AKS на порталі Azure або програмно через API. Azure також підтримує додаткові функції, такі як розширені мережеві можливості, інтеграція з Azure Active Directory та моніторинг за допомогою Azure Monitor.

AKS також підтримує контейнери Windows Server. Продуктивність кластера та розгорнутої програми можна відслідковувати за допомогою Azure Monitor. Журнали зберігаються у робочій області Azure Log Analytics.

AKS сертифікований як Kubernetes-сумісний продукт.

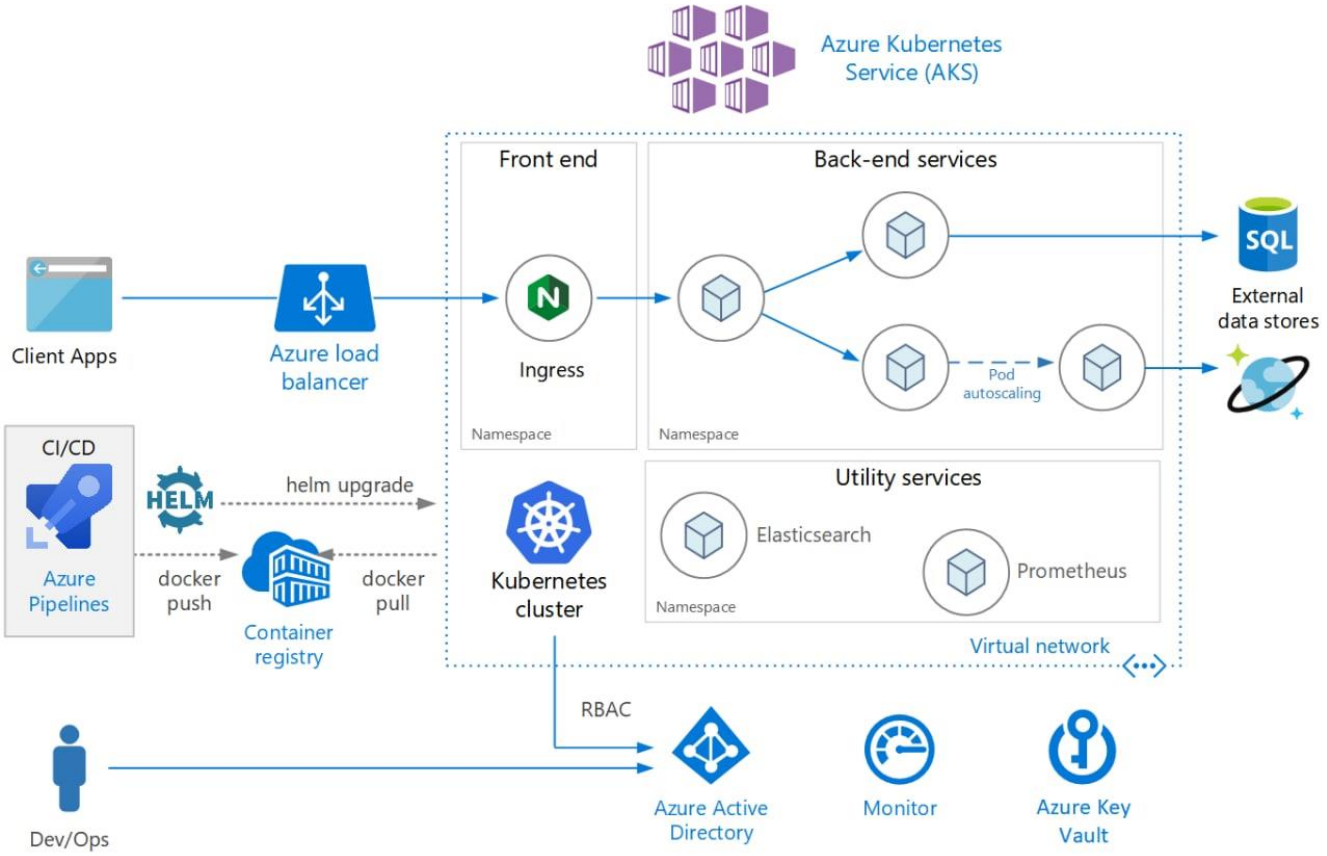

Рисунок 4.3 - Зразок імплементації Azure AKS

4.2.2 GCP GKE

Google Kubernetes Engine надає повністю кероване рішення для оркестрації контейнерних програм на Google Cloud Platform. Кластери GKE працюють на Kubernetes. Ви можете взаємодіяти із кластерами за допомогою Kubernetes CLI. Команди Kubernetes можна використовувати для розгортання програм та управління ними, виконання завдань адміністрування, встановлення політик та моніторингу стану розгорнутих робочих навантажень.

Розширені функції керування Google Cloud також доступні з кластерами GKE, такими як балансування навантаження Google Cloud, пули вузлів, автоматичне масштабування вузлів, автоматичне оновлення, автоматичне

відновлення вузлів, ведення журналу та моніторинг за допомогою пакета операцій Google Cloud.

Google Cloud надає інструменти CI/CD, які допоможуть створювати та обслуговувати контейнери програм. Можна використовувати Cloud Build для створення образів контейнерів (наприклад, Docker) із різних репозиторіїв вихідного коду та реєстру контейнерів для зберігання образів контейнерів.

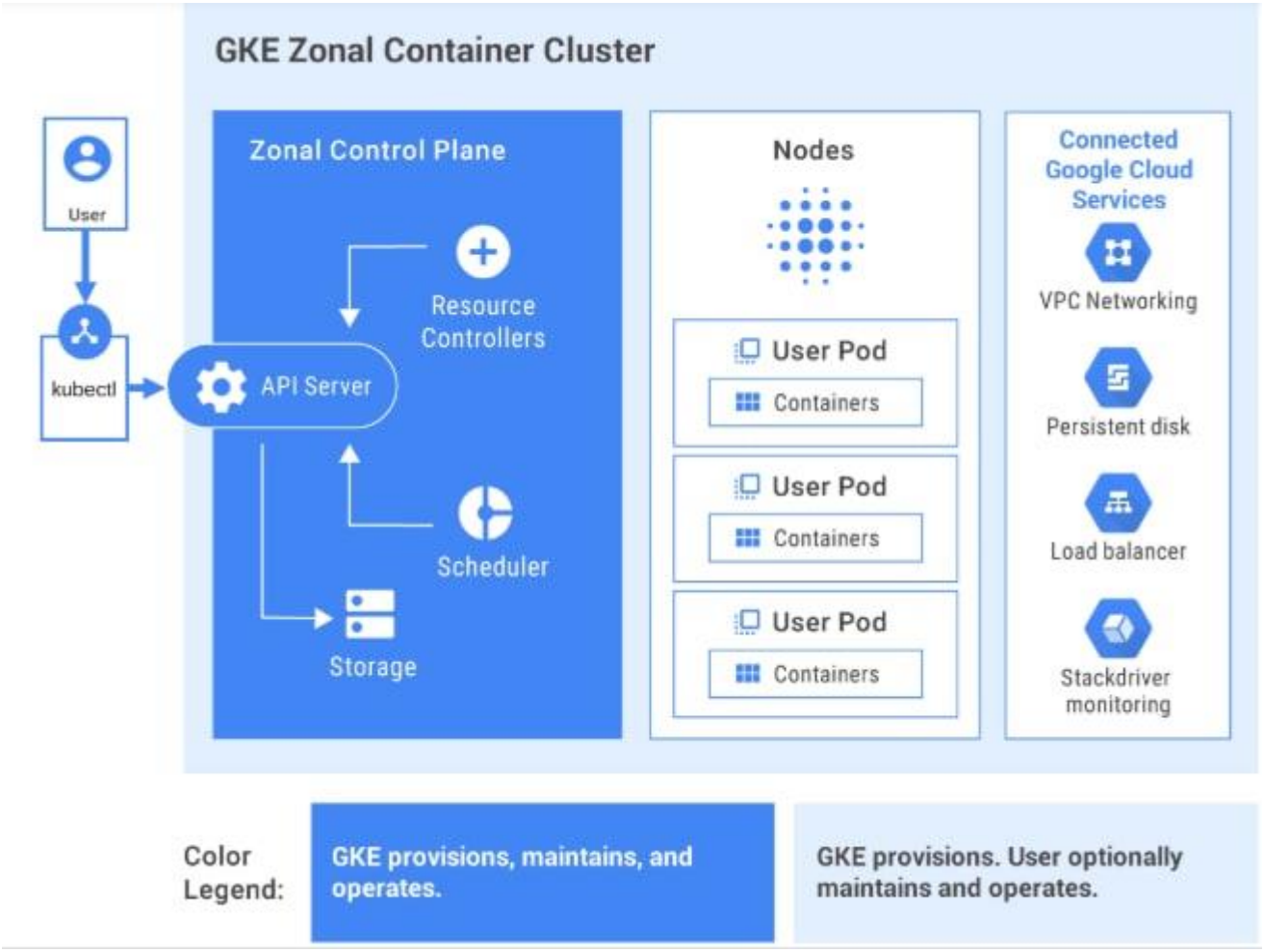

Рисунок 4.4 - Архітектура та розподіл відповідальності у GKE

4.2.3 AWS EKS

Amazon Elastic Kubernetes Service (Amazon EKS) – це керований сервіс Kubernetes, який спрощує запуск Kubernetes на AWS та локально. Kubernetes - це система з відкритим вихідним кодом для автоматизації розгортання, масштабування та управління контейнерними програмами. Amazon EKS сертифікований як Kubernetes-сумісний, тому існуючі програми, що працюють у висхідному потоці Kubernetes, сумісні з Amazon EKS.

Amazon EKS автоматично керує доступністю та масштабованістю вузлів площини управління Kubernetes, що відповідають за планування контейнерів, керування доступністю додатків, зберігання даних кластера та інші ключові завдання.

Amazon EKS дозволяє запускати програми Kubernetes як у Amazon Elastic Compute Cloud (Amazon EC2), так і в AWS Fargate. З Amazon EKS можливо скористатися перевагами продуктивності, масштабованості та доступності інфраструктури AWS, а також інтеграції з мережевими службами AWS та службами безпеки, такими як балансувальники навантаження програм (ALB) для розподілу навантаження, AWS Identity та Access. Інтеграція управління (IAM) з контролем доступу на основі ролей (RBAC) та підтримка AWS Virtual Private Cloud (VPC) для мережевих модулів.

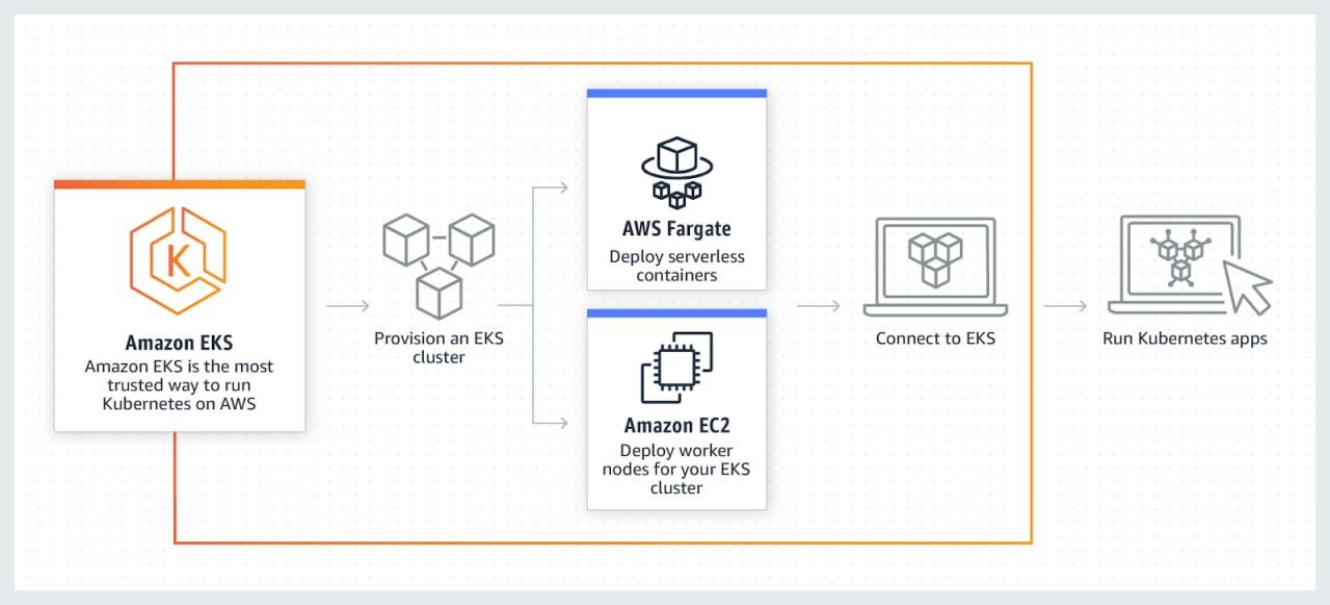

Рисунок 4.5 - Архітектура EKS

5 СИНТЕЗ СИСТЕМИ

Перед початком налаштування кластера Amazon EKS в обліковому записі Amazon Web Services у зазначеному регіоні потрібно перевірити наявність інших кластерів за допомогою CLI утиліти.

Command:

```
aws eks list-clusters
```

Output:

```
{
    "clusters": [
        "devel",
        "prod"
    ]
}
```

Рисунок 5.1 - CLI команда list-clusters

Наступним кроком можна отримати опис та деталі створених кластерів.

Command:

```
aws eks describe-cluster --name devel
```

Output:


```
{
    "cluster": {
        "name": "devel",
        "arn": "arn:aws:eks:us-west-2:012345678910:cluster/devel",
        "createdAt": 1527807879.988,
        "version": "1.10",
        "endpoint": "https://EXAMPLE0A04F01705DD065655C30CC3D.yl4.us-west-2.eks.amazonaw
s.com",
        "roleArn": "arn:aws:iam::012345678910:role/eks-service-role-AWSServiceRoleForAmaz
onEKS-J7ONKE3BQ4PI",
        "resourcesVpcConfig": {
            "subnetIds": [
                "subnet-6782e71e",
                "subnet-e7e761ac"
            ],
            "securityGroupIds": [
                "sg-6979fe18"
            ],
            "vpcId": "vpc-950809ec"
        },
        "status": "ACTIVE",
        "certificateAuthority": {
            "data": "EXAMPLECRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUN5RENDQWJDZ0F3SUJBZ0lCQURB
TkJna3Foa2lHOXcwQkFRc0ZBREFWTVJNd0VRWURWUVFERXdwcmRXSmwKY201bGRHVnpNQjRYRFRFNE1EVXpNVEl6T
VRFek1Wb1hEVEk0TURVeU9ESXpNVEV6TVZvd0ZURVRNQkVHQTFVRQpBeE1LYTNWaVpYSnVaWFJsY3pDQ0FTSXdEUV
lKS29aSWh2Y05BUUVCQlFBRGdnRVBBRENDQVFvQ2dnRUJBTTZWCjVUaG4rdFcySm9Xa2hQMzRlVUZMNitaRXJOZGI
vWVdrTmtDdWNGS2RaaXl2TjlMVmdvUmV2MjlFVFZlN1ZGbSsKUTJ3ZURyRXJiQyt0dVlibkFuN1ZLYmE3ay9hb1BH
ekZMdmVnb0t6b0M1N2NUdGVwZzRIazRlK2tIWHNaME10MApyb3NzcjhFM1ROeExETnNJTThGL1cwdjhsTGNCbWRPc
jQyV2VuTjFHZXJnaDNSZ2wzR3JIazBnNTU0SjFWenJZCm9hTi8zODFUczlOTFF2QTBXb0xIcjBFRlZpTFdSZEoyZ3
lXaC9ybDVyOFNDOHZaQXg1YW1BU0hVd01aTFpWRC8KTDBpOW4wRVM0MkpVdzQyQmxHOEdpd3NhTkJWV3lUTHZKclN
hRXlDSHFtVVZaUTFDZkFXUjl0L3JleVVOVXM3TApWV1FqM3BFbk9RMitMSWJrc0RzQ0F3RUFBYU1qTUNFd0RnWURW
UjBQQVFIL0JBUURBZ0trTUE4R0ExVWRFd0VCCi93UUZNQU1CQWY4d0RRWUpLb1pJaHZjTkFRRUxCUUFEZ2dFQkFNZ
3RsQ1dIQ2U2YzVHMXl2YlFTS0Q4K2hUalkKSm1NSG56L2EvRGt0WG9YUjFVQzIrZUgzT1BZWmVjRVZZZHVaSlZCck
NNQ2VWR0ZkeWdBYlNLc1FxWDg0S2RXbAp1MU5QaERDSmEyRHliN2pVMUV6VThTQjFGZUZ5ZFE3a0hNS1E1blpBRVF
QOTY4S01hSGUrSm0yQ2x1UFJWbEJVCjF4WlhTS1gzTVZ0K1Q0SU1EV2d6c3JRSjVuQkRjdEtLcUZtM3pKdVVubHo5
ZEpVckdscEltMjVJWXJDckxYUFgKWkUwRUtRNWEzMHhkVWNrTHRGQkQrOEtBdFdqSS9yZUZPNzM1YnBMdVoyOTBaN
m42QlF3elRrS0p4cnhVc3QvOAppNGsxcnlsaUdWMm5SSjBUYjNORkczNHgrYWdzYTRoSTFPbU90TFM0TmgvRXJxT3
lIUXNDc2hEQUtKUT0KLS0tLS1FTkQgQ0VSVElGSUNBVEUtLS0tLQo="
        }
    }
}
```


Рисунок 5.2 - CLI команда describe-cluster

5.1 Налаштування залежностей EKS

Amazon EKS рекомендує запускати кластер у VPC із загальнодоступними та приватними підмережами, щоб Kubernetes міг створювати загальнодоступні балансувальники навантаження у загальнодоступних підмережах, які балансують трафік для модулів, що працюють на вузлах, що знаходяться у приватних підмережах.

5.1.1 Налаштування мережі

Під час створення кластера Amazon EKS вказуються підмережі VPC, в яких Amazon EKS може розмістити еластичні мережеві інтерфейси. Amazon EKS вимагає наявності підмереж як мінімум у двох зонах доступності та створює до чотирьох мережних інтерфейсів у цих підмережах для полегшення зв'язку рівня управління з вашими вузлами. Цей канал зв'язку підтримує такі функції Kubernetes як kubectl exec і kubectl logs. Група безпеки кластеру, створена Amazon EKS, та будь-які додаткові групи безпеки, вказані під час створення кластера, застосовуються до цих мережних інтерфейсів. В описі кожного інтерфейсу, створеного Amazon EKS, вказано ім'я кластера Amazon EKS.

Потрібно переконатися, що в підмережах, вказаних під час створення кластера, достатньо доступних IP-адрес для мережевих інтерфейсів, створених Amazon EKS. Рекомендовано створювати невеликі (/28) виділені підмережі для мережних інтерфейсів, створених Amazon EKS, та вказувати ці підмережі лише як частину створення кластера. Інші ресурси, такі як вузли та балансувальники навантаження, повинні запускатися у підмережах, відмінних від підмереж, зазначених під час створення кластера.

Вузли повинні мати можливість зв'язуватися з площиною управління та іншими сервісами AWS. Якщо вузли розгорнуті у приватній підмережі, то підмережа повинна відповідати одній з наступних вимог:

- Має стандартний маршрут до шлюзу NAT. Шлюзу NAT має бути призначена загальнодоступна IP-адреса для забезпечення доступу до Інтернету.
- Мережа налаштована з необхідними опціями та вимогами до Приватних кластерів.

Якщо самоврядні вузли розгортаються у загальнодоступній підмережі, підмережа повинна бути настроєна на автоматичне призначення загальнодоступних IP-адрес. Якщо мережа не настроєна на автоматичне призначення загальнодоступних IP-адрес, вузлам не призначається загальнодоступна IP-адреса. Визначити, чи налаштовані загальнодоступні підмережі для автоматичного призначення загальнодоступних IP-адрес, можна за допомогою наступної команди.

```
aws ec2 describe-subnets \
    --filters "Name=vpc-id,Values=VPC-ID" | grep 'SubnetId\|MapPublicIpOnLaunch'
```

Рисунок 5.3 CLI команда describe-subnets

Для будь-яких підмереж, для яких для MapPublicIpOnLaunch встановлено значення false, можна змінити значення параметра на true.

```
aws ec2 modify-subnet-attribute --map-public-ip-on-launch --subnet-id subnet-aaaaaaaaaaaaaaaaa
```

Рисунок 5.4 - Встановлення атрібуту MapPublicIpOnLaunch

5.1.2 Налаштування IAM

AWS Identity and Access Management (IAM) – це сервіс AWS, який допомагає адміністратору безпечно контролювати доступ до ресурсів AWS. Адміністратори

IAM контролюють, хто може бути аутентифікований (увійти в систему) та авторизований (мати дозволи) використання ресурсів Amazon EKS. IAM це сервіс AWS, яким можна користуватися без додаткової оплати.

Користувач служби. Якщо служба Amazon EKS використовується для виконання своєї роботи, адміністратор надасть користувачу необхідні облікові дані та дозволи.

Адміністратор служби. Якщо особа відповідає за ресурси Amazon EKS у компанії, ймовірно, що мається повний доступ до Amazon EKS. Потрібно визначити, до яких функцій та ресурсів Amazon EKS слід звертатися. Потім адміністратору IAM відсилаються запити на зміну дозволів користувачів служби.

Адміністратор IAM. Повинен писати глобальні політики для керування доступом до Amazon EKS.

Існує два типи політик: Identity-based policy and Resource-based policy.

Політики на основі ідентифікаційних даних - це документи політики дозволів JSON, які можна прикріпити до, наприклад, користувача IAM, групи користувачів або ролі. Ці політики визначають, які дії можуть виконувати користувачі та ролі, на яких ресурсах та за яких умов.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "service-prefix:action-name",
            "Resource": "*",
            "Condition": {
                "DateGreaterThan": {"aws:CurrentTime": "2020-04-01T00:00:00Z"},
                "DateLessThan": {"aws:CurrentTime": "2020-06-30T23:59:59Z"}
            }
        }
    ]
}
```

Рисунок 5.5 - Політика на основі ідентифікаційних даних

Політики на основі ідентифікаційних даних можна розділити на вбудовані або керовані політики. Вбудовані політики вбудовуються безпосередньо в одного користувача, групу чи роль. Керовані політики - це окремі політики, які можна прикріпити до кількох користувачів, груп та ролей у своєму обліковому записі AWS. До керованих політик відносяться політики, керовані AWS, і політики, керовані клієнтами.

Політики на основі ресурсів – це документи політики JSON, які прикріплюються до ресурсу. Прикладами політик на основі ресурсів є політики довіри ролей IAM та політики бакетів Amazon S3. У службах, які підтримують політики на основі ресурсів, адміністратори служб можуть використовувати їх для керування доступом до певного ресурсу. Для ресурсу, якого прикріплена політика, політика визначає, які дії зазначений принципал може виконувати з цим ресурсом і за яких умов. Приципали можуть містити облікові записи, користувачів, ролі, федеративних користувачів або сервіси AWS.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "AccountBAccess1",
    "Effect": "Allow",
    "Principal": {"AWS": "111122223333"},
    "Action": "s3:*",
    "Resource": [
      "arn:aws:s3:::mybucket",
      "arn:aws:s3:::mybucket/*"
    ]
  }
}
```

Рисунок 5.6 - Політика на основі ресурсів

Політики на основі ресурсів – це вбудовані політики, які перебувають у цій службі.

6 ЕКСПЕРИМЕНТАЛЬНИЙ РОЗДІЛ

6.1 Контейнеризація додатка

Docker може автоматично створювати образи, читаючи вказівки з файлу. Dockerfile - це текстовий документ, що містить усі команди, які користувач може викликати у командному рядку для збирання зображення. Використовуючи docker build, користувач може створити автоматизований процес складання, який послідовно виконує кілька інструкцій командного рядка.

Dockerfile

```
1   FROM golang:1.10.0
2   RUN go get github.com/codegangsta/negroni \
3                  github.com/gorilla/mux \
4                  github.com/xyproto/simpleredis
5   WORKDIR /app
6   ADD ./main.go .
7   RUN CGO_ENABLED=0 GOOS=linux go build -o main .
8
9   FROM scratch
10  WORKDIR /app
11  COPY --from=0 /app/main .
12  COPY ./public/index.html public/index.html
13  COPY ./public/script.js public/script.js
14  COPY ./public/style.css public/style.css
15  CMD ["/app/main"]
16  EXPOSE 3000
```

Рисунок 6.1 - Контенти Dockerfile

Демон Docker запускає інструкції в Dockerfile одну за іншою, при необхідності фіксуючи результат кожної інструкції в новому образі, перш ніж нарешті вивести ідентифікатор нового образу. Демон Docker автоматично очистить надісланий контекст.

Ця CLI команда виведе список створених локально Docker Images

```
$ docker images

REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
<none>              <none>              77af4d6b9913        19 hours ago        1.089 GB
committ             latest              b6fa739cedf5        19 hours ago        1.089 GB
<none>              <none>              78a85c484f71        19 hours ago        1.089 GB
docker              latest              30557a29d5ab        20 hours ago        1.089 GB
<none>              <none>              5ed6274db6ce        24 hours ago        1.089 GB
postgres            9                   746b819f315e        4 days ago          213.4 MB
postgres            9.3                 746b819f315e        4 days ago          213.4 MB
postgres            9.3.5               746b819f315e        4 days ago          213.4 MB
postgres            latest              746b819f315e        4 days ago          213.4 MB
```

Рисунок 6.2 - Список Docker Images

6.2 Створення сутностей Kubernetes

Об'єкти Kubernetes – це постійні сутності в системі Kubernetes. Kubernetes використовує ці сутності для представлення стану кластера. Зокрема, вони можуть описувати:

- Які контейнеризовані додатки працюють (і на яких вузлах)
- Ресурси, доступні цим додаткам
- Політики поведінки цих додатків, такі як політики перезапуску, оновлення та відмовостійкості.

Об'єкт Kubernetes - це запис про наміри: як тільки об'єкт буде створен, система Kubernetes буде постійно працювати щоб гарантувати, що об'єкт існує. Створюючи об'єкт, фактично системі Kubernetes відправляється повідомлення, як має виглядати робоче навантаження кластера.

Для роботи з об'єктами Kubernetes – будь то їх створення, зміна або видалення – необхідно використовувати Kubernetes API. Наприклад, коли використовується інтерфейс командного рядка kubectl, CLI виконує за необхідні виклики Kubernetes API.

Об'єкт ReplicaSet

Ціль ReplicaSet - підтримувати стабільний набір реплік Pod, що працюють у будь-який момент часу. Таким чином, він часто використовується для забезпечення доступності певної кількості ідентичних модулів.

ReplicaSet визначається полями, включаючи селектор, який вказує, як ідентифікувати поди, які він може отримати, кількість реплік, що вказує скільки подів він повинен підтримувати, і шаблон пода, що визначає дані нових подів, які він повинен створити, щоб відповідати кількості. критеріїв реплік. Потім ReplicaSet виконує своє завдання, створюючи та видаляючи модулі в міру потреби для

досягнення бажаного числа. Коли ReplicaSet необхідно створити нові модулі, він використовує шаблон модуля.

Об'єкт Deployment

Deployment забезпечує декларативні оновлення для модулів Pod та ReplicaSet.

Бажаний стан є описаним у Deployment, а Deployment Controller змінює фактичний стан на бажане з контрольованою швидкістю. Deployment може бути визначиним для створення нових наборів реплік або видалення існуючих Deployment та використання всіх їх ресурсів з новими Deployment.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        resources:
          limits:
            cpu: 400m
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 200Mi
        ports:
        - containerPort: 80
```

Рисунок 6.3 - Зразок налаштування Deployment

Об'єкт DaemonSet

DaemonSet гарантує, що всі (або деякі) вузли запускають копію Pod. У міру додавання вузлів у кластер до них додаються поди. Коли вузли віддаляються із кластера, ці поди збираються збирачем сміття. Видалення DaemonSet очистить створені ним поди.

Типові варіанти використання DaemonSet:

- запуск демону сховища кластера на кожному вузлі
- запуск демона збору логів на кожному вузлі
- запуск демона моніторингу вузлів на кожному вузлі

У простому випадку один DaemonSet, що охоплює всі вузли, використовуватиметься для кожного типу демона. Більш складна установка може використовувати кілька наборів DaemonSet для одного типу демона, але з різними прапорами та/або різними запитами пам'яті та процесора для різних типів обладнання.

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-elasticsearch
  namespace: kube-system
  labels:
    k8s-app: fluentd-logging
spec:
  selector:
    matchLabels:
      name: fluentd-elasticsearch
  template:
    metadata:
      labels:
        name: fluentd-elasticsearch
    spec:
      tolerations:
      # this toleration is to have the daemonset runnable on master nodes
      # remove it if your masters can't run pods
      - key: node-role.kubernetes.io/master
        operator: Exists
        effect: NoSchedule
      containers:
      - name: fluentd-elasticsearch
        image: quay.io/fluentd_elasticsearch/fluentd:v2.5.2
        resources:
          limits:
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 200Mi
        volumeMounts:
        - name: varlog
          mountPath: /var/log
        - name: varlibdockercontainers
          mountPath: /var/lib/docker/containers
          readOnly: true
      terminationGracePeriodSeconds: 30
      volumes:
      - name: varlog
        hostPath:
          path: /var/log
      - name: varlibdockercontainers
        hostPath:
          path: /var/lib/docker/containers
```

Рисунок 6.4 - Зразок налаштування DaemonSet

Об'єкт StatefulSet

StatefulSet - це об'єкт API робочого навантаження, який використовується для керування програмами з відстеженням стану.

Як і Deployment, StatefulSet управляє модулями, заснованими на ідентичній специфікації контейнера. На відміну від Deployment, StatefulSet підтримує постійну ідентифікацію кожного зі своїх модулів. Ці модулі створені на основі однієї і тієї ж специфікації, але не взаємозамінні: у кожного є постійний ідентифікатор, який він підтримує при будь-якій зміні розкладу.

Якщо потрібно використовувати томи зберігання для забезпечення стійкості робочого навантаження, можна використовувати StatefulSet як частину рішення. Хоча окремі модулі в StatefulSet схильні до збоїв, постійні ідентифікатори модулів спрощують зіставлення існуючих томів з новими модулями, що замінюють всіх, хто вийшов з ладу.

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: nginx
  serviceName: "nginx"
  replicas: 3
  minReadySeconds: 10
  template:
    metadata:
      labels:
        app: nginx
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: nginx
        image: k8s.gcr.io/nginx-slim:0.8
        resources:
          limits:
            cpu: 400m
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 200Mi
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: www
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
  - metadata:
      name: www
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: "my-storage-class"
      resources:
        requests:
          storage: 1Gi
```

Рисунок 6.5 - Зразок налаштування StatefulSet

6.3 Отримання стану сутностей Kubernetes

Інструмент командного рядка Kubernetes, kubectl, дозволяє запускати команди для кластерів Kubernetes. kubectl можна використовувати для розгортання програм, перевірки ресурсів кластера та управління ними, а також для перегляду логів.

```
# Get commands with basic output
kubectl get services                          # List all services in the namespace
kubectl get pods --all-namespaces             # List all pods in all namespaces
kubectl get pods -o wide                      # List all pods in the current namespace, with more details
kubectl get deployment my-dep                 # List a particular deployment
kubectl get pods                              # List all pods in the namespace
kubectl get pod my-pod -o yaml                # Get a pod's YAML

# Describe commands with verbose output
kubectl describe nodes my-node
kubectl describe pods my-pod

# List Services Sorted by Name
kubectl get services --sort-by=.metadata.name
```

Рисунок 6.6 - Список команд для отримання деяких сутностей Kubernetes

Kubectl get друкує таблицю найважливішої інформації про зазначені ресурси. Можна фільтрувати список, використовуючи селектор тегів та лейбл -selector. Якщо бажаний тип ресурсу знаходиться в просторі імен, надруковане буде тільки в поточному просторі імен, якщо не буде параметру --all-namespaces.

```
NAMESPACE     NAME                              READY   STATUS    RESTARTS   AGE
kube-system   coredns-5644d7b6d9-nv9mj          0/1     Pending   0          12m
kube-system   coredns-5644d7b6d9-p7cz5          0/1     Pending   0          12m
kube-system   etcd-master                       1/1     Running   0          11m
kube-system   kube-apiserver-master             1/1     Running   0          11m
kube-system   kube-controller-manager-master    1/1     Running   0          11m
kube-system   kube-flannel-ds-amd64-xdkwp       1/1     Running   0          5m56s
kube-system   kube-proxy-k7875                  1/1     Running   0          12m
kube-system   kube-scheduler-master             1/1     Running   0          11m
```

Рисунок 6.7 - Зразок виводу kubectl get pods –namespace kube-system

Kubectl describe друкує детальний опис вибраних ресурсів, включаючи пов'язані ресурси, такі як події або контролери. Можна вибрати один об'єкт на ім'я, всі об'єкти цього типу, вказати префікс імені або селектор міток.

```
$ kubectl describe pod nginx-5557945897-pbqcc
Name:               nginx-5557945897-pbqcc
Namespace:          default
Priority:           0
PriorityClassName:  <none>
Node:               gke-k8s-security-default-pool-f7030ead-71wg/10.128.0.17
Start Time:         Tue, 05 Feb 2019 09:56:30 +0530
Labels:             pod-template-hash=1113501453
                    run=nginx
Annotations:        kubernetes.io/limit-ranger: LimitRanger plugin set: cpu request for container nginx
Status:             Running
IP:                 10.36.3.10
Controlled By:      ReplicaSet/nginx-5557945897
Containers:
  nginx:
    Container ID:   docker://ac885ec636bbc715f6faa6f6acca5816c973bb25103a065300bff2cf341ca648
    Image:          nginx:alpine
    Image ID:       docker-pullable://nginx@sha256:c9a462bce16d85dde1861d45aebfae80c812d733187d045cbb7ffe3d71ac37bd
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Tue, 05 Feb 2019 09:57:39 +0530
    Ready:          True
    Restart Count:  0
    Requests:
      cpu:        100m
    Environment:  <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-ngfj7 (ro)
```

Рисунок 6.8 - Зразок виводу kubectl describe pod

ВИСНОВКИ

В рамках кваліфікаційної роботи магістра було досліджено типи обчислювальних потужностей, доступні для оптимізації існуючої інфраструктури. Було вивчено сферу контейнеризації, з типами запуску додатків та сервісів., їх характеристики та переваги. Були порівняні оркестратори, їх характеристики та їхня актуальність на ринку.

В результаті аналізу сучасних платформ контейнеризації, враховуючи поточну архітектуру та раніше обраний провайдер хмарних послуг було виявлено, що продукт від Amazon Web Services Elastic Kubernetes Service відповідає всім критеріям та має великий вибір інструментів для гнучкого налаштування додатків та сервісів.

Для налаштування та налагодження оточення були використані нативні інструменти, такі як AWS CLI, kubectl та формат конфігураційних файлів YAML.

ПЕРЕЛІК ПОСИЛАННЬ

1. Brendan Burns, Joe Beda, Kelsey Hightower, Kubernetes: Up and Running: Dive Into the Future of Infrastructure, 2017 – 278 с.
2. Порівняння оркестраторів [Електронний ресурс] - https://geekflare.com/container-orchestration-software/
3. Документація Docker [Електронний ресурс] - https://www.docker.com/
4. AWS Documentation [Електронний ресурс] - https://aws.amazon.com/
5. VMWare Documentation [Електронний ресурс] - https://www.vmware.com/
6. History of Containers [Електронний ресурс] - https://www.section.io/engineering-education/history-of-container-technology/
7. Containerd [Електронний ресурс] - https://habr.com/ru/company/flant/blog/325358/
8. Container Orchestration Tools [Електронний ресурс] - https://geekflare.com/container-orchestration-software/#anchor-aws-eks
9. Kubernetes [Електронний ресурс] - https://kubernetes.io/docs

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
“ДНІПРОВСЬКА ПОЛІТЕХНІКА”

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ
КОНФІГУРАЦІЯ ТА ДОКУМЕНТАЦІЯ ПРОГРАМИ

Текст програми
804.02070743.22005-01 12 01

Листів 6

2022

АНОТАЦІЯ

Даний лістинг містить в собі частину програмного коду для створення контейнерів використовуючи Docker Dockerfile, та YAML конфігурації об'єктів Kubernetes, котрі потрібні для створення додатку всередині інфраструктури оркестратора.

ЗМІСТ

1. Dockerfile додатку

```
FROM golang:1.10.0

RUN go get github.com/codegangsta/negroni \

        github.com/gorilla/mux \

        github.com/xyproto/simpleredis

WORKDIR /app

ADD ./main.go .

RUN CGO_ENABLED=0 GOOS=linux go build -o main .

FROM scratch

WORKDIR /app

COPY --from=0 /app/main .

COPY ./public/index.html public/index.html

COPY ./public/script.js public/script.js

COPY ./public/style.css public/style.css

CMD ["/app/main"]
```

```
EXPOSE 3000
```

2. Налаштування розгортання додатку

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: go-deployment
  labels:
    app: go-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: go-app
  template:
    metadata:
      labels:
        app: go-app
    spec:
      serviceAccountName: go-app-sa
      containers:
      - name: go-app
        image: go-app:1.0.0
        resources:
          limits:
            cpu: 400m
            memory: 200Mi
          requests:
```

```
            cpu: 100m
            memory: 100Mi
        ports:
        - containerPort: 88
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: go-app-sa
secrets:
- name: generated-token
imagePullSecrets:
- name: myregistrykey
---
apiVersion: v1
kind: Service
metadata:
  name: go-service
  labels:
    app: go-app
spec:
  ports:
  - port: 88
    targetPort: 88
  selector:
    app: go-app
```

3. Налаштування StatefulSet

```
apiVersion: apps/v1
```

```
kind: StatefulSet
metadata:
  name: golang-ss
spec:
  selector:
    matchLabels:
      app: golang-ss
  serviceName: "golang-ss"
  replicas: 3
  minReadySeconds: 10
  template:
    metadata:
      labels:
        app: golang-ss
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: golang-ss
        image: golang-ss:0.8
        resources:
          limits:
            cpu: 400m
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 200Mi
        ports:
        - containerPort: 88
          name: application
```

```
        volumeMounts:
        - name: app
          mountPath: /usr/share/app
  volumeClaimTemplates:
  - metadata:
      name: app
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: "local-sc"
      resources:
        requests:
          storage: 1Gi
---
apiVersion: v1
kind: Service
metadata:
  name: golang-ss
  labels:
    app: golang-ss
spec:
  ports:
  - port: 88
    targetPort: 88
  selector:
    app: golang-ss
```