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

Навчально-науковий інститут телекомунікаційних систем

Кафедра телекомунікацій

«На правах рукопису»
УДК _____

«До захисту допущено»
Завідувач кафедри
_____ Сергій КРАВЧУК
«___»_____2024 р.

Магістерська дисертація

на здобуття ступеня магістра
за освітньо-професійною програмою «Інженерія та програмування інфокомунікацій»
зі спеціальності 172 «Електронні комунікації та радіотехніка»

на тему: «Аналіз та моделювання впливу мережевої топології на надійність та доступність кластерів Kubernetes»

Виконав:
студент ІІ курсу, групи ЦК-32мп
Кулиняк Данило Михайлович _____

Керівник:
Старший викладач кафедри ТК НН ІТС, к.т.н.
Маньківський Володимир Броніславович _____

Рецензент:
Старший викладач кафедри ЕКІР НН ІТС, к.т.н.
Новіков Валерій Іванович _____

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

Київ – 2024 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут телекомунікаційних систем
Кафедра телекомунікацій

Рівень вищої освіти – другий (магістерський)

Спеціальність – 172 «Електронні комунікації та радіотехніка»

Освітньо-професійна програма «Інженерія та програмування інфокомунікацій»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Сергій КРАВЧУК

«___»_____2024 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Кулиняці Данилу Михайловичу

1. Тема дисертації «Аналіз та моделювання впливу мережевої топології на надійність та доступність кластерів Kubernetes», науковий керівник дисертації Маньківський Володимир Броніславович, к.т.н., затверджені наказом по університету від «07» листопада 2024 р. № 4989-с.

2. Термін подання студентом дисертації 10.12.2024 р.

3. Об’єкт дослідження - процеси функціонування Kubernetes-кластерів, зокрема їхня мережева інфраструктура, що включає такі компоненти як control plane, вузли, поди та інші сервіси, які забезпечують безперервну взаємодію в кластері

4. Предмет дослідження - методи та моделі, які дозволяють оцінити та оптимізувати мережеву топологію Kubernetes-кластера для забезпечення його надійності та доступності, з урахуванням механізмів балансування навантаження, планування pod'ів, обмежень на ресурси та інших факторів впливу.

5. Перелік завдань, які потрібно розробити:

- Провести аналіз літературних джерел щодо принципів побудови та функціонування Kubernetes кластерів
- Дослідити мережеві аспекти та показники надійності Kubernetes
- Розробити математичну модель мережевої топології кластера
- Формалізувати мережеву структуру та створити модель оцінки надійності
- Розробити програмне рішення для аналізу різних топологій кластера
- Провести порівняльний аналіз метрик надійності та продуктивності
- Розробити стартап-проект та економічний аналіз
- Оформити пояснювальну записку та підготуватись до захисту

6. Орієнтовний перелік ілюстративного матеріалу:

- Схема архітектури Kubernetes кластера
- Діаграми основних компонентів мережевої топології
- Графіки порівняння показників надійності різних топологій
- Блок-схема алгоритму аналізу топології
- Результати моделювання та оцінки продуктивності
- Візуалізація результатів експериментальних досліджень
- Економічні показники та метрики ефективності
- Презентаційні матеріали для захисту роботи

7. Дата видачі завдання “ 20 ” жовтня 2023 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Аналіз літературних джерел щодо принципів побудови та функціонування Kubernetes кластерів	20.10.2023 - 03.11.2023	виконано
2	Дослідження мережевих аспектів та показників надійності Kubernetes	04.11.2023 - 17.11.2023	виконано
3	Розробка математичної моделі мережевої топології кластера	18.11.2023 - 01.12.2023	виконано
4	Формалізація мережевої структури та створення моделі оцінки надійності	02.12.2023 - 15.12.2023	виконано
5	Розробка програмного рішення для аналізу різних топологій кластера	16.12.2023 - 12.01.2024	виконано
6	Проведення порівняльного аналізу метрик надійності та продуктивності	13.01.2024 - 26.01.2024	виконано

7	Розробка стартап-проекту та економічний аналіз	27.01.2024 - 09.02.2024	виконано
8	Оформлення пояснювальної записки та підготовка до захисту	10.02.2024 - 10.12.2024	виконано

Студент Данило КУЛИНЯК

Науковий керівник дисертації Володимир МАНЬКІВСЬКИЙ

РЕФЕРАТ

Робота виконана на 111 сторінках, містить 27 ілюстрацій, 15 таблиць. При підготовці використовувалася література з 29 різних джерел.

Актуальність: У сучасному світі підприємства все більше залежать від стабільності та продуктивності своїх IT-систем. Із розвитком контейнеризації виникла необхідність у спеціалізованих інструментах оркестрації контейнерів, серед яких провідне місце займає Kubernetes. Зростання складності інфраструктури зумовлює нові вимоги до надійності та доступності Kubernetes кластерів, що вимагає ретельного планування та оптимізації мережевої топології.

Мета роботи: аналіз методів і моделей, що впливають на надійність та доступність Kubernetes кластерів, зокрема з акцентом на мережеву топологію.

Задачі дослідження:

- Провести аналіз принципів побудови та функціонування Kubernetes кластерів
- Дослідити мережеві аспекти та показники надійності
- Розробити математичну модель мережевої топології
- Створити програмне рішення для аналізу різних топологій
- Провести порівняльний аналіз метрик надійності
- Розробити рекомендації щодо оптимізації топології

Об'єкт дослідження: процеси функціонування Kubernetes-кластерів, зокрема їхня мережева інфраструктура.

Предмет дослідження: методи та моделі оцінки та оптимізації мережевої топології Kubernetes-кластера.

Методи дослідження: теоретичний аналіз, математичне моделювання, теорія графів, статистичний аналіз, експериментальні дослідження.

Наукова новизна: розроблено математичну модель мережевої топології Kubernetes кластера та створено комплексний підхід до аналізу продуктивності різних топологій.

Практичне значення: результати роботи дозволяють оптимізувати мережеву топологію Kubernetes кластерів для підвищення їх надійності та доступності.

Ключові слова: KUBERNETES, МЕРЕЖЕВА ТОПОЛОГІЯ, НАДІЙНІСТЬ, ДОСТУПНІСТЬ, КОНТЕЙНЕРИЗАЦІЯ, ОРКЕСТРАЦІЯ, КЛАСТЕР, CONTROL PLANE.

ABSTRACT

The thesis is completed on 111 pages, contains 27 illustrations, 15 tables. The bibliography list consists of 29 sources.

Relevance: In today's world, enterprises increasingly depend on the stability and performance of their IT systems. With the development of containerization, there is a need for specialized container orchestration tools, among which Kubernetes holds a leading position. The growing complexity of infrastructure creates new requirements for the reliability and availability of Kubernetes clusters, which requires careful planning and optimization of network topology.

Objective: analysis of methods and models that affect the reliability and availability of Kubernetes clusters, particularly focusing on network topology.

Research tasks:

- Analyze the principles of building and operating Kubernetes clusters
- Research network aspects and reliability indicators
- Develop a mathematical model of network topology
- Create software solution for analyzing different topologies
- Conduct comparative analysis of reliability metrics
- Develop recommendations for topology optimization

Object of research: operational processes of Kubernetes clusters, particularly their network infrastructure.

Subject of research: methods and models for evaluating and optimizing the network topology of Kubernetes clusters.

Research methods: theoretical analysis, mathematical modeling, graph theory, statistical analysis, experimental research.

Scientific novelty: developed a mathematical model of Kubernetes cluster network topology and created a comprehensive approach to analyzing the performance of different topologies.

Practical significance: the results allow optimizing the network topology of Kubernetes clusters to increase their reliability and availability.

Keywords: KUBERNETES, NETWORK TOPOLOGY, RELIABILITY, AVAILABILITY, CONTAINERIZATION, ORCHESTRATION, CLUSTER, CONTROL PLANE.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API	(Application Programming Interface) - програмний інтерфейс додатку
CNI	(Container Network Interface) - мережевий інтерфейс контейнера
CPU	(Central Processing Unit) - центральний процесор
DNS	(Domain Name System) - система доменних імен
F2C	(Fog-to-Cloud) - від туманних до хмарних обчислень
HTTP	(HyperText Transfer Protocol) - протокол передачі гіпертексту
IP	(Internet Protocol) - інтернет протокол
IT	(Information Technology) - інформаційні технології
K8s	(Kubernetes) - система оркестрації контейнерів
MTBF	(Mean Time Between Failures) - середній час між відмовами
MTTF	(Mean Time To Failure) - середній час до відмови
MTTR	(Mean Time To Repair) - середній час до відновлення
MTTA	(Mean Time To Acknowledge) - середній час до підтвердження
NAT	(Network Address Translation) - перетворення мережевих адрес
OSI	(Open Systems Interconnection) - взаємодія відкритих систем
QoS	(Quality of Service) - якість обслуговування
RAM	(Random Access Memory) - оперативна пам'ять
REST	(Representational State Transfer) - передача репрезентативного стану
RF	(Resilience Factor) - коефіцієнт відмовостійкості
SLA	(Service Level Agreement) - угода про рівень обслуговування
SLI	(Service Level Indicator) - індикатор рівня обслуговування
SLO	(Service Level Objective) - ціль рівня обслуговування
SSL	(Secure Sockets Layer) - рівень захищених сокетів

TCP	(Transmission Control Protocol) - протокол керування передачею
UDP	(User Datagram Protocol) - протокол датаграм користувача
VXLAN	(Virtual Extensible LAN) - віртуальна розширювана локальна мережа

ВСТУП

У сучасному світі підприємства й організації все більше залежать від стабільності та продуктивності своїх IT-систем, адже цифрові технології стали невід'ємною частиною бізнес-процесів та надання послуг. Із розвитком контейнеризації, яка забезпечує стандартизацію, гнучкість і легкість у розгортанні додатків, виникла необхідність у спеціалізованих інструментах оркестрації контейнерів. Серед таких платформ провідне місце займає Kubernetes, який дозволяє автоматизувати процеси керування контейнерами, їх масштабування та балансування навантаження. Проте зростання складності інфраструктури зумовлює нові вимоги до надійності та доступності Kubernetes кластерів, що вимагає ретельного планування та оптимізації мережевої топології.

Kubernetes швидко стає новим стандартом індустрії для запуску хмарних додатків. Оскільки більше розробників та організацій відходять від локальної інфраструктури, щоб скористатися перевагами хмари, потрібне просунуте управління для забезпечення високої доступності та масштабованості контейнеризованих додатків. Щоб створювати та підтримувати додатки, необхідно координувати ресурси між різними типами машин, мережами та середовищами.

Kubernetes дозволяє розробникам контейнеризованих додатків—таких як ті, що створені за допомогою Docker—розробляти більш надійну інфраструктуру, що є критично важливим для додатків та платформ, які повинні відповідати на події, такі як швидкі сплески трафіку або необхідність перезапуску збоїв. З Kubernetes ви можете делегувати події, які б вимагали ручного втручання розробника на виклику.

Kubernetes (K8s) оптимізує розгортання та управління оркестрацією контейнерів для хмарної інфраструктури шляхом створення груп, або Pod-ів,

контейнерів, які масштабуються без необхідності написання додаткових рядків коду і відповідають на потреби додатку. Основними перевагами переходу до контейнеро-центричної інфраструктури з Kubernetes є впевненість у тому, що інфраструктура буде самовідновлюватись, а також забезпечення консистентності середовища від розробки до продукції.

Актуальність теми обумовлена потребою сучасних IT-систем у високій надійності, гнучкості та ефективності. Kubernetes, як платформа для оркестрації контейнерів, грає ключову роль у забезпеченні цих вимог. Мережева топологія є одним із головних чинників, що впливають на стабільність і продуктивність Kubernetes-кластерів, забезпечуючи ефективний розподіл трафіку, захист від відмов і оптимальне використання ресурсів. Оптимізація мережевої структури допомагає значно підвищити стійкість кластерів, мінімізувати простої та підвищити доступність критично важливих сервісів, що робить дослідження у цій сфері надзвичайно важливим.

Метою дослідження є аналіз методів і моделей, що впливають на надійність та доступність Kubernetes кластерів, зокрема з акцентом на мережеву топологію. У рамках дослідження буде розглянуто, як правильно спроектована мережева інфраструктура підвищує стійкість Kubernetes до відмов, забезпечує оптимізацію ресурсів та покращує продуктивність системи в цілому.

Об'єктом дослідження є процеси функціонування Kubernetes кластерів, зокрема їхня мережева інфраструктура, що включає такі компоненти як control plane, вузли (nodes), поди (pods) та інші сервіси, які забезпечують безперервну взаємодію в кластері.

Предметом дослідження є методи та моделі, які дозволяють оцінити та оптимізувати мережеву топологію Kubernetes кластера для забезпечення його надійності та доступності, з урахуванням механізмів балансування

навантаження, планування pod'ів, обмежень на ресурси та інших факторів впливу.

Методологічна база дослідження включає теоретичний аналіз сучасних підходів до побудови мережевої топології Kubernetes-кластерів, математичні методи моделювання надійності та доступності, а також практичні методи для збору та обробки даних, моніторингу продуктивності та тестування програмних рішень. Застосування математичного апарату для моделювання мережевої інфраструктури, розробка моделей оцінки продуктивності та інструментів для аналізу є основними методами дослідження.

Наукова новизна роботи полягає у розробці математичної моделі мережевої топології Kubernetes кластера, яка дозволяє оцінити його надійність і доступність, враховуючи ключові мережеві параметри. У роботі також вперше розглядається комплексний підхід до створення програмного інструменту для аналізу мережевої структури, який дозволяє автоматизувати оцінку продуктивності та виявляти потенційні ризики в мережевій інфраструктурі Kubernetes.

Результатом дипломної роботи є розробка рекомендацій та інструментарію для побудови надійної мережевої топології Kubernetes кластера, що забезпечує високу доступність і продуктивність системи. Запропонована модель і програмне рішення дозволяють автоматизувати процеси моніторингу та аналізу мережевої структури, виявляти вузькі місця та підвищувати ефективність використання ресурсів у Kubernetes кластерах.

РОЗДІЛ 1
АНАЛІЗ ПРИНЦИПІВ ПОБУДОВИ ТА ФУНКЦІОНУВАННЯ KUBERNETES КЛАСТЕРІВ

1.1. Архітектура Kubernetes

Kubernetes — це система оркестрації контейнерів, яку спочатку розробила компанія Google для масштабування контейнеризованих додатків у хмарі. Kubernetes може керувати життєвим циклом контейнерів, створюючи та знищуючи їх залежно від потреб додатку, а також надаючи безліч інших функцій. Kubernetes став однією з найбільш обговорюваних концепцій у розробці хмарних додатків, а його зростання сигналізує про зміну підходу до розробки та розгортання додатків.

Загалом, Kubernetes складається з кластера серверів, які називаються вузлами (Nodes), кожен з яких запускає агентні процеси Kubernetes та спілкується з іншими. Головний вузол (Master Node) містить колекцію процесів, які називаються контрольною площиною (control plane), що допомагають впровадити та підтримувати бажаний стан кластера Kubernetes, у той час як робочі вузли (Worker Nodes) відповідають за запуск контейнерів, які складають ваші додатки та сервіси.

Kubernetes — це інструмент оркестрації **контейнерів**, тому для його роботи необхідно встановити середовище виконання контейнерів (container runtime). На практиці, за замовчуванням контейнерним середовищем для Kubernetes є Docker, хоча інші середовища, такі як rkt та LXD, також можуть використовуватися. З появою інтерфейсу середовища виконання контейнерів (CRI), що прагне стандартизувати взаємодію Kubernetes з контейнерами, стали доступні й інші варіанти, такі як containerd, cri-o та Frakti.

Контейнери подібні до віртуальних машин. Вони є легковаговими ізольованими середовищами виконання, які використовують ресурси операційної системи, не потребуючи запуску повної операційної системи. Контейнери споживають мало ресурсів, але містять повний набір інформації, необхідної для виконання вбудованих у них додатків, таких як файли, змінні середовища та бібліотеки.

Контейнеризація — це метод віртуалізації для запуску розподілених додатків у контейнерах із використанням мікросервісів. Контейнеризація додатку вимагає базового образу для створення екземпляра контейнера. Як тільки образ додатку існує, його можна завантажити в централізований реєстр контейнерів, який Kubernetes може використовувати для розгортання екземплярів контейнерів у Pod-ах кластера

Оркестрація — це автоматизована конфігурація, координація та управління комп'ютерними системами, програмним забезпеченням, проміжним програмним забезпеченням і сервісами. Вона використовує автоматизовані завдання для виконання процесів. Для Kubernetes оркестрація контейнерів автоматизує всі аспекти забезпечення, розгортання та доступності контейнерів; балансування навантаження; розподіл ресурсів між контейнерами; а також моніторинг здоров'я кластера. **[1]**

Архітектура Kubernetes має багато компонентів. Насамперед, слід розглянути кластер, який являє собою сукупність хостів та мережевих ресурсів. Кластер Kubernetes може бути однонодовим, що зазвичай використовується для тестування та навчання. Однак навіть для найпростішого робочого середовища рекомендується мати кластер з одним майстер-вузлом та одним робочим вузлом. Для продуктивного використання бажано використовувати кластер з одним майстер-вузлом і трьома робочими вузлами.

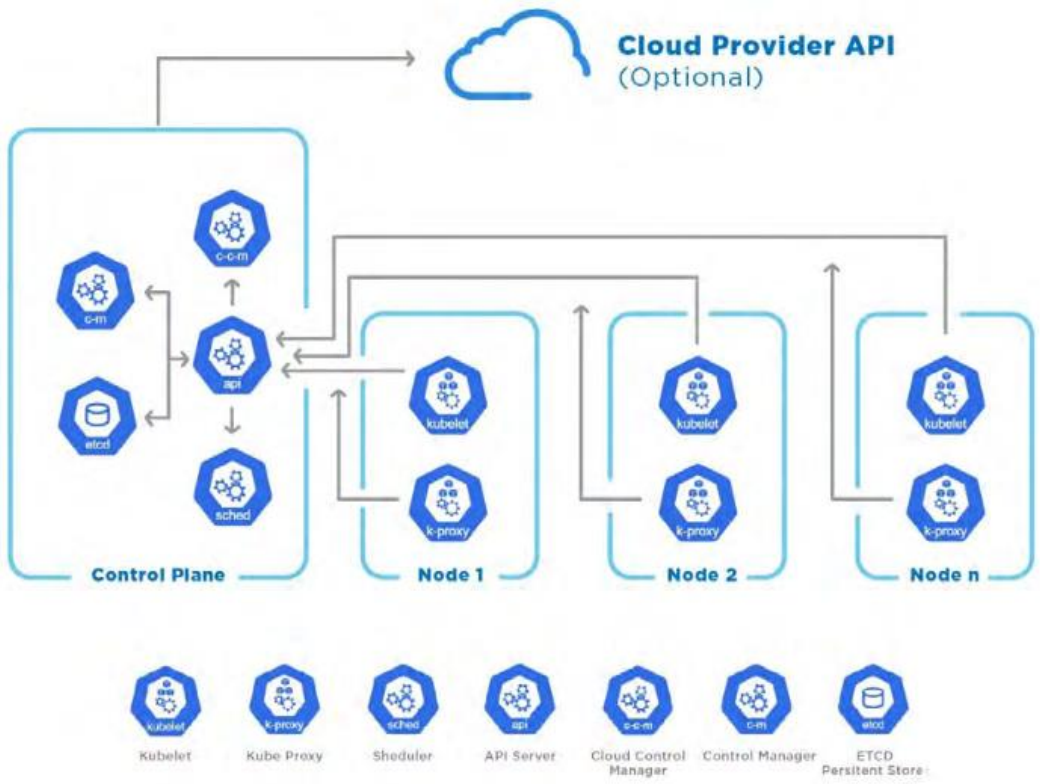

Рис. 1.1 Kubernetes

Kubelet — це основний процес Kubernetes, який забезпечує взаємодію між вузлами кластера, дозволяє їм спілкуватися між собою і виконувати дії на цих вузлах, такі як запуск процесів додатків. Кожен робочий вузол містить контейнери для різних розгорнутих додатків, тому користувач може мати різноманітні Docker-контейнери, що працюють на робочих вузлах, залежно від того, як розподілено навантаження. Основна робота виконується саме на робочих вузлах, а додатки користувача запускаються саме на них.

Тепер слід розглянути майстер-вузли. Майстер-вузли запускають численні процеси Kubernetes, необхідні для належного функціонування та управління кластером. Одним із цих процесів є API-сервер, який також працює в контейнері. API-сервер є точкою входу до кластера Kubernetes, що дозволяє іншим клієнтам Kubernetes взаємодіяти між собою, наприклад, через графічний інтерфейс, якщо

користувач використовує панель керування Kubernetes, або через API, якщо користувач застосовує скрипти та технології автоматизації, а також командний рядок.

Контрольна площина (control plane) є важливою частиною архітектури Kubernetes. Вона складається з різних компонентів, таких як API-сервер, менеджер контролерів, планувальник та інші, які керують станом кластера, розподіляють завдання, відстежують події та забезпечують належне функціонування системи. Компоненти контрольної площини взаємодіють між собою та з вузлами кластера для забезпечення стабільної роботи додатків.

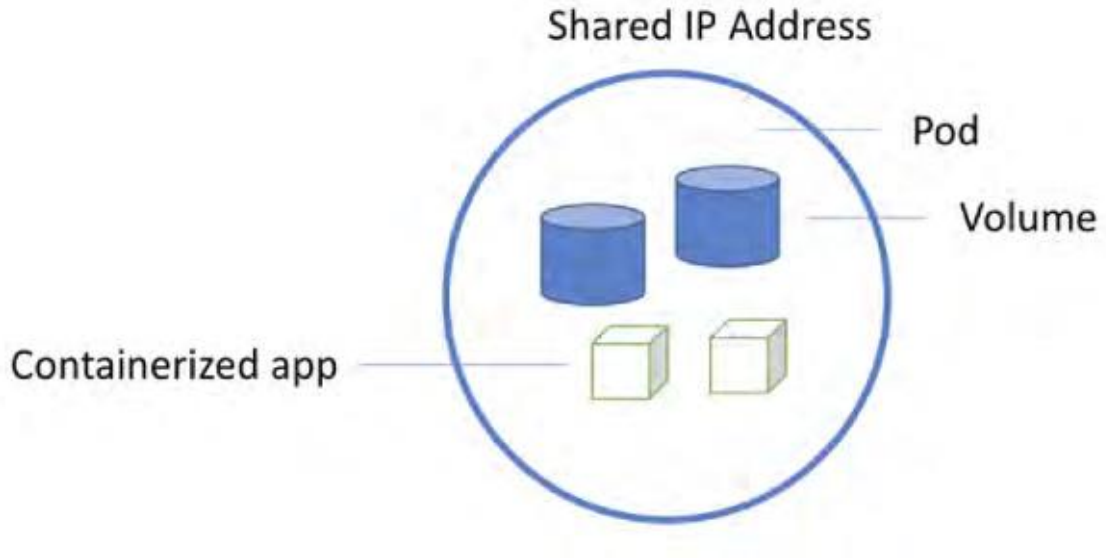

Рис. 1.2 Kubernetes поди

Kube-API-сервер керує з'єднанням між вузлами та контролерами. Коли до кластера додаються нові вузли, API-сервер взаємодіє з ними та масштабує кластер. Він надсилає команди вузлам і відстежує їх поведінку, а також розподіляє навантаження на введення/виведення.

Іншим важливим процесом є менеджер контролерів (controller manager), який працює на майстер-вузлах. Він відповідає за відстеження подій у кластері, наприклад, визначення, чи потрібно щось виправити, чи контейнер помер і потребує перезапуску тощо. Ще одним важливим компонентом є планувальник (scheduler), який відповідає за планування розміщення контейнерів на вузлах залежно від навантаження та доступних ресурсів сервера. Це інтелектуальний процес, який визначає, на якому з робочих вузлів буде розміщено наступний контейнер, враховуючи наявні ресурси вузлів і вимоги до навантаження контейнерів. Існують різні типи контролерів:

- **Контролери вузлів**: відповідають за виявлення відмов і вжиття відповідних заходів.
- **Контролер завдань (Job Controller)**: слідкує за об'єктами Job, що представляють спорадичні завдання, і створює Pod-и для їх виконання.
- **Контролер кінцевих точок (Endpoints Controller)**: додає об'єкти до об'єкта Endpoints, об'єднуючи сервіси та Pod-и.
- **Контролер облікових записів та токенів (Service Account & Token Controllers)**: генерує токени доступу до API та облікові записи за замовчуванням для нових просторів імен.

Etcd — це система зберігання даних у вигляді ключ-значення, яка є важливим компонентом всього кластера. Вона містить актуальний стан кластера Kubernetes на будь-який момент часу. Таким чином, Etcd включає всю конфігурацію даних та статус кожного вузла і контейнера в кластері. **[2]**

У багатьох випадках Kubernetes працює в хмарі, і для цього постачальник хмарних послуг додає свою програму до кластера для керування кластерами та забезпечення клієнтських запитів на вимогу. Реалізації для різних хмарних

провайдерів можуть відрізнятися. Cloud-Controller Manager містить такі компоненти:

- **Контролер вузлів**: перевіряє постачальника хмарних послуг, якщо вузол перестає відповідати, щоб з'ясувати, чи був він видалений.
- **Контролер маршрутизації (Route Controller)**: керує внутрішньою маршрутизацією.
- **Контролер сервісів (Service Controller)**: керує сервісами Kubernetes.

Останнім, але не менш важливим компонентом є віртуальна мережа, яка дозволяє робочим і майстер-вузлам взаємодіяти один з одним. У спрощених термінах віртуальна мережа перетворює всі вузли в кластері на потужну машину, яка має сумарні ресурси всіх вузлів. Отже, архітектура Kubernetes включає в себе ці ключові компоненти, що працюють разом, забезпечуючи ефективне управління контейнеризованими додатками.

Кожен вузол має певні служби, які працюють разом із застосунками Kubernetes. Вони взаємодіють із майстер-вузлом або вузлами для отримання завдань та інструкцій. Вони відстежують використання ресурсів Pod-ами та відправляють ці дані на майстер. Якщо щось іде не так, вузол повідомляє про це майстер-вузлу.

Kube-proxy забезпечує мережеву взаємодію між сервісами Kubernetes. Він налаштовує мережеві правила та протоколи між вузлами, що дозволяє надавати доступ до контейнерів, незважаючи на складність мережевої архітектури.

Kubernetes підтримує різні середовища виконання контейнерів, окрім Docker, такі як containerd, CRI-O тощо. Користувачі можуть вибрати, яке середовище виконання вони хочуть використовувати для свого застосунку.

Для доступу до застосунків, що працюють у Kubernetes, використовуються сервіси, такі як Node Port, Load Balancer, Cluster IP та Ingress. Наприклад, Ingress використовується для використання повного доменного імені (FQDN) для застосунку.

1.2. Взаємодія компонентів системи Kubernetes

Взаємодія між компонентами Kubernetes є основою для забезпечення стабільної роботи кластера. Кожен компонент виконує чітко визначену роль, і всі компоненти разом формують складну інфраструктуру, яка забезпечує управління життєвим циклом подів, балансування навантаження, моніторинг і збереження конфігурацій кластера. Взаємодія між ними відбувається через API-запити, обмін даними та синхронізацію статусу об'єктів кластера. **[2]**

Компоненти Kubernetes поділяються на дві основні групи: **Control Plane** та **Node Level**.

- **Control Plane** — це керуючий рівень, який моніторить стан кластера, призначає поди, управляє конфігураціями та координує роботу компонентів. До нього входять:
    - **kube-apiserver**: основний інтерфейс для комунікації всередині кластера, приймає та обробляє запити від інших компонентів та користувачів.
    - **etcd**: розподілене сховище ключ-значення, яке зберігає конфігурації та дані про стан кластера.
    - **kube-scheduler**: відповідає за призначення подів на вузли з урахуванням ресурсів, доступних на кожному з них.
    - **kube-controller-manager**: контролює стан об'єктів кластера та синхронізує його із заданими конфігураціями.

- **Node Level** — рівень вузлів, де розміщуються поди та контейнери. Основні компоненти:
    - kubelet: взаємодіє з API-сервером, запускає поди на вузлі та моніторить їх стан.
    - kube-proxy: забезпечує маршрутизацію трафіку між подами, підтримуючи зв’язність та балансування навантаження.

Механізми взаємодії компонентів:

1. **API-запити через kube-apiserver**. Взаємодія між компонентами Kubernetes відбувається через kube-apiserver, що є центральним інтерфейсом для комунікації між Control Plane і вузлами. Наприклад, при створенні нового pod'a запит надходить до API-сервера, де зберігається його стан. Потім kube-scheduler визначає найкращий вузол для пода і передає відповідні інструкції kubelet.
2. **Сховище даних etcd**. Всі конфігураційні дані зберігаються в etcd, який забезпечує надійне зберігання інформації про стан об’єктів кластера. Коли змінюється стан об'єкта (наприклад, пода або сервісу), дані автоматично оновлюються в etcd, що дозволяє іншим компонентам синхронізуватися зі змінами.
3. **Планування подів за допомогою kube-scheduler**. kube-scheduler призначає поди на вузли кластера, отримуючи дані про нові поди через kube-apiserver. Після того як вузол для пода обраний, kube-scheduler повідомляє про це API-сервер, а kubelet на відповідному вузлі виконує запит на запуск пода.
4. **Взаємодія kubelet з контейнерним runtime**. Kubelet на кожному вузлі виконує інструкції Control Plane, взаємодіючи з контейнерним runtime

(Docker, containerd). kubelet створює і запускає контейнери всередині подів відповідно до вимог і передає звіт про стан контейнерів API-серверу.

5. **Маршрутизація та балансування навантаження через kube-proxy** kube-proxy налаштовує маршрутизацію трафіку між подами і сервісами кластера, використовуючи iptables або IPVS. Це забезпечує балансування навантаження між подами одного сервісу, дозволяючи рівномірний розподіл трафіку та забезпечення високої доступності.

Додаткові компоненти, такі як DNS та Ingress controllers, також забезпечують ключові функції для зручності доступу до подів і сервісів **[3]**:

- **DNS**: надає можливість звертатися до подів і сервісів за DNS-іменами. Kubernetes автоматично створює DNS-записи для нових сервісів, що спрощує комунікацію між додатками всередині кластера.
- **Ingress controllers**: забезпечують зовнішній доступ до сервісів кластера, керуючи маршрутизацією запитів ззовні до подів усередині кластера. Ingress controllers також підтримують SSL-термінацію, що підвищує безпеку при обміні даними з зовнішніми клієнтами.

Приклад життєвого циклу запиту через основні компоненти Kubernetes:

1. Користувач або інший компонент Kubernetes (наприклад, контролер) надсилає запит до kube-apiserver.
2. API-сервер перевіряє конфігурацію запиту та передає його до відповідного компонента. Наприклад, при створенні нового пода запит надходить до kube-scheduler, який призначає вузол для пода.
3. kubelet на обраному вузлі запускає под і постійно оновлює його стан на API-сервері.

4. kube-proxy налаштовує правила маршрутизації для зв’язку подів та балансування трафіку, дозволяючи доступ до пода відповідно до політик сервісу.
5. DNS або Ingress controller дозволяють додаткам звертатися до подів за встановленими іменами або маршрутами.

Завдяки такій чіткій взаємодії компонентів Kubernetes забезпечує стабільність роботи кластера, автоматизує життєвий цикл подів та ефективно керує навантаженням між компонентами.

Життєвий цикл pod'ів і сервісів у Kubernetes є важливою частиною роботи системи, яка забезпечує стабільність, надійність та автоматизацію процесів розгортання і управління контейнеризованими додатками. Kubernetes спрощує ці процеси завдяки автоматизованій системі оркестрації, яка відповідає за створення, моніторинг і завершення pod'ів та підтримку стабільних точок доступу через сервіси.

Pod є базовою одиницею розгортання в Kubernetes. Він складається з одного або кількох контейнерів, що працюють разом, поділяючи між собою мережеві і файлові ресурси. Життєвий цикл pod'ів починається з етапу **Pending** (очікування), коли pod створений, але ще не розміщений на вузлі, оскільки система планування ще не вибрала оптимальний вузол для його запуску. Потім pod переходить у стан **Running**, коли контейнери всередині нього успішно запущені.

У разі завершення роботи пода або його успішного виконання, pod переходить у стан **Succeeded**. Якщо pod завершився через помилку, він отримує статус **Failed**. У деяких випадках Kubernetes може виявляти pod'и, які постійно завершують роботу з помилкою і перезапускаються — для таких pod'ів надається

стан **CrashLoopBackOff**, що сигналізує про проблему і вводить паузу перед повторним запуском.

Сервіси у Kubernetes створюються для того, щоб надавати стабільну точку доступу до груп подів і підтримувати зв'язок між компонентами системи. Коли сервіс створений, він реєструється в API-сервері, і kube-proxy на кожному вузлі налаштовує маршрутизацію для зв’язку з подами, що відповідають критеріям вибору цього сервісу. Це дозволяє сервісу динамічно оновлювати список доступних подів у разі їх створення чи видалення, підтримуючи стабільність з’єднання.

Сервіс надає статичну IP-адресу (ClusterIP), яка забезпечує доступ до групи подів незалежно від їх розташування на вузлах. У разі змін у складі подів сервіс автоматично оновлює свої кінцеві точки, що дозволяє зберігати стабільний доступ до додатків навіть при зміні конфігурації. Коли сервіс більше не потрібен, його можна видалити, після чого всі пов’язані маршрути й кінцеві точки видаляються.

Поди та сервіси у Kubernetes функціонують у тісному зв’язку, щоб забезпечити безперебійне управління і доступність додатків. Сервіси надають стабільний зв'язок між подами, навіть якщо окремі поди перезапускаються або переміщуються на інші вузли, що гарантує надійну маршрутизацію та балансування навантаження. Завдяки цьому Kubernetes забезпечує гнучкість та високу доступність, дозволяючи динамічно масштабувати додатки, автоматично замінюючи поди у разі збою, і зберігаючи стабільний зв’язок між компонентами.

Kubernetes забезпечує потужні механізми оркестрації та балансування навантаження, які дозволяють ефективно розподіляти ресурси і підтримувати високу доступність додатків. Оркестрація контейнерів в Kubernetes включає автоматичне розгортання, масштабування, оновлення та відновлення подів у разі

збоїв. Балансування навантаження, в свою чергу, гарантує рівномірний розподіл запитів між подами, забезпечуючи стабільну продуктивність і запобігаючи перевантаженню окремих вузлів.

Kubernetes автоматично керує життєвим циклом подів завдяки функціям планування, масштабування та відновлення:

- **Планування подів**: kube-scheduler аналізує потреби кожного пода у ресурсах (процесор, пам'ять) і вибирає найбільш відповідний вузол, враховуючи доступність ресурсів, обмеження політик та поточне навантаження на вузли. Це дозволяє Kubernetes оптимально використовувати ресурси кластера.
- **Масштабування**: Kubernetes підтримує ручне, автоматичне та горизонтальне масштабування подів. Ручне масштабування виконується шляхом зміни кількості подів вручну, автоматичне регулює кількість подів на основі показників використання ресурсів (наприклад, CPU або пам'ять), а горизонтальне масштабування дозволяє розширювати кластер за рахунок додавання нових подів для обслуговування зростаючого навантаження.
- **Відновлення подів**: Kubernetes автоматично контролює стан подів і перезапускає їх у разі збоїв, забезпечуючи високу доступність додатків. Якщо под припиняє роботу через помилку або проблеми на вузлі, Kubernetes відновлює його або створює новий под на іншому вузлі, гарантуючи безперервність роботи.

Kubernetes надає кілька рівнів балансування навантаження для ефективного розподілу трафіку між подами, що допомагає уникнути перевантаження окремих компонентів і покращує продуктивність додатків.

Внутрішньокластерне балансування: Всі поди, які обслуговуються одним сервісом, отримують стабільну IP-адресу (ClusterIP). kube-proxy, працюючи на кожному вузлі, налаштовує правила маршрутизації, які дозволяють рівномірно розподіляти трафік між подами сервісу. Це забезпечує внутрішнє балансування запитів, яке автоматично оновлюється при зміні складу подів.

Використання Ingress для зовнішнього трафіку: Ingress надає механізм для керування зовнішнім доступом до сервісів у кластері. Ingress controllers виконують балансування трафіку ззовні на основі правил маршрутизації, таких як HTTP-шляхи або доменні імена, і можуть також підтримувати SSL-термінацію для захищеного з'єднання.

NodePort і LoadBalancer: NodePort дозволяє відкрити доступ до сервісу на певному порту кожного вузла, а LoadBalancer інтегрує кластер із зовнішнім балансувальником навантаження в хмарному середовищі (наприклад, AWS або Google Cloud). Використання LoadBalancer забезпечує розподіл запитів між вузлами, що дозволяє краще обробляти великі обсяги трафіку ззовні.

Механізми оркестрації та балансування навантаження у Kubernetes сприяють підвищенню стабільності, адаптивності та продуктивності додатків. Це дозволяє Kubernetes автоматично реагувати на зміни у навантаженні, підтримувати оптимальне використання ресурсів і забезпечувати доступність додатків навіть при великих навантаженнях.

Архітектура Kubernetes представляє собою потужну та гнучку систему для оркестрації контейнеризованих додатків, що дозволяє ефективно керувати життєвим циклом подів, забезпечувати високу доступність і продуктивність додатків. Завдяки модульному підходу, Kubernetes розділений на два основних рівні: **Control Plane** та **Node Level**. Кожен рівень має чітко визначені компоненти,

які відповідають за конкретні функції, що дозволяє гнучко масштабувати і керувати ресурсами.

Control Plane виконує центральні завдання, включаючи управління станом кластера, прийом і обробку запитів, зберігання конфігураційних даних та розміщення подів на вузлах. kube-apiserver служить основним інтерфейсом для комунікації між компонентами, забезпечуючи синхронізацію і доступ до даних, збережених у etcd. Завдяки kube-scheduler, Kubernetes аналізує ресурси і вимоги до подів, обираючи оптимальні вузли для їхнього розміщення, що дозволяє забезпечити рівномірне навантаження і ефективне використання ресурсів кластера.

На рівні вузлів (**Node Level**) kubelet виконує інструкції Control Plane і відповідає за запуск подів і моніторинг контейнерів, забезпечуючи стабільне функціонування додатків. kube-proxy здійснює маршрутизацію трафіку між подами та сервісами, налаштовуючи балансування навантаження всередині кластера. Додаткові компоненти, такі як DNS та Ingress controllers, підвищують зручність доступу до додатків і дозволяють ефективно керувати зовнішнім трафіком, забезпечуючи захищене з’єднання та маршрутизацію.

Завдяки механізмам оркестрації Kubernetes автоматично управляє подами, забезпечуючи масштабування і відновлення в разі збоїв. Механізми балансування навантаження, що працюють як на внутрішньому рівні (через ClusterIP та kube-proxy), так і на зовнішньому (через Ingress), забезпечують стабільний розподіл трафіку і зменшують ризик перевантаження окремих компонентів. Це дозволяє кластеру ефективно адаптуватися до змін у навантаженні і підтримувати високу доступність додатків.

Загалом, архітектура Kubernetes забезпечує високий рівень автоматизації для керування життєвим циклом додатків, пропонуючи інструменти для

масштабування, балансування навантаження та відновлення після збоїв. Це робить Kubernetes ідеальною платформою для побудови надійних, високопродуктивних і легко масштабованих додатків у контейнеризованих середовищах, що є критично важливим для сучасних IT-інфраструктур.

1.3. Мережеві аспекти Kubernetes

Мережева інфраструктура є одним із ключових аспектів, що забезпечує стабільну і продуктивну роботу Kubernetes-кластерів. Оскільки Kubernetes призначений для розподілених додатків, які функціонують у контейнерах, мережа виступає важливим компонентом, що забезпечує зв'язок між подами, сервісами та зовнішніми користувачами. Правильна налаштовка мережевої топології дозволяє підвищити ефективність і стійкість до збоїв, забезпечуючи стабільний доступ до додатків навіть у разі масштабування або зміни конфігурації кластера.

Мережева модель Kubernetes підтримує різні підходи до організації комунікацій, включаючи мережеві політики, под-мережі, сервісні мережі, а також використання контейнерних мережевих інтерфейсів (CNI) для інтеграції з різноманітними мережевими плагінами. Такі мережеві плагіни, як Calico, Flannel та Weave, забезпечують додаткові можливості для оптимізації трафіку, захисту мережевих з'єднань та реалізації мережевих політик. Крім того, використання overlay-мереж (наприклад, VXLAN, IP-in-IP) дозволяє Kubernetes створювати віртуальні мережі поверх фізичної інфраструктури, що підвищує гнучкість у побудові кластера та забезпечує ефективний розподіл мережевого навантаження.

Таким чином, розуміння мережевих аспектів Kubernetes є важливим для забезпечення надійності та безперервності роботи додатків, що працюють у контейнеризованих середовищах. У цьому розділі розглядаються основні

елементи мережевої архітектури Kubernetes, типи мережевих моделей та інструменти для управління мережевою інфраструктурою, що є критично важливими для підтримки високої доступності та продуктивності кластера.

Інтернет, який ми знаємо сьогодні, є величезним: кабелі простягаються через океани і гори, з'єднуючи міста з нижчою затримкою, ніж будь-коли раніше. Метою будь-якої мережі є обмін інформацією від однієї системи до іншої. Це величезне завдання для розподіленої глобальної системи, але Інтернет не завжди був глобальним; він починався як концептуальна модель і поступово розвивався, доки не став тим гігантом. Вивчаючи основи мереж, потрібно враховувати багато факторів, таких як остання миля — з'єднання між домівкою користувача і мережею його провайдера, аж до геополітичного ландшафту Інтернету. Інтернет став невід'ємною частиною нашого суспільства.

У своїх найперших формах мережі керувалися або спонсорувалися урядом; у Сполучених Штатах Міністерство оборони (DOD) фінансувало створення Мережі агентства передових досліджень (ARPANET) задовго до того, як Аль Гор почав свою політичну діяльність, про що буде йтися далі. У 1969 році ARPANET була розгорнута в Каліфорнійському університеті в Лос-Анджелесі, Центрі досліджень розширення в Стенфордському дослідницькому інституті, Каліфорнійському університеті в Санта-Барбарі та Школі обчислювальної техніки Університету Юти. Обмін даними між цими вузлами був завершений тільки в 1970 році, коли вони почали використовувати Протокол управління мережею (NCP). NCP сприяв розробці та використанню перших протоколів для зв'язку між комп'ютерами, таких як Telnet і File Transfer Protocol (FTP).

Успіх ARPANET і NCP, першого протоколу для роботи ARPANET, врешті-решт призвів до його заміни: він не міг справлятися з попитом мережі та різноманітністю підключених мереж. У 1974 році Вінт Серф, Йоген Далал і Карл

Саншайн почали створювати RFC 675 для Протоколу управління передачею (TCP**). [4]**

TCP став стандартом для мережевих з'єднань, дозволяючи обмін пакетами між різними типами мереж. У 1981 році Інтернет-протокол (IP), визначений у RFC 791, допоміг розподілити обов’язки TCP на окремий протокол, збільшивши модульність мережі. У наступні роки багато організацій, включно з Міністерством оборони США, прийняли TCP як стандарт. До січня 1983 року TCP/IP став єдиним затвердженим протоколом у ARPANET, замінивши попередній NCP завдяки своїй універсальності та модульності.

Конкуруюча організація зі стандартизації, Міжнародна організація зі стандартизації (ISO), розробила та опублікувала ISO 7498, «Еталонну модель відкритих систем» (Open Systems Interconnection Reference Model), що описує модель OSI. Разом із публікацією були створені протоколи для її підтримки. На жаль, протоколи моделі OSI так і не набули популярності, поступившись популярності TCP/IP. Проте модель OSI залишається відмінним навчальним інструментом для розуміння багаторівневого підходу до мереж.

У 1991 році Аль Гор «винайшов» інтернет (насправді, він допоміг ухвалити Закон про національну інформаційну інфраструктуру [NII]), що зрештою сприяло створенню Робочої групи з інженерії інтернету (IETF). На сьогоднішній день стандарти для інтернету перебувають під управлінням IETF, відкритого консорціуму провідних експертів і компаній у галузі мережевих технологій, таких як Cisco та Juniper. RFC публікуються Інтернет-товариством (Internet Society) та Робочою групою з інженерії інтернету. RFC зазвичай авторизуються окремими інженерами або групами інженерів та комп'ютерних науковців, детально описуючи процеси, операції та застосування для функціонування інтернету.

Використання хмарних технологій та їх сервісів значно зросло: 77% підприємств використовують публічну хмару в тій чи іншій формі, і 81% можуть швидше впроваджувати інновації за допомогою публічної хмари, ніж у локальних середовищах. Завдяки популярності та інноваційним можливостям хмари, запуск Kubernetes у хмарному середовищі є логічним кроком.

Модель OSI є відомою багаторівневою моделлю, розробленою для систематизації процесів мережевої взаємодії між різними компонентами і технологіями. Ця модель розділяє складну мережеву архітектуру на окремі рівні, кожен з яких виконує свою функцію у забезпеченні обміну даними між системами. При адаптації моделі OSI до Kubernetes і контейнеризованих середовищ вона виступає основою для структурованого розуміння різних рівнів, від апаратної інфраструктури до управління контейнерами, що робить її корисним інструментом для аналізу кластерної архітектури.

Кожен рівень у адаптованій OSI-моделі для Kubernetes визначає певний набір компонентів і технологій, які відповідають за окремі аспекти функціонування кластера. Така структура дозволяє глибше зрозуміти, як різні системні і програмні елементи взаємодіють, щоб забезпечити стабільну, масштабовану і продуктивну роботу кластера. Цей підхід допомагає відстежувати взаємозв'язок між різними частинами інфраструктури, починаючи від фізичних ресурсів (як-от CPU, пам’ять і мережеві підключення) на базовому рівні, і закінчуючи високорівневими технологіями оркестрації контейнерів, які забезпечують безперервність і автоматизацію розгортання додатків.

Модель OSI для Kubernetes є не лише корисним інструментом для адміністраторів, які хочуть оптимально налаштувати кластер, але й служить ефективною основою для інтеграції нових технологій, таких як інтерфейси контейнерних мереж (CNI) та інтерфейси контейнерного зберігання (CSI). Вона

також дозволяє краще зрозуміти механізми управління та узгодження стану кластера, як, наприклад, etcd, який відповідає за збереження конфігураційних даних у розподіленій системі. Таким чином, використання OSI-моделі в контексті Kubernetes забезпечує ефективне управління як фізичними, так і логічними аспектами кластерної архітектури, а також оптимізує роботу кластера, зберігаючи його стабільність та гнучкість для масштабування.

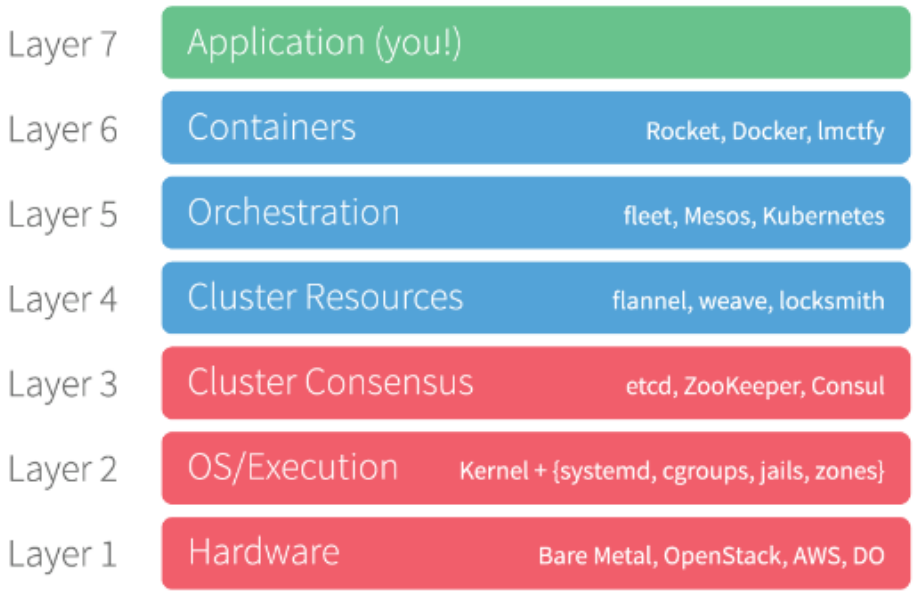

Рис 1.3 Адаптована модель OSI для Kubernetes. **[5]**

Фізичний рівень (1) — сервери, віртуальні машини та хмара. На цьому рівні розташовані фізичні сервери, віртуальні машини або хмарні інфраструктури, які забезпечують базові ресурси, такі як процесор (CPU), оперативна пам'ять (RAM), дискове сховище і мережа. Саме ці ресурси забезпечують основу для функціонування всього кластера Kubernetes.

Операційна система та ABI (Application Binary Interface) (2). На цьому рівні працює операційна система, інтерфейс прикладного програмного забезпечення (ABI), що забезпечує взаємодію між системними викликами, драйверами пристроїв та ядром ОС. Такі ОС, як RHEL і CoreOS, використовують системні простори (cgroups, namespaces) та різні ізоляційні технології (jails, zones), які забезпечують управління ресурсами та безпеку.

Консенсус у кластері (3). Консенсус є критично важливим для підтримки єдиного стану в розподіленій системі. Основними інструментами цього рівня є etcd, Apache ZooKeeper і consul. Наприклад, etcd — це база даних консенсусу, створена за алгоритмом Raft, яка дозволяє розподілено зберігати інформацію про конфігурацію, стан і метадані всередині кластера та контролювати зміни даних. Це дозволяє забезпечити цілісність даних і синхронізацію в межах кластера.

Ресурси кластера (4). На цьому рівні працюють мережеві (CNI) та системи зберігання (CSI) інтерфейси, які забезпечують управління мережевими ресурсами та сховищем даних у кластері. Ці інструменти дозволяють Kubernetes динамічно керувати мережевими та дисковими ресурсами, що є критичними для масштабованості та стабільності кластерів.

Оркестрація контейнерів (CO) (5). Цей рівень відповідає за управління контейнеризованими додатками, їх автоматизоване розгортання та балансування навантаження. Основними платформами цього рівня є Kubernetes, OpenShift і CloudFoundry, які дозволяють створювати, керувати і масштабувати контейнери в межах кластеру.

Контейнери (6). Цей рівень охоплює контейнери, такі як Docker, Podman, Rocket, CRIO, containerd, що забезпечують ізоляцію додатків і управління їхніми середовищами. Контейнери працюють незалежно, що дозволяє забезпечувати гнучкість і швидке розгортання додатків.

Проєкт etcd має на меті безпечно зберігати критичні дані розподіленої системи. Створений командою CoreOS у 2013 році, etcd був натхненний Chubby, системою сховища ключ-значення, розробленою для внутрішньої кластерної інфраструктури Google. У грудні 2018 року CoreOS і Red Hat передали etcd до CNCF, що підтверджує його значення для сучасних кластерних систем.

Pod Networking у Kubernetes визначає комунікацію між подами в межах кластера. Кожен pod отримує власну IP-адресу, яка дозволяє йому напряму обмінюватися даними з іншими подами, без необхідності налаштовувати додаткові мережеві засоби. Це спрощує комунікацію всередині кластера, забезпечуючи можливість з'єднання між усіма подами, що відповідає концепції «мережа рівноправних учасників». Важливою складовою под-мереж є використання **Container Network Interface (CNI)**, що дозволяє Kubernetes інтегрувати різні мережеві плагіни (наприклад, Calico, Flannel або Weave), забезпечуючи налаштування мережевих ресурсів.

Pod Networking базується на кількох основних принципах:

1. Кожен pod має свою IP-адресу, тому контейнери всередині одного pod'a можуть взаємодіяти за локальною адресою, а різні pod'и — за їх окремими IP-адресами.
2. Поди можуть комунікувати один з одним у межах кластера без NAT.
3. Мережа повинна бути зручна для автоматизації і підтримки, що досягається завдяки інтеграції мережевих плагінів.

Цей підхід до под-мереж дозволяє підтримувати масштабованість і динамічність Kubernetes, забезпечуючи безперервність з’єднання між подами навіть при їх переміщенні або створенні нових.

Service Networking у Kubernetes вирішує завдання забезпечення стабільного доступу до подів, навіть якщо окремі поди додаються або

видаляються. Сервіси у Kubernetes створюють віртуальну IP-адресу (ClusterIP), через яку можна звертатися до набору подів, незалежно від їхнього фізичного розташування в кластері. Це дозволяє організувати балансування навантаження і забезпечити стійкість до відмов. **[6]**

Основні типи сервісів у Kubernetes включають:

- **ClusterIP**: забезпечує внутрішній доступ до сервісу всередині кластера.
- **NodePort**: відкриває порт на кожному вузлі для доступу до сервісу ззовні.
- **LoadBalancer**: інтегрує кластер із зовнішнім балансувальником навантаження в хмарному середовищі (як-от AWS чи GCP).
- **ExternalName**: дозволяє перенаправляти запити на зовнішні сервіси за допомогою DNS-записів.

Service Networking забезпечує ефективне розподілення трафіку між подами, підтримуючи доступність і стабільність додатків у Kubernetes, навіть при динамічній зміні складу подів у сервісі.

Network Policies у Kubernetes — це інструмент, що визначає правила мережевої взаємодії між подами і контролює доступ до них, підвищуючи рівень безпеки кластера. Вони дозволяють обмежувати доступ до подів на основі певних умов, таких як теги подів, ідентифікатори додатків або IP-адреси, тим самим забезпечуючи сегментацію трафіку всередині кластера.

Network Policies підтримуються багатьма CNI-плагінами (такими як Calico) і забезпечують контроль над трафіком всередині кластера, що є важливим для підвищення безпеки та відповідності вимогам до мережевого контролю.

Ці три складові — Pod Networking, Service Networking і Network Policies — разом забезпечують гнучкість, масштабованість і безпеку мережевої

інфраструктури в Kubernetes, роблячи її здатною підтримувати сучасні додатки в динамічних середовищах.

CNI (Container Network Interface) є стандартом, який забезпечує механізм інтеграції мережевих рішень із контейнерними оркестраторами, такими як Kubernetes. CNI є ключовим компонентом мережевої інфраструктури Kubernetes, що забезпечує стандартизований інтерфейс для налаштування мережевої взаємодії між контейнерами. Мережева модель Kubernetes створює уніфіковане середовище, де поди можуть взаємодіяти між собою подібно до фізичних хостів.Оскільки Kubernetes не має вбудованої мережевої реалізації, CNI дозволяє обрати та налаштувати мережевий плагін, який відповідатиме потребам конкретного кластера. CNI працює як інтерфейс між Kubernetes і різними мережевими плагінами, дозволяючи керувати IP-адресами подів, підтримувати політики безпеки та налаштовувати комунікацію між компонентами кластера. Ця архітектура надає гнучкість і дозволяє інтегрувати плагіни, розроблені для різних сценаріїв використання — від простих overlay-мереж до складних рішень з повною підтримкою мережевих політик.

Мережева модель Kubernetes створює уніфіковане середовище, де поди можуть взаємодіяти між собою подібно до фізичних хостів.
Основні вимоги мережевої моделі Kubernetes включають:

- Всі контейнери можуть комунікувати між собою без NAT
- Всі вузли можуть комунікувати з контейнерами без NAT
- IP-адреса, яку бачить контейнер, співпадає з тією, яку бачать інші компоненти системи

Calico є одним із найбільш функціональних і популярних CNI-плагінів у Kubernetes. Він забезпечує маршрутизацію трафіку на основі IP, дозволяючи налаштовувати мережу без використання overlay. Calico забезпечує низьку

затримку, що робить його підходящим для середовищ із високими вимогами до продуктивності. Плагін також підтримує BGP (Border Gateway Protocol), що дозволяє інтегрувати мережу кластера з зовнішніми мережами, забезпечуючи надійність і стабільність з'єднань. Calico має повну підтримку Network Policies, завдяки чому можна налаштувати точні правила для обмеження трафіку, що проходить між подами. Ця функціональність особливо корисна у великих корпоративних середовищах, де критично важливим є управління безпекою мережі.

Flannel є простим і легким у використанні CNI-плагіном, орієнтованим на overlay-мережі для Kubernetes. На відміну від Calico, Flannel використовує протоколи VXLAN або UDP для створення віртуальної мережі поверх фізичної інфраструктури. Це дозволяє йому забезпечувати просту і надійну комунікацію між подами, не вимагаючи складної конфігурації. Однак, Flannel не підтримує Network Policies, що обмежує його можливості в середовищах, де необхідна детальна сегментація трафіку і контроль безпеки. Flannel підходить для простих середовищ, де основна потреба — забезпечення базового зв’язку між подами, без значних вимог до складних правил маршрутизації або контролю доступу.

Weave створює overlay-мережу для Kubernetes і забезпечує маршрутизацію трафіку між подами за допомогою власного протоколу. Weave підтримує шифрування трафіку між подами, що забезпечує додатковий рівень безпеки. Він також має базову підтримку Network Policies, що дозволяє контролювати вхідний трафік на рівні подів, але не настільки розширену, як у Calico. Weave працює добре в середовищах, де потрібен баланс між базовою підтримкою безпеки та простотою налаштування. Однак його продуктивність може знижуватися в середовищах із великою кількістю трафіку або при масштабуванні, оскільки overlay-мережі часто мають більшу затримку.

Мережеві політики (Network Policies) в Kubernetes дозволяють задавати правила для вхідного та вихідного трафіку, що проходить між подами. Це важливий інструмент безпеки, який дозволяє обмежувати доступ до певних подів або сегментувати трафік всередині кластера. Реалізація Network Policies залежить від CNI-плагіна: не всі плагіни підтримують цю функцію, і рівень підтримки може варіюватися.

Calico має повну підтримку Network Policies і дозволяє задавати комплексні правила для контролю трафіку. Адміністратор може задавати політики, що обмежують доступ за допомогою тегів, IP-адрес або протоколів. Calico також дозволяє визначати політики для вхідного і вихідного трафіку, що забезпечує високий рівень безпеки та контроль над мережею. Це робить його ідеальним для середовищ, де важливим є точний контроль над мережевими з'єднаннями.

Flannel не підтримує Network Policies, що значно обмежує його можливості в середовищах, де потрібен контроль доступу. Це робить Flannel менш придатним для середовищ з підвищеними вимогами до безпеки, оскільки він забезпечує лише базову мережеву комунікацію між подами без можливості гнучкого налаштування трафіку.

Weave підтримує Network Policies на базовому рівні, дозволяючи контролювати вхідний трафік до подів. Це дозволяє налаштовувати базові правила для захисту додатків, проте можливості Weave щодо налаштування мережевих політик є обмеженими порівняно з Calico. Weave забезпечує базову сегментацію, але не підходить для складних середовищ, де потрібні гнучкі правила для всіх типів трафіку.

Завдяки Network Policies Kubernetes забезпечує рівень безпеки та контролю над мережею, що дозволяє адміністраторам сегментувати трафік і обмежувати доступ у кластері відповідно до вимог безпеки. Використання відповідного CNI-

плагіна для підтримки Network Policies дозволяє налаштувати Kubernetes для роботи в безпечному середовищі, яке відповідає вимогам корпоративних стандартів та знижує ризики несанкціонованого доступу.

Таблиця 1.1.

Основні характеристики та функціональні можливості

Характеристика	**Calico**	**Flannel**	**Weave Net**
Рівень реалізації	L3	L2 (VXLAN)/L3 (Host-gw)	L2
Тип мережі	Underlay/Overlay	Overlay/Underlay	Overlay
Network Policies	Розширені	Відсутні	Базові
Шифрування	IPSec/WireGuard	IPSec (експериментально)	IPSec
Multicast підтримка	Ні	Ні	Так
BGP підтримка	Так	Ні	Ні
Складність налаштування	Висока	Низька	Середня
Масштабованість	До 5000 вузлів	До 2000 вузлів	До 1000 вузлів

Описані особливості включають рівень мережевої моделі OSI, на якому працює кожен плагін, тип мережі, підтримку мережевих політик (Network Policies), можливість шифрування та підтримку BGP і multicast. Також враховується складність налаштування та масштабованість кожного плагіна. **[7]**

Рівень реалізації визначає на якому рівні мережевої моделі OSI працює плагін. Calico працює на L3 (мережевому) рівні, що забезпечує кращу продуктивність. Flannel підтримує обидва режими, а Weave працює на L2 (канальному) рівні.

Overlay мережі створюють додатковий рівень інкапсуляції, тоді як Underlay використовує фізичну мережеву інфраструктуру напряму.

Можливості з налаштування мережевих політик безпеки. Calico надає найбільш розширені можливості.

Таблиця 1.2

Продуктивність та рекомендації щодо використання

Сценарій використання	**Рекомендований CNI**	**Обґрунтування**
Великі промислові кластери	Calico	Найкраща масштабованість, продуктивність та можливості контролю трафіку
Прості розгортання	Flannel	Легке налаштування, низькі вимоги до ресурсів, достатня функціональність для базових сценаріїв
Середовища з multicast-комунікаціями	Weave Net	Вбудована підтримка multicast, зручність налаштування мульти-кластерних розгортань
Підвищені вимоги до безпеки	Calico	Розширені можливості Network Policy, інт

Для великих промислових кластерів, де потрібна максимальна масштабованість і надійність, Calico є оптимальним вибором. Це рішення підтримує високий рівень продуктивності та гнучкий контроль над трафіком, що важливо для великих організацій, де кластер може налічувати тисячі вузлів. Calico дозволяє ефективно управляти трафіком і мережевими політиками, забезпечуючи інтеграцію з BGP та можливість роботи в обох режимах — underlay та overlay, що робить його незамінним у масштабних корпоративних середовищах.

У випадках, коли основною метою є простота налаштування та економія ресурсів, наприклад, для тестових середовищ або базових сценаріїв розгортання, Flannel є ідеальним вибором. Цей CNI-плагін дозволяє швидко і просто налаштувати мережу без зайвих складнощів, що є перевагою для менших проектів або команд, яким не потрібні розширені можливості мережевого контролю. Flannel працює в overlay-режимі, використовуючи VXLAN для

інкапсуляції трафіку, що забезпечує просте з'єднання між подами, хоча й без підтримки детальних мережевих політик.

Для середовищ, де потрібні multicast-комунікації, наприклад, у розподілених системах або мульти-кластерних розгортаннях, Weave Net є кращим вибором. Weave Net підтримує multicast на рівні контейнерів, що дозволяє створювати стабільні з'єднання для обміну даними у великих розподілених системах. Крім того, Weave Net забезпечує зручне налаштування, яке спрощує мульти-кластерні розгортання, роблячи його привабливим для середовищ, що вимагають підтримки широкомовних комунікацій між різними компонентами.

У середовищах із підвищеними вимогами до безпеки, таких як фінансовий сектор або державні установи, Calico знову є оптимальним вибором через свої розширені можливості налаштування мережевих політик (Network Policies). Calico дозволяє детально контролювати доступ між подами завдяки підтримці microsegmentation, що дозволяє задавати політики на рівні окремих подів або їх груп. Це забезпечує ізоляцію різних компонентів додатка, мінімізуючи ризик несанкціонованого доступу. Calico також підтримує інтеграцію з системами виявлення та запобігання вторгненням (IDS/IPS), що робить його ефективним рішенням для середовищ з підвищеними вимогами до мережевої безпеки.

1.4. Показники надійності та доступності

Метрики управління інцидентами відіграють важливу роль у підтримці високої доступності та надійності IT-систем. Вони пропонують структурований підхід для виявлення, реагування та швидкого вирішення інцидентів. Відстеження метрик дозволяє вчасно виявляти потенційні проблеми, запобігати

перебоям у сервісі, підвищувати надійність системи та забезпечувати дотримання угод про рівень обслуговування (SLA).

Основні скорочення в управлінні інцидентами включають:

- **MTTR**: середній час до ремонту або відновлення (Mean Time to Repair or Recovery)
- **MTBF**: середній час між відмовами (Mean Time Between Failures)
- **MTTF**: середній час до відмови (Mean Time to Failure)
- **MTTA**: середній час до підтвердження (Mean Time to Acknowledge)

Розуміння цих показників є необхідним для команд IT та розробників, оскільки вони надають інформацію про продуктивність системи та допомагають розробляти стратегії для її покращення, що сприяє підвищенню задоволеності клієнтів та операційної ефективності.

1.4.1 MTTR: Середній час до ремонту або відновлення

MTTR має два тлумачення: середній час до ремонту (Mean Time to Repair) та середній час до відновлення (Mean Time to Recovery).

Середній час до ремонту визначає середню тривалість, необхідну для повернення системи або сервісу до нормального робочого стану після інциденту або збою. Це включає весь процес ремонту: від діагностики та усунення несправностей до перевірки ефективності вирішення проблеми.

Середній час до відновлення виходить за рамки простого ремонту та охоплює весь процес, необхідний для повного відновлення функціональності системи відповідно до експлуатаційних вимог, включаючи можливе відновлення даних, коригування конфігурацій або перемикання на резервні системи. [8]

Зниження MTTR в IT-управлінні критично важливе для підвищення доступності систем і задоволеності користувачів. Скорочений MTTR свідчить про ефективний процес відновлення, який мінімізує перебої та покращує надійність системи.

Основні переваги зниження MTTR:

- **Збільшена доступність** — скорочення MTTR призводить до зменшення тривалості простою, підвищуючи загальну тривалість безперервної роботи.
- **Зменшення збоїв** — швидке відновлення зменшує вплив збоїв на операційну діяльність, що сприяє збереженню продуктивності та зниженню фінансових втрат.
- **Покращений досвід користувачів** — користувачі відчувають менше незручностей та фрустрацій при швидкому вирішенні інцидентів, що підвищує їхню задоволеність.
- **Відповідність SLA** — багато організацій мають SLA, що визначають допустимий час вирішення інцидентів. Зниження MTTR допомагає командам дотримуватися цих угод, забезпечуючи надійність обслуговування.

Для зниження MTTR варто:

- Пріоритизувати інциденти за їх впливом та терміновістю.
- Впроваджувати автоматизацію для рутинних завдань, щоб зменшити втручання вручну та пришвидшити процес вирішення.
- Налагодити ефективні канали комунікації для координації зусиль відновлення.
- Впроваджувати інструменти моніторингу для швидкого виявлення інцидентів.

- Проводити аналіз першопричин для попередження повторних інцидентів, що сприяє довготривалому зниженню MTTR.

1.4.2 MTBF: Середній час між відмовами

MTBF — це метрика надійності для систем, які можна відновлювати після відмов. Вона показує середній час, протягом якого система може працювати без збоїв. Високий MTBF є критичним, оскільки він прямо впливає на доступність сервісу, досвід користувачів і загальну операційну ефективність.

MTBF надає уявлення про надійність системи, дозволяючи оцінювати і порівнювати її здатність стабільно функціонувати з часом. Це допомагає у прийнятті стратегічних рішень щодо інвестицій в інфраструктуру та конфігурацій системи.

MTBF можна використовувати для оцінки та прогнозування надійності системи. Високий MTBF вказує на довші періоди між відмовами, що свідчить про надійність і стабільність системи.

MTBF визначає стратегії обслуговування та оперативного планування. Системи з високим MTBF дозволяють планувати технічне обслуговування без суттєвих перерв у роботі. У той же час системи з низьким MTBF потребують частішого обслуговування, що вимагає більше ресурсів.

1.4.3 MTTF: Середній час до відмови

MTTF — це метрика, яка показує середній час до відмови для систем, які неможливо відремонтувати після збою. Вона часто використовується для оцінки тривалості життя незамінних компонентів, таких як апаратні компоненти або модулі програмного забезпечення, які після першої відмови потребують заміни.

MTTF застосовується для компонентів, які замінюються після першої відмови, тоді як MTBF використовується для компонентів, які ремонтуються і повертаються в експлуатацію.

MTTF (Mean Time To Failure) є ключовою метрикою, що визначає середній час роботи незамінного компонента до його повної відмови. Ця метрика є особливо важливою для компонентів, які не можна відремонтувати або повернути в експлуатацію після поломки, таких як деякі апаратні деталі (наприклад, жорсткі диски або блоки живлення) або певні програмні модулі. MTTF допомагає прогнозувати очікуваний термін служби цих компонентів і планувати їхню заміну заздалегідь, щоб уникнути раптових збоїв.

Визначення MTTF базується на аналізі історичних даних про попередні відмови аналогічних компонентів. Якщо компоненти мають високий MTTF, це свідчить про їх надійність і довговічність. Для IT-адміністраторів та інженерів ця метрика дозволяє:

- **Планувати заміни**: Знання MTTF допомагає передбачити, коли певні компоненти наближаються до кінця свого життєвого циклу. Це дає змогу замінити їх до того, як відбудеться відмова, що сприяє безперервній роботі системи.
- **Управляти запасами запасних частин**: Завдяки прогнозам MTTF можна заздалегідь закупити необхідні компоненти, зберігаючи їх у запасі.
- **Оптимізувати витрати**: Розуміння середнього часу до відмови дозволяє раціонально використовувати ресурси, уникаючи як передчасної заміни компонентів, так і ризиків, пов'язаних із затримкою їх заміни.

MTTF і MTBF (Mean Time Between Failures) обидві є метриками надійності, але вони застосовуються до різних типів систем. MTTF відноситься до незамінних компонентів, які потребують заміни після першого збою, тоді як

MTBF використовується для компонентів, які можуть бути відремонтовані та повернуті в експлуатацію після кожної відмови. Таким чином, MTTF дає уявлення про тривалість життя компонентів, які замінюються при відмові, тоді як MTBF дозволяє визначити частоту, з якою ремонтовані компоненти потребують обслуговування.

1.4.4 MTTA: Середній час до підтвердження

MTTA (Mean Time To Acknowledge) є показником швидкості реакції команди на інциденти. Він визначає середній час, який проходить від моменту виникнення інциденту до його визнання командою, що відповідає за вирішення. MTTA відображає швидкість, з якою команда помічає і визнає проблему, і є ключовим показником оперативності реагування на інциденти.

MTTA охоплює всі початкові етапи реагування, зокрема моніторинг, сповіщення та комунікацію з відповідальними особами. Чим коротший MTTA, тим швидше команда починає працювати над вирішенням проблеми, що є важливим для мінімізації часу простою та підвищення надійності системи. MTTA є критично важливим для організацій, де швидкість реагування на інциденти може безпосередньо впливати на бізнес-процеси та задоволеність користувачів.

Скорочення MTTA має ряд переваг для організації, оскільки дозволяє забезпечити швидке реагування на інциденти та мінімізувати негативний вплив на операції:

- **Швидше вирішення інцидентів**: Скорочений MTTA дозволяє оперативно ініціювати процес вирішення проблеми, що сприяє загальному зниженню часу простою.

- **Зменшення втрат продуктивності**: Коли проблема швидко виявляється і визнається, це знижує час, протягом якого система залишається в неробочому стані, що зберігає продуктивність організації.
- **Поліпшення досвіду користувачів**: Користувачі, які стикаються з інцидентами, менше відчувають незручності при своєчасному інформуванні та вчасній реакції на проблему.
- **Зміцнення довіри до IT-команди**: Швидкий відгук на інциденти демонструє готовність команди до оперативного реагування, що підвищує довіру користувачів та інших зацікавлених сторін.
- **Відповідність SLA**: Вчасне підтвердження інцидентів дозволяє організації дотримуватись встановлених SLA, що важливо для репутації та збереження довіри клієнтів.

Зниження MTTA вимагає впровадження кількох ключових стратегій, які допомагають підвищити ефективність початкового реагування:

- **Автоматизація сповіщень**: Впровадження систем моніторингу та автоматизованих сповіщень дозволяє миттєво інформувати команду про інцидент.
- **Чітко визначені ролі та відповідальність**: Наявність попередньо визначених відповідальних осіб, які можуть швидко підтвердити інцидент, знижує час на пошук виконавця.
- **Ефективні канали комунікації**: Налаштування надійних комунікаційних каналів і процедур обміну інформацією допомагає координувати дії команди, що пришвидшує процес реагування.
- **Регулярне навчання та інструктажі**: Постійне підвищення кваліфікації команди з управління інцидентами сприяє швидшому розпізнаванню і реагуванню на проблеми.

MTTA є однією з ключових складових ефективного управління інцидентами, яка дозволяє організаціям своєчасно розпочати роботу над вирішенням проблем, мінімізуючи їхній вплив на бізнес-процеси.

1.4.5 Порівняння та взаємозв'язок метрик інцидентів

Метрики **MTTR**, **MTBF**, **MTTF** і **MTTA** разом забезпечують комплексне уявлення про продуктивність та надійність системи. Ці метрики взаємопов’язані та при спільному аналізі дозволяють удосконалити процеси управління інцидентами, підвищити стабільність роботи та швидкість реагування на збої.

- **MTTR та MTTA мають прямий зв’язок**. Чим менше значення MTTA, тим швидше команда починає працювати над інцидентом, що призводить до скорочення MTTR.
- **MTBF та MTTF тісно пов’язані**. MTBF використовується для ремонтованих систем, тоді як MTTF — для компонентів, які замінюють після першого збою. Обидві метрики важливі для розуміння надійності.
- **MTTR має зворотній зв'язок з MTBF та MTTF**. Менший MTTR свідчить про швидший ремонт, що підвищує загальну надійність системи та може продовжити періоди між відмовами.
- **MTTA впливає на MTTR**. Швидке підтвердження інциденту сприяє пришвидшенню процесу реагування та зниженню загального MTTR.

Наведена таблиця допомагає зрозуміти, як кожна метрика впливає на надійність і продуктивність системи. Взаємозв'язок між метриками підкреслює важливість комплексного підходу до аналізу для досягнення більшої стабільності та ефективності управління інцидентами.

Таблиця 1.3

Порівняння та взаємозв'язок метрик інцидентів

Метрика	Визначення	Внесок у надійність та продуктивність	Взаємозв’язок з іншими метриками
MTTR	Середній час до ремонту: час, потрібний для відновлення після відмови	Оцінює швидкість відновлення. Менший MTTR підвищує надійність і зменшує час простою.	Зворотно пропорційний MTBF і MTTF. Менший MTTR підвищує загальну доступність.
MTBF	Середній час між відмовами: час між послідовними відмовами	Відображає тривалість роботи без збоїв. Вищий MTBF підвищує надійність системи.	Зворотно пропорційний MTTR і MTTF: більший MTBF знижує частоту ремонту.
MTTF	Середній час до відмови: середній строк служби до першої відмови	Оцінює очікуваний термін роботи незамінного компонента. Вищий MTTF підвищує надійність.	Непрямий зв'язок з MTBF, обидві метрики оцінюють надійність, але з різним фокусом.
MTTA	Середній час до підтвердження: час для визнання інциденту	Вимірює швидкість реагування на інциденти. Низький MTTA скорочує час до початку відновлення.	Тісно пов’язаний з MTTR: менший MTTA сприяє скороченню MTTR.

1.5. SLA, SLO та SLI для розгортання Kubernetes

В Kubernetes, як і в інших сучасних IT-системах, ключовими показниками ефективності та якості надання послуг є **SLA** (Угода про рівень обслуговування), **SLO** (Цільовий рівень обслуговування) та **SLI** (Показник рівня обслуговування). Ці три поняття взаємопов'язані і допомагають адміністраторам Kubernetes забезпечувати високий рівень доступності та надійності, а також досягати узгоджених з користувачами вимог. [9]

SLA — це формальна угода між постачальником послуг та споживачем, яка визначає допустимий рівень доступності, продуктивності та часу реагування на інциденти. В контексті Kubernetes SLA можуть включати такі показники, як доступність кластера (наприклад, 99.9% часу), максимальний час відновлення після збоїв або час реагування на критичні інциденти. SLA важлива для підтримки довіри користувачів, оскільки порушення SLA може призвести до штрафних санкцій або інших наслідків для постачальника послуг. [10]

Для Kubernetes це може означати, наприклад, що кластер зобов'язаний забезпечити 99.95% доступності сервісів протягом місяця, що означає допустимий час простою приблизно 21.56 хвилини на місяць.

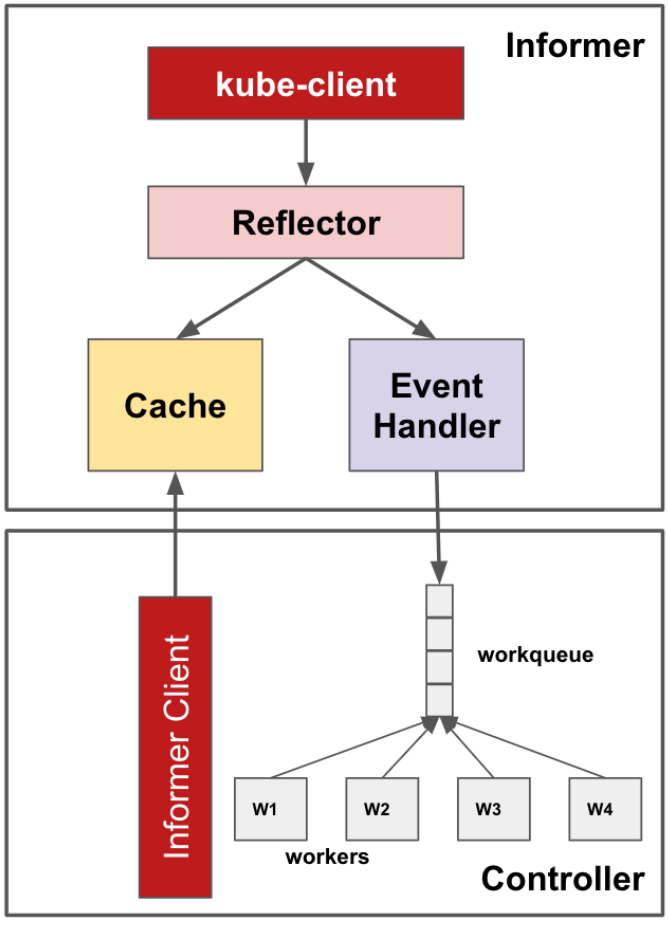

Рис. 1.3 Kubernetes Controller Framework

SLO — це специфічна метрика всередині SLA, яка встановлює конкретні цілі для рівня обслуговування. SLO задає, наприклад, що кластер Kubernetes повинен забезпечити 99.95% доступності подів і сервісів, або що час реагування на інциденти не перевищуватиме 5 хвилин для критичних інцидентів. Якщо SLA є юридичною угодою, то SLO — це внутрішній стандарт або мета, який команда підтримки має досягти.

Для команди Kubernetes SLO може включати цілі щодо часу відновлення кластера (наприклад, 5 хвилин після перезапуску вузла) або часу реакції на зміни у навантаженні, забезпечуючи автоматичне масштабування.

SLI — це метрика, яка вимірює фактичний рівень обслуговування, що надається системою, і використовується для оцінки відповідності SLO. Наприклад, якщо SLO вказує на необхідність 99.95% доступності, то SLI вимірює реальний відсоток доступності, наприклад, за допомогою метрик, таких як час відповіді, успішність запитів або час відновлення після збоїв.

Типові SLI для Kubernetes включають:

- **Доступність подів і сервісів**: відсоток часу, коли сервіси доступні для користувачів.
- **Час відповіді API**: середній час відповіді Kubernetes API на запити.
- **Час відновлення подів**: середній час, за який поди відновлюються після перезапуску або переміщення між вузлами.

Наявні SLIs/SLOs забезпечують базові показники, необхідні для гарантування працездатності кластера. Проте вони не повністю відповідають очікуванням користувачів у багатьох областях системи, тому наразі активно працюємо над розширенням їх покриття.

Передумови:

1. Кластер Kubernetes доступний і виконує обслуговування.

2. Величина "churn" (динаміка змін в кластері) складає не більше 20 змін за секунду, де:
    - churn = #(створення/оновлення/видалення Pod spec) + #(запити від користувачів) за секунду.

Таблиця 1.5

Kubernetes SLIs/SLOs

Статус	**SLI**	**SLO**
Офіційний	Затримка мутаційних API-запитів для (ресурс, дія), 99-й перцентиль за 5 хв	У стандартній інсталяції, для (ресурс, дія) (без віртуальних та агрегованих ресурсів): 99-й перцентиль на кластер-день <= 1 сек
Офіційний	Затримка лише читальних API-запитів для (ресурс, область), 99-й перцентиль за 5 хв	У стандартній інсталяції, для (ресурс, область) (без віртуальних та агрегованих ресурсів):
(a) <= 1 сек для області=ресурс;		
(b) <= 30 сек для області=namespace або область=кластер		
Офіційний	Затримка запуску подів без стану (від створення до запуску контейнерів), 99-й перцентиль	У стандартній інсталяції, 99-й перцентиль на кластер-день <= 5 сек
Розробка	Затримка запуску подів зі станом (від створення до запуску контейнерів), 99-й перцентиль	У стандартній інсталяції, 99-й перцентиль на кластер-день <= X (залежить від сховища)

Розробка	Затримка налаштування балансування навантаження, 99-й перцентиль	У стандартній інсталяції, 99-й перцентиль на кластер-день <= X
Розробка	Затримка налаштування DNS, 99-й перцентиль	У стандартній інсталяції, 99-й перцентиль на кластер-день <= X

Опис основних метрик:

Офіційні показники:

- Включають вимірювання затримки обробки API-запитів на зміну (mutation) та читання даних для різних ресурсів у кластері. Наприклад, вимірюється, наскільки швидко система обробляє мутаційні API-запити та запити на читання для конкретних ресурсів.
- Визначають час запуску подів без стану, що допомагає оцінити швидкість розгортання подів та готовність системи до нових навантажень.

Показники в розробці:

- Включають метрики, які покликані вимірювати більш специфічні аспекти продуктивності кластера, такі як затримка запуску подів зі станом, налаштування балансування навантаження та робота DNS.
- Також додаються метрики для моніторингу мережевої затримки всередині кластера, наприклад, затримка пінгу між подами та затримка обробки DNS-запитів.

Кожна метрика вимірюється як 99-й перцентиль, що означає, що для 99% запитів або дій час обробки не повинен перевищувати задане значення (наприклад, 1 секунда для мутаційних API-запитів). Це дозволяє встановити строгі показники продуктивності, які допомагають забезпечити стабільність і надійність кластерів.

Всі ці SLI/SLO налаштовані для роботи в межах стандартної інсталяції Kubernetes. Це означає, що показники можуть служити орієнтиром для оцінки базової ефективності інсталяції, а також допомагають у визначенні областей для оптимізації та масштабування.

У Kubernetes надійність кластера визначається багатьма факторами, які впливають на стабільність і безперебійну роботу системи. Серед основних чинників можна виділити **розділення мережі** (Network partitioning), **розподіл подів** (Pod scheduling) і **обмеження ресурсів** (Resource constraints). Кожен з них має значний вплив на доступність і ефективність роботи Kubernetes-кластера.

Network partitioning — це ситуація, коли частини мережі стають недоступними одна для одної, що може призвести до порушення комунікації між компонентами кластера. У випадку Kubernetes це може вплинути на здатність подів і вузлів взаємодіяти належним чином. В результаті можуть виникати такі проблеми, як:

- **Неповний обмін даними** між подами, що впливає на взаємодію мікросервісів і доступність додатків.
- **Збої у функціонуванні тощо** — якщо компоненти не можуть обмінюватися інформацією, це може порушити балансування навантаження і відновлення сервісів.

Для зменшення впливу розділення мережі адміністратори можуть впроваджувати мережеві політики та механізми моніторингу, що дозволяють швидко виявляти і усувати проблеми з комунікацією. Також важливо забезпечити резервування мережі, щоб у разі розділення мережі зберігати часткову доступність сервісів.

Pod scheduling — це процес розподілу подів по вузлах у кластері. Надійність розподілу подів є критичною для забезпечення стабільності роботи додатків у Kubernetes. Неправильний або нерівномірний розподіл подів може призвести до:

- **Навантаження на окремі вузли**: якщо багато подів запущено на одному вузлі, це може призвести до його перевантаження, збоїв та зниження загальної доступності.
- **Залежності між подами**: поди, які мають працювати разом, можуть опинитися на різних вузлах, що може підвищити затримки або знизити ефективність роботи додатка.

Kubernetes використовує планувальник (scheduler) для забезпечення оптимального розподілу подів на основі доступних ресурсів, специфічних вимог додатків і умов вузлів. Для підвищення надійності важливо налаштувати політики розподілу, такі як **анти-афініті** (anty-affinity), яка розподіляє поди з однаковими характеристиками на різні вузли для запобігання перевантаженням.

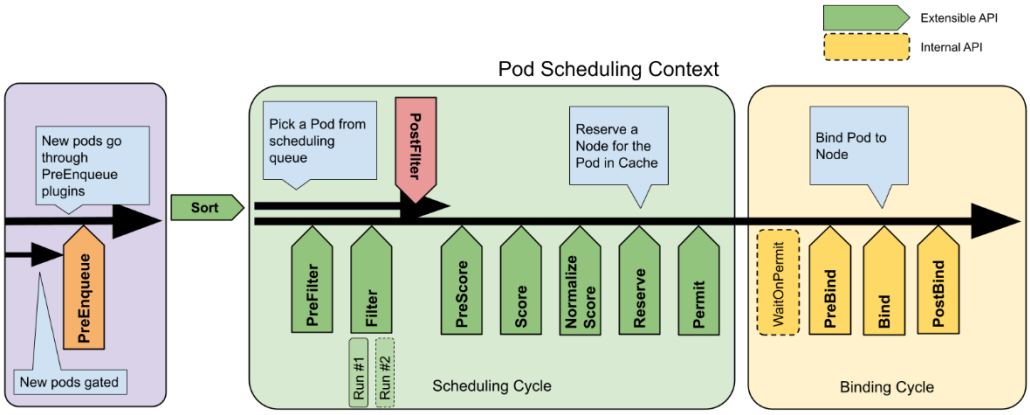

Рис 1.4 Контекст планування подів

Pod scheduling — це процес визначення, на якому вузлі кластера Kubernetes буде розміщений кожен под. Цей процес є критично важливим для забезпечення надійності, продуктивності та ефективного використання ресурсів в кластері. Планувальник Kubernetes автоматично розподіляє поди по вузлах на основі доступних ресурсів, політик, пріоритетів і вимог додатків. Неправильний розподіл подів може призвести до перевантаження вузлів, неефективного використання ресурсів або навіть простою додатків. Тому налаштування та оптимізація процесу розподілу подів мають вирішальне значення для стабільної роботи Kubernetes-кластера.

Одним із головних факторів, які впливають на розподіл подів, є доступні ресурси вузлів, такі як процесор (CPU), оперативна пам'ять (RAM) та дисковий простір. Планувальник розподіляє поди так, щоб задовольнити вимоги до ресурсів кожного пода, уникнувши перевантаження вузлів і зберігши стабільність системи. Якщо под вимагає більше ресурсів, ніж доступно на вузлі, він не буде розміщений на цьому вузлі, що захищає від перевантаження інших подів.

Також важливим фактором є топологічна розподіленість, яка враховує взаємодію подів один з одним. Kubernetes дозволяє налаштовувати політики афініті та антиафініті для подів, що визначають їхнє розміщення відносно інших подів. Наприклад, поди, які тісно взаємодіють між собою, можуть бути розміщені на одному вузлі для зниження затримок. З іншого боку, антиафініті дозволяє розподілити поди по різних вузлах, що знижує ризик одночасної відмови кількох подів через збій одного вузла і підвищує відмовостійкість додатків.

Пріоритет подів також грає важливу роль у розподілі. Kubernetes дозволяє задавати пріоритети для подів, що визначає черговість їх розміщення на вузлах. Поди з високим пріоритетом розміщуються першими, що дозволяє забезпечити

їхню доступність навіть при дефіциті ресурсів. Якщо ресурси обмежені, поди з нижчим пріоритетом можуть бути видалені, щоб звільнити місце для подів із вищим пріоритетом. Це особливо корисно для критичних сервісів, де деякі поди є важливішими для бізнес-процесів, а інші виконують менш важливі завдання.

Крім того, розподіл подів може враховувати обмеження по зоні розташування, особливо в хмарних середовищах. Kubernetes може розподіляти поди по різних зонах доступності (наприклад, у AWS або Google Cloud), що забезпечує відмовостійкість у разі збоїв у одній із зон. Такий підхід дозволяє зберегти додаток доступним завдяки розміщенню подів у різних дата-центрах або зонах, навіть якщо в одній зоні виникнуть проблеми.

Node affinity є ще одним інструментом, який дозволяє задавати специфічні правила розміщення подів на вузлах з певними характеристиками. Наприклад, поди, що потребують GPU для машинного навчання, можуть бути розміщені тільки на вузлах, які мають цей ресурс. Це забезпечує оптимальне використання специфічного обладнання та зменшує ризик запуску подів на невідповідних вузлах.

Налаштування та оптимізація розподілу подів також включає управління ресурсами через політики requests та limits, що визначають мінімальний і максимальний обсяг ресурсів, який може використовувати под. Це допомагає забезпечити стабільну роботу всіх додатків у кластері та уникнути перевантаження ресурсів. Крім того, за допомогою taints і tolerations можна налаштувати ізоляцію подів, створюючи спеціалізовані зони в кластері, де певні поди не можуть розміщуватися або навпаки мають право запуску, незважаючи на обмеження.

Kubernetes також підтримує preemption — функцію, яка дозволяє переміщати поди з низьким пріоритетом, щоб звільнити ресурси для подів із

високим пріоритетом. Це особливо корисно у випадках дефіциту ресурсів і допомагає зберегти доступність критично важливих сервісів навіть при високих навантаженнях. Ще одним важливим аспектом є балансування навантаження, коли Kubernetes переміщує поди для уникнення перевантаження окремих вузлів, забезпечуючи загальну стабільність і продуктивність кластера.

Функція горизонтального автоматичного масштабування (Horizontal Pod Autoscaling) дозволяє автоматично збільшувати або зменшувати кількість подів у залежності від навантаження. Наприклад, якщо навантаження на додаток зростає, Kubernetes може додати більше подів, щоб задовольнити запити користувачів. Це дозволяє зберігати високу продуктивність додатків навіть при пікових навантаженнях.

Розподіл подів у Kubernetes може бути складним завданням, особливо в динамічних середовищах, де навантаження та доступність ресурсів швидко змінюються. У середовищах із високими вимогами до відмовостійкості важливо забезпечити оптимальну розподіленість подів по вузлах та зонах доступності. Загалом, pod scheduling є важливою складовою ефективного управління Kubernetes-кластером, оскільки правильна конфігурація планувальника дозволяє досягти стабільної роботи додатків та ефективного використання ресурсів.

Висновки

Kubernetes є потужною платформою для оркестрації контейнеризованих додатків, яка значно спрощує управління складними IT-системами завдяки автоматизації, гнучкості та масштабованості. Її функціонал забезпечує стабільність роботи додатків, ефективне використання ресурсів і високу доступність навіть у динамічних середовищах.

Основні переваги Kubernetes:

1. **Автоматизація управління контейнерами:** Kubernetes автоматично розгортає, масштабує та відновлює поди, що значно зменшує необхідність ручного втручання і підвищує продуктивність команд.
2. **Балансування навантаження:** Вбудовані механізми балансування трафіку забезпечують стабільну роботу додатків, розподіляючи запити між подами для уникнення перевантаження.
3. **Мережева інтеграція:** Підтримка CNI-плагінів, таких як Calico, Flannel, і Weave, дозволяє адаптувати мережеву топологію до конкретних вимог проєкту, забезпечуючи безпеку та продуктивність.
4. **Гнучке масштабування:** Kubernetes дозволяє динамічно збільшувати кількість подів або вузлів для задоволення змінюваних потреб додатків.
5. **Висока доступність:** Завдяки можливостям автоматичного відновлення подів і вузлів, система може швидко реагувати на збої та забезпечувати безперервність роботи.

Правильна побудова мережевої інфраструктури є критично важливою для стабільності та продуктивності кластерів. Вибір CNI-плагіна залежить від масштабів проєкту та потреб у безпеці:

- **Calico:** Оптимальний для великих і безпечних кластерів завдяки розширеній підтримці Network Policies.
- **Flannel:** Проста у використанні опція для базових сценаріїв.
- **Weave:** Підходить для середовищ, які потребують multicast-з'єднань або мультикластерної інтеграції.

Управління надійністю кластерів базується на ключових метриках:

1. **MTTR (Mean Time to Repair):** Час, необхідний для відновлення системи після збою, прямо впливає на доступність додатків.

2. **MTBF (Mean Time Between Failures):** Середній час безперервної роботи відображає надійність системи.
3. **MTTF (Mean Time to Failure):** Використовується для оцінки тривалості роботи незамінних компонентів.
4. **MTTA (Mean Time to Acknowledge):** Швидкість реагування на інциденти є ключовим фактором підтримки SLA.

Kubernetes є невід'ємною частиною сучасних IT-інфраструктур, яка забезпечує стабільність і масштабованість додатків. Завдяки гнучкій архітектурі та автоматизації, ця платформа підтримує високий рівень доступності та продуктивності. Однак успішне використання Kubernetes залежить від правильного планування мережевої топології, вибору інструментів управління та моніторингу, а також впровадження найкращих практик забезпечення надійності.

РОЗДІЛ 2
МАТЕМАТИЧНА МОДЕЛЬ МЕРЕЖЕВОЇ ТОПОЛОГІЇ КЛАСТЕРА

2.1. Формалізація мережевої структури

2.1.1. Теоретико-графова модель кластера

Формалізація мережевої структури Kubernetes-кластера є необхідним етапом для аналізу його топології, оцінки надійності, продуктивності та ефективності управління ресурсами. Використання теоретико-графового підходу дозволяє створити формальну модель мережі, яка відображає взаємодію між її компонентами. У цьому підході Kubernetes-кластер представляється у вигляді орієнтованого графа $G = (V, E)$ де V — множина вузлів, а E — множина ребер. **[11]**

Вузли графа (V) відповідають компонентам мережевої структури Kubernetes. Серед них:

1. **Майстер-вузли (Master Nodes)**. Це центральні вузли, які виконують функції управління кластером. Вони містять компоненти контрольної площини (Control Plane), такі як:
    - API-сервер, що є точкою входу для всіх запитів до кластера;
    - etcd, який виступає як розподілене сховище ключ-значення, що зберігає конфігурацію та стан кластера;
    - планувальник (Scheduler), який визначає, на якому з робочих вузлів буде запущено pod;
    - менеджер контролерів, що забезпечує відповідність поточного стану системи заданим конфігураціям.
2. **Робочі вузли (Worker Nodes)**. Ці вузли забезпечують запуск контейнеризованих додатків у pod'ах. Кожен вузол працює з kubelet — агентом, який взаємодіє з контрольними компонентами майстер-вузла для запуску pod'ів та моніторингу їхнього стану.
3. **Поди (Pods)**. Поди є базовими одиницями розгортання в Kubernetes. Кожен pod може містити один або декілька взаємопов’язаних контейнерів. Поди взаємодіють між собою за допомогою унікальних IP-адрес у межах кластера.
4. **Сервіси (Services)**. Сервіси створюють стабільну точку доступу до групи подів, автоматично оновлюючи маршрутизацію при їхньому створенні чи видаленні. Це забезпечує високий рівень абстракції для споживачів додатків.
5. **Ingress-контролери**. Вони відповідають за маршрутизацію зовнішніх запитів до внутрішніх сервісів і можуть включати механізми SSL-термінації.

Теоретико-графова модель дозволяє чітко формалізувати мережеву структуру Kubernetes-кластера, що полегшує аналіз і оптимізацію її роботи.

На рисунку 2.1 представлено орієнтований граф, де вузли відповідають компонентам системи (майстер-вузол, робочі вузли, поди, сервіси), а ребра відображають зв'язки між ними.

Видно, що майстер-вузол взаємодіє з робочими вузлами через API-сервер, поди отримують інструкції для виконання завдань, а сервіси забезпечують маршрутизацію трафіку до подів.

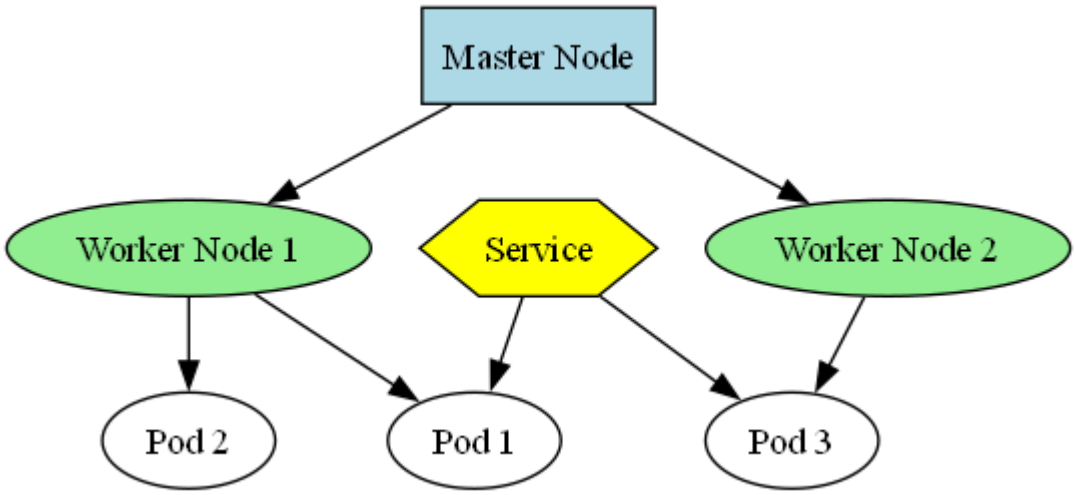

Рис 2.1 Теоретико-графова модель Kubernetes-кластера

Ребра графа (E) описують зв’язки між компонентами мережі. Ці зв’язки відображають обмін даними, комунікацію та управління між вузлами, подами та сервісами. Наприклад:

- API-сервер взаємодіє з kubelet на робочих вузлах, передаючи інструкції щодо розгортання або зміни подів;
- kubelet повідомляє майстер-вузол про поточний стан контейнерів;

- поди комунікують через мережеву модель Kubernetes, яка забезпечує унікальні IP-адреси для кожного pod'a, що дозволяє прямий обмін даними без NAT;
- сервіси з’єднують зовнішній трафік із подами через механізми балансування навантаження.

Графова модель дозволяє чітко уявити структуру та взаємодію компонентів кластера. Вона є основою для аналізу стійкості, продуктивності та масштабованості мережі. Наприклад, вузли можуть бути класифіковані за їхньою роллю (управління, обчислення, зберігання), а ребра — за типами взаємодії (керування, передача даних, балансування). Модель також враховує особливості контейнерної мережі Kubernetes: унікальність IP-адрес кожного pod'a, що усуває необхідність додаткових рівнів трансляції; можливість використання різних плагінів Container Network Interface (CNI) для налаштування мережевої взаємодії.

Формалізація мережевої структури у вигляді графа створює основу для оцінки ключових характеристик мережі таких як стійкості до збоїв, яка визначається можливістю збереження функціональності при відмові окремих вузлів або ребер, продуктивності, що залежить від ефективності маршрутизації та швидкості обробки запитів, масштабованості, яка визначає здатність кластера ефективно обробляти збільшення навантаження.

Додавання ваг та метрик у теоретико-графову модель Kubernetes-кластера дозволяє не лише формалізувати мережеву структуру, але й глибше проаналізувати її ефективність, продуктивність та надійність. Ваги присвоюються вузлам та ребрам графа для відображення їхніх характеристик чи впливу на загальну роботу системи, а метрики використовуються для оцінки ключових параметрів мережі. **[12]**

Вузли графа (V) у моделі Kubernetes мають різні ваги, що відображають їхню роль у кластері. Майстер-вузли, як ключові елементи управління, отримують високі ваги через їхню важливість у забезпеченні стабільної роботи всієї системи. Наприклад, вагу вузла, який відповідає за API-сервер, можна збільшити, оскільки саме через нього проходять усі запити до кластера. Робочі вузли отримують ваги залежно від кількості подів, що працюють на них, або рівня використання їхніх обчислювальних ресурсів. Поди, у свою чергу, оцінюються за критичністю для додатків: ті, які обслуговують сервіси з високим навантаженням, отримують вищу вагу. Сервіси також можуть бути оцінені за кількістю подів, які вони обслуговують, або обсягом трафіку, що проходить через них.

Ребра графа (E) представляють з'єднання між компонентами кластера, і їхні ваги можуть базуватися на кількох важливих характеристиках. Наприклад, пропускна здатність зв'язку між вузлами визначає максимальну швидкість передачі даних, а затримка відображає час, необхідний для передачі пакета між компонентами. Надійність зв'язку, яка враховує ймовірність успішної передачі даних, також є важливим параметром. Наприклад, з'єднання між API-сервером і kubelet на робочих вузлах може мати високу вагу через критичність цього каналу. Крім того, вага ребра може залежати від обсягу трафіку, що через нього проходить: наприклад, ребра, через які проходить найбільша кількість запитів, можуть мати вищу вагу.

Для візуалізації ваг вузлів та ребер у теоретико-графовій моделі кластера Kubernetes на рисунку 2.2 представлено граф, який демонструє ваги вузлів (залежно від їх критичності) та ребер (залежно від пропускної здатності чи затримки зв'язків

Метрики, які базуються на вагових характеристиках вузлів та ребер, дозволяють оцінити продуктивність та стійкість мережі. Однією з таких метрик є коефіцієнт центральності вузлів, який визначає важливість конкретного вузла у топології. Наприклад, API-сервер матиме високу центральність, оскільки більшість з'єднань проходить через нього. Іншою важливою метрикою є навантаження на вузли, яке розраховується як сума ваг усіх ребер, що входять у вузол. Це дозволяє виявити вузли з високим рівнем навантаження, які можуть стати вузьким місцем у кластері.

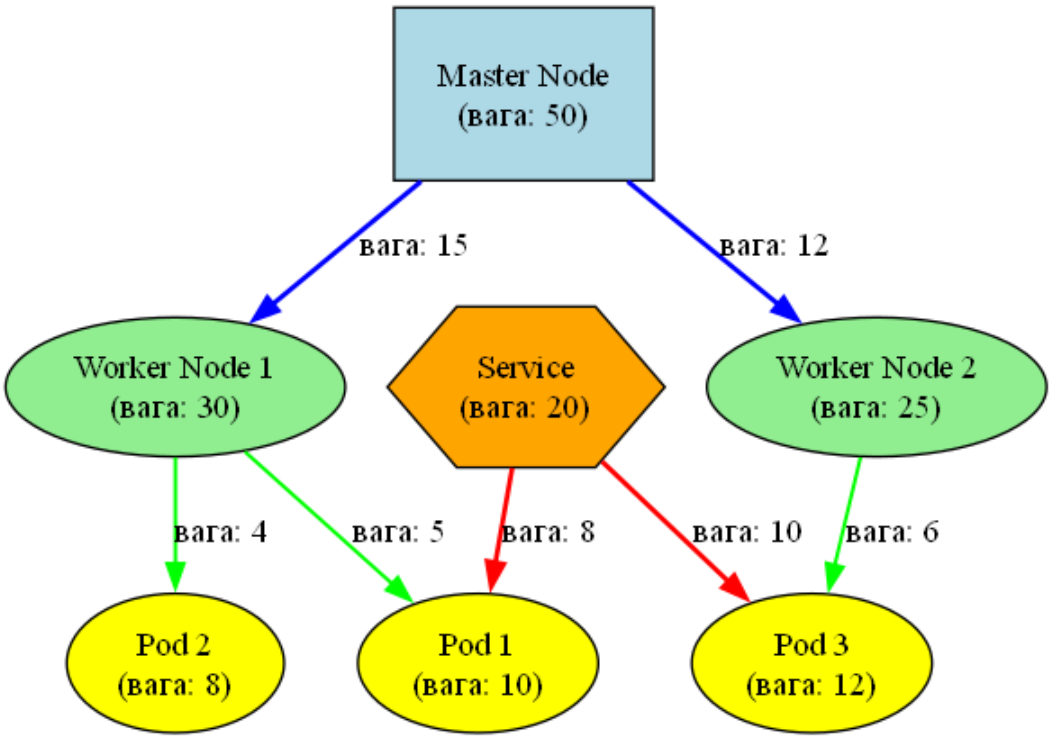

Рис 2.2 Граф із вагами вузлів і ребер у теоретико-графовій моделі Kubernetes-кластера

Середня затримка мережі є ще однією корисною метрикою, яка обчислюється як середня вага всіх ребер. Вона дозволяє оцінити загальну продуктивність системи. Для аналізу стійкості можна використовувати індекс

стійкості, який враховує кількість критичних вузлів у кластері. Наприклад, якщо відмова одного вузла спричиняє значні збої у функціонуванні системи, цей вузол є критичним, і його роль слід переоцінити.

Використання ваг і метрик у теоретико-графовій моделі дозволяє враховувати як структурні, так і динамічні аспекти роботи Kubernetes-кластера. Це надає можливість виконувати комплексний аналіз та оптимізацію мережевої топології для досягнення високої продуктивності та надійності. **[13]**

2.1.2 Матричне представлення топології

Матричне представлення топології використовується для формального опису та аналізу зв'язків між компонентами системи. У випадку Kubernetes-кластера це дозволяє чітко структурувати взаємодії між вузлами, такими як Master Node, Worker Nodes, Pods, Services тощо. Основна перевага матричного підходу полягає у його компактності, що полегшує обчислення, аналіз та автоматизацію процесів дослідження топології.

Одним із ключових інструментів матричного підходу є **матриця суміжності**. Вона є квадратною матрицею розміром $n \times n$, де n — кількість вузлів у системі. Елементи цієї матриці вказують на наявність або відсутність з'єднання між вузлами. Наприклад, якщо $a_{ij} = 1$, це означає, що існує зв'язок між вузлом i та вузлом j. У разі відсутності з'єднання значення елемента дорівнює нулю. Таким чином, матриця суміжності дозволяє легко зрозуміти, які компоненти кластера взаємодіють один із одним.

Для формалізації топології Kubernetes-кластера з такими вузлами, як Master Node, Worker Node 1, Worker Node 2, Pod 1, Pod 2, Pod 3 та Service, матриця суміжності може мати такий вигляд

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \quad (2.1)$$

Кожен рядок і стовпець матриці відповідає конкретному вузлу, а значення в клітинках — зв’язкам між ними. Наприклад, у першому рядку видно, що Master Node має зв’язки з Worker Node 1 і Worker Node 2, але не взаємодіє безпосередньо з іншими вузлами.

Ще одним важливим видом матриці є **вагова матриця**, яка враховує характеристики зв’язків між вузлами, такі як пропускна здатність, затримка або обсяг трафіку. Елементи вагової матриці містять числові значення, які відображають ці характеристики. Якщо зв’язок відсутній, відповідний елемент матриці дорівнює нулю. Для Kubernetes-кластера вагова матриця може виглядати так:

$$W = \begin{bmatrix} 0 & 15 & 12 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 & 0 & 10 & 0 \end{bmatrix} \quad (2.2)$$

У цій матриці, наприклад, значення 15 у комірці (1,2) вказує на вагу з’єднання між Master Node і Worker Node 1, що може відображати пропускну здатність або рівень трафіку між цими вузлами.

Матричне представлення дозволяє виконувати ефективні обчислення, спрямовані на аналіз топології мережі. Одним із ключових аспектів є можливість обчислення ступеня вузлів, тобто кількості з'єднань, які виходять з вузла або входять до нього. Це дозволяє визначити вузли з найбільшим навантаженням або найважливішу роль у топології. Наприклад, вузол із найвищим ступенем може бути критичним для роботи кластера. Крім того, множення матриці суміжності на саму себе дозволяє визначити кількість шляхів певної довжини між вузлами.

Важливо також використовувати вагову матрицю для оцінки загальної продуктивності та стійкості системи. Наприклад, середнє значення ваг всіх зв'язків у матриці дає уявлення про середню пропускну здатність мережі, а найнижчі значення можуть вказувати на вузькі місця.

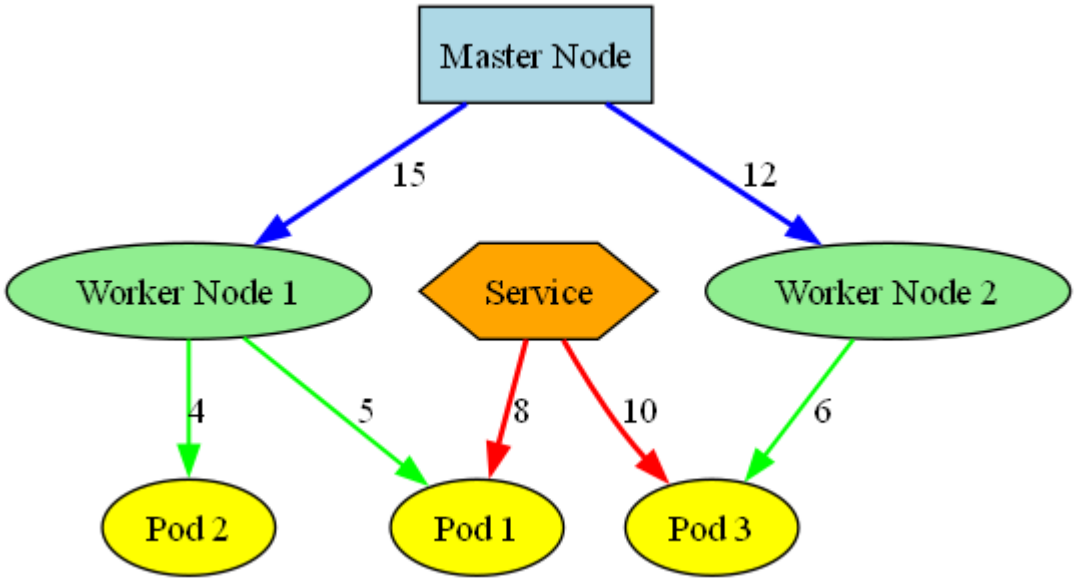

Рис 2.3 Топологія Kubernetes-кластера

2.1.3 Параметри мережевої структури

Мережеві структури Kubernetes-кластерів визначають ключові параметри, які безпосередньо впливають на якість і стабільність роботи системи. До таких

параметрів належать зв’язність, латентність та пропускна здатність. Кожен із цих аспектів характеризує різні властивості мережі, такі як її топологічна стійкість, ефективність передачі даних між вузлами та здатність витримувати високі навантаження. Розуміння та оптимізація цих параметрів є важливим завданням при розгортанні Kubernetes-кластерів, особливо в умовах великої кількості компонентів, що взаємодіють між собою.

Зв’язність описує можливість вузлів у мережі взаємодіяти один із одним. Вона є фундаментальним параметром, оскільки забезпечує безперебійну передачу даних між компонентами системи. У контексті Kubernetes-кластерів зв’язність відображає, наскільки вузли, такі як Master Node, Worker Nodes, Pods та сервіси, інтегровані в загальну мережеву структуру. Висока зв’язність є критично важливою, оскільки гарантує, що дані можуть досягати свого призначення навіть у разі відмови одного або кількох з’єднань. Наприклад, у топології Kubernetes Master Node зазвичай має найвищу зв’язність, оскільки він є центральною точкою координації для всіх Worker Nodes і повинен забезпечувати швидку комунікацію з ними. У графовій моделі зв’язність оцінюється за такими характеристиками, як ступінь вузлів, кількість компонентів зв’язності та діаметр графа. Діаметр графа визначає максимальну відстань між будь-якими двома вузлами в мережі, що вказує на ефективність передачі даних у кластері. Наприклад, у повністю зв’язній топології діаметр буде мінімальним, що гарантує швидкість взаємодії між компонентами. Якщо зв’язність порушується через відмову вузла або фізичного з’єднання, це може призвести до часткової втрати функціональності кластера або навіть до його повної недієздатності. Тому забезпечення резервних зв’язків і балансування навантаження між компонентами є необхідними для збереження стабільності системи.

Латентність, або час затримки передачі даних, є ще одним важливим параметром, який значно впливає на продуктивність Kubernetes-кластерів. Латентність визначає, скільки часу займає передача даних між двома вузлами мережі, і є особливо критичною для додатків, які працюють у реальному часі. У кластері Kubernetes латентність впливає на три основні рівні взаємодії: між Master Node і Worker Nodes, між Pods і між зовнішніми клієнтами та Ingress. Наприклад, висока латентність між Master Node і Worker Nodes може призвести до уповільнення процесів розподілу подів, оновлення конфігурацій або реагування на події. Латентність між Pods визначає ефективність внутрішньокластерних сервісів, особливо у мікросервісній архітектурі, де сервіси часто взаємодіють один із одним. Латентність між Ingress і зовнішніми клієнтами впливає на час відповіді додатків для кінцевих користувачів. Основними складовими латентності є фізичний час передачі даних через з'єднання, час обробки запитів у вузлах і затримки через перевантаження мережі або вузлів. Для зниження латентності важливо використовувати високошвидкісні фізичні з'єднання, такі як Ethernet, оптимізувати маршрутизацію та впроваджувати сучасні алгоритми балансування навантаження. **[14]**

Пропускна здатність є параметром, який визначає обсяг даних, що може бути переданий через з'єднання за одиницю часу. Цей параметр визначає, наскільки ефективно мережа може підтримувати великі обсяги трафіку або обробляти запити від численних клієнтів. У контексті Kubernetes пропускна здатність є критично важливою для взаємодії між Master Node та Worker Nodes, оскільки вона впливає на швидкість передачі великих обсягів даних, таких як журнали, метрики або оновлення конфігурацій. Між Pods пропускна здатність визначає, наскільки ефективно сервіси можуть передавати дані один одному, особливо в умовах високого навантаження. Для взаємодії між зовнішніми

клієнтами та Ingress висока пропускна здатність гарантує стабільну роботу додатків, які обробляють великий потік даних, таких як мультимедіа чи потокові сервіси. Пропускна здатність залежить від ширини каналу зв’язку, завантаження мережі та ефективності мережевих плагінів, таких як Calico або Flannel, які оптимізують маршрутизацію. Наприклад, використання 10-гігабітного Ethernet або впровадження політик пріоритетності трафіку може значно підвищити пропускну здатність і, відповідно, продуктивність системи.

Таким чином, параметри зв’язності, латентності та пропускної здатності є ключовими для забезпечення стабільної та ефективної роботи Kubernetes-кластера. Їхній аналіз дозволяє проектувати мережі, які відповідають вимогам до продуктивності та масштабованості, з урахуванням особливостей реальних сценаріїв використання.

2.2. Модель оцінки надійності

Модель оцінки надійності є важливим елементом аналізу системних характеристик Kubernetes-кластера, оскільки вона дозволяє передбачати ймовірність відмов, оцінювати стабільність роботи системи та планувати заходи для забезпечення безперебійного функціонування. Для цього використовуються ймовірнісні підходи, марковські моделі та розрахунок ключових показників надійності, таких як середній час між відмовами (MTBF), прогноз доступності (Availability prediction) та ймовірність відмов (Failure probability calculation). **[15]**

Ймовірнісна модель відмов описує, як відмови виникають у системі, з урахуванням статистичних характеристик компонентів і їхньої взаємодії. В основі моделі лежать такі припущення:

1. **Випадковість відмов.** Відмови в системі є випадковими подіями, які можна описати за допомогою ймовірнісних розподілів (наприклад, експоненційного чи нормального).
2. **Час до відмови (TTF).** Це час, протягом якого компонент системи працює без збоїв. Для систем із незалежними компонентами TTF може описуватися експоненційним розподілом із параметром інтенсивності відмов (λ).
3. **Резервування компонентів.** Модель враховує можливість резервування, що знижує ймовірність повної відмови системи.

Ймовірність безвідмовної роботи ($R(t)$) може бути розрахована за формулою:

$$R(t) = e^{-\lambda t},$$

де λ — інтенсивність відмов, t — час.

Марковські ланцюги дозволяють моделювати перехід системи між різними станами, такими як робочий стан, стан з частковою відмовою або повний збій. Основна ідея полягає в тому, що ймовірність переходу залежить тільки від поточного стану системи, а не від попередніх станів (властивість Марковості).**[16]**

Модель Марковських ланцюгів описується такими елементами:

1. **Стан системи.** Кожен стан представляє конфігурацію системи, наприклад, "усі компоненти працюють" або "відмова одного вузла".
2. **Матриця ймовірностей переходів.** Це квадратна матриця P, де кожен елемент p_{ij} визначає ймовірність переходу зі стану i до стану j за одиницю часу.
3. **Рівноважний стан.** Через певний час система досягає рівноважного стану, в якому ймовірності перебування в кожному стані стабілізуються.

Наприклад, для простого кластера з двома станами ("робочий стан" і "збій"), матриця ймовірностей переходів може бути такою:

$$P = \begin{bmatrix} 1-\lambda & \lambda \\ \mu & 1-\mu \end{bmatrix} \quad (2.3)$$

де λ — ймовірність відмови, μ — ймовірність відновлення.

Середній час між відмовами ($MTBF$) — це середній період часу, протягом якого система працює без збоїв. Він розраховується як обернена величина до інтенсивності відмов (λ):

$$MBTF = \frac{1}{\lambda} \quad (2.4)$$

Цей показник використовується для оцінки середньої тривалості безперебійної роботи системи, зокрема її окремих компонентів.

Прогноз доступності (AAA) визначає ймовірність того, що система працює в заданий момент часу. Він розраховується за формулою:

$$\mathrm{A} = \frac{MTTF}{MTTF + MTTR} \quad (2.5)$$

де $MTTF$ — середній час до відмови, $MTTR$— середній час відновлення після відмови.

Даний показник дозволяє оцінити, наскільки доступною буде система для виконання завдань у реальних умовах експлуатації.

Ймовірність відмови системи ($Q(t)$) визначає частку часу, коли система перебуває у стані збою. Вона обчислюється як доповнення до ймовірності безвідмовної роботи:

$$Q(t) = 1 - R(t), \tag{2.6}$$

де $R(t) -$ ймовірність безвідмовної роботи.

У багатокомпонентних системах із резервуванням ймовірність відмови зменшується за рахунок паралельної роботи резервних компонентів.

Модель оцінки надійності, яка базується на ймовірнісному аналізі, Марковських ланцюгах та ключових показниках, дозволяє комплексно оцінити стабільність Kubernetes-кластера. Це забезпечує основи для проектування надійних систем і розробки стратегій їх обслуговування.

2.3. Аналіз факторів впливу

Мережеві фактори відіграють ключову роль у функціонуванні Kubernetes-кластера, оскільки від них залежить стабільність, продуктивність та ефективність взаємодії між компонентами системи. Ці фактори визначають, наскільки якісно вузли, поди та сервіси можуть обмінюватися даними в рамках кластерної інфраструктури. Глибокий аналіз мережевих факторів дає змогу ідентифікувати вузькі місця, які можуть впливати на продуктивність, і дозволяє приймати обґрунтовані рішення щодо оптимізації мережевої інфраструктури. **[17]**

Пропускна здатність мережі визначає максимальний обсяг даних, який може бути переданий між компонентами системи за одиницю часу. У контексті Kubernetes-кластера це стосується передачі даних між Master Node, Worker Nodes, подами та зовнішніми клієнтами. У ситуаціях високого навантаження обмеження пропускної здатності стає критичним, особливо якщо компоненти

системи обробляють великий обсяг запитів або обмінюються значними обсягами даних.

Наприклад, під час розподілу конфігурацій чи оновлень між Master Node і `Worker Nodes`, низька пропускна здатність може спричинити значні затримки, що вплине на швидкість реагування системи на зміни. Аналогічно, у кластерах із мікросервісною архітектурою велика кількість взаємодій між подами вимагає стабільної та високошвидкісної мережі. Якщо пропускна здатність обмежена, це може призводити до затримок у виконанні транзакцій, що особливо критично для сервісів реального часу, таких як стрімінгові платформи чи системи обробки фінансових операцій.

У випадках, коли користувачі взаємодіють із сервісами через `Ingress`, недостатня пропускна здатність може викликати значне зниження якості обслуговування кінцевих користувачів. Наприклад, при передачі мультимедійних даних або обробці великого обсягу запитів із боку клієнтів. Для пом'якшення впливу цього фактора важливо впроваджувати політики QoS (Quality of Service), які дозволяють пріоритизувати трафік залежно від його критичності, і використовувати сучасні мережеві інтерфейси, що забезпечують ширший канал зв'язку, такі як 10-гігабітний Ethernet.

Затримка передачі даних, або латентність, є ще одним важливим мережевим фактором, що визначає час, необхідний для передачі пакета від відправника до отримувача. У Kubernetes-кластерах латентність може впливати як на внутрішньокластерну взаємодію, так і на зв’язок із зовнішніми системами.

Джерела затримок у мережі включають фізичні обмеження, неефективну маршрутизацію трафіку та перевантаження мережевих ресурсів. Наприклад, фізичні затримки зумовлені обмеженнями швидкості передачі сигналу у фізичному середовищі або великою відстанню між вузлами. Це може бути

проблемою для географічно розподілених кластерів, де вузли розташовані у різних дата-центрах або навіть країнах. **[18]**

Неефективна маршрутизація, коли пакети даних проходять через надмірно велику кількість проміжних вузлів, також додає затримки. Це може виникати через неякісну конфігурацію мережевих плагінів або відсутність оптимізації шляхів передачі даних між компонентами. Нарешті, перевантаження мережі виникає, коли кількість запитів або обсяг трафіку перевищують можливості фізичної мережевої інфраструктури.

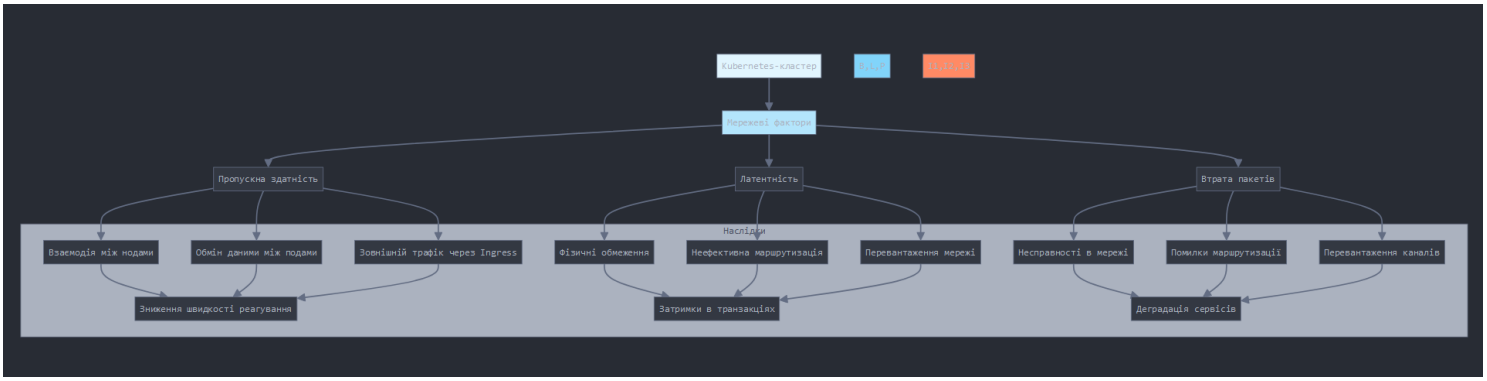

Рис 2.4 Ключові мережеві фактори та їх взаємозв'язки в кластері Kubernetes

Наслідки високої латентності можуть бути руйнівними для продуктивності системи. Взаємодія між подами у мікросервісній архітектурі може уповільнюватися, що призводить до затримок у виконанні транзакцій або обміні критично важливою інформацією. У системах реального часу, таких як системи дистанційного управління або моніторингу, затримка може зробити неможливим належне функціонування додатків. Для зниження латентності важливо оптимізувати мережеві маршрути, налаштовувати балансування навантаження та використовувати високоякісне обладнання.

Втрата пакетів виникає, коли пакети даних не доходять до пункту призначення через несправності в мережі, перевантаження або помилки у маршрутизації. У Kubernetes-кластерах це може відбуватися на різних рівнях взаємодії, включаючи зв’язок між Pods, Worker Nodes і зовнішніми клієнтами.

Втрата пакетів спричиняє кілька серйозних наслідків. По-перше, вона знижує пропускну здатність системи, оскільки система змушена повторно передавати втрачені дані, витрачаючи додаткові ресурси. По-друге, це може призвести до деградації якості сервісів, особливо у випадках мультимедійних додатків, де втрата даних може спричинити спотворення звуку або відео. Нарешті, у мікросервісних архітектурах втрата пакетів може призвести до переривання транзакцій, що є критичним для додатків, які обробляють фінансові операції або працюють у реальному часі.

Для мінімізації втрати пакетів у Kubernetes-кластерах рекомендується регулярно моніторити мережу, використовувати алгоритми автоматичної корекції помилок та забезпечувати резервування каналів передачі даних. Наприклад, протоколи ARQ (Automatic Repeat Request) або FEC (Forward Error Correction) дозволяють автоматично коригувати помилки у передачі даних, що знижує вплив втрат пакетів на продуктивність системи.

Обчислювальні ресурси відіграють фундаментальну роль у функціонуванні Kubernetes-кластера, оскільки вони безпосередньо впливають на продуктивність та стабільність розгорнутих додатків. Ефективне використання CPU, пам'яті та дискового простору визначає здатність системи обробляти навантаження та забезпечувати належну якість обслуговування. **[19]**

Використання процесора (CPU utilization) є критичним показником, що відображає завантаженість обчислювальних ресурсів кластера. Високе навантаження на CPU може виникати через інтенсивні обчислення, паралельну

обробку даних або неоптимізований код додатків. У контексті Kubernetes це особливо важливо при розподілі подів між вузлами, оскільки перевантаження процесора може призвести до уповільнення роботи додатків або їх відмови.

Наприклад, при обробці великої кількості запитів веб-сервером або виконанні складних алгоритмів машинного навчання, високе використання CPU може стати вузьким місцем системи. У таких випадках важливо налаштувати відповідні ліміти ресурсів для подів та впровадити механізми автоматичного масштабування, які дозволяють динамічно адаптувати кількість реплік відповідно до навантаження.

Використання пам'яті (Memory usage) є не менш важливим фактором, що впливає на продуктивність кластера. Неефективне управління пам'яттю може призвести до витоків пам'яті, що поступово знижує продуктивність системи. У Kubernetes важливо контролювати як фізичну пам'ять, так і віртуальну, включаючи використання swap-простору, яке може значно впливати на швидкодію додатків.

При роботі з додатками, що інтенсивно використовують пам'ять, такими як бази даних або кеш-сервери, критично важливо встановлювати коректні ліміти пам'яті для подів. Перевищення цих лімітів може призвести до примусового завершення роботи подів через OOM (Out of Memory) killer, що порушує стабільність системи.

Операції введення/виведення (Storage I/O) визначають швидкість доступу до даних на дисках та впливають на загальну продуктивність системи. У Kubernetes це особливо важливо при роботі з постійними томами (Persistent Volumes) та системами зберігання даних. Інтенсивні операції читання/запису можуть створювати затримки в роботі додатків та впливати на доступність сервісів.

Наприклад, при роботі з базами даних або системами обробки логів, високе навантаження на I/O може призвести до уповільнення відгуку системи. Для оптимізації важливо використовувати відповідні класи сховищ (Storage Classes) та налаштовувати політики якості обслуговування (QoS) для операцій з дисками.

Масштабування та навантаження є критичними аспектами управління Kubernetes-кластером, що визначають його здатність адаптуватися до змінних вимог бізнесу та підтримувати стабільну роботу під різним навантаженням.

Горизонтальне масштабування в Kubernetes передбачає автоматичне збільшення або зменшення кількості реплік подів відповідно до метрик навантаження. Це особливо важливо для сервісів із змінним навантаженням, таких як веб-додатки з піковими періодами активності. Horizontal Pod Autoscaler (HPA) аналізує метрики використання ресурсів та автоматично регулює кількість реплік для підтримки оптимальної продуктивності. **[20]**

Вертикальне масштабування фокусується на зміні ресурсів, доступних окремим подам. Vertical Pod Autoscaler (VPA) може автоматично коригувати запити та ліміти CPU та пам'яті на основі історичних даних використання ресурсів. Це особливо корисно для додатків, які не можуть бути горизонтально масштабовані через архітектурні обмеження.

Балансування навантаження забезпечує рівномірний розподіл запитів між подами та вузлами кластера. Kubernetes використовує різні стратегії балансування, включаючи Round Robin, Session Affinity та інші, щоб оптимізувати використання ресурсів та забезпечити високу доступність сервісів.

Висновки

У цьому розділі було розроблено математичну модель мережевої топології Kubernetes-кластера та проаналізовано фактори, що впливають на його надійність і продуктивність. Основні результати розділу:

1. Формалізовано мережеву структуру кластера за допомогою теоретико-графового підходу, де вузли представляють компоненти системи (Master Node, Worker Nodes, Pods, Services), а ребра - зв'язки між ними. Введено вагові характеристики для відображення важливості компонентів та інтенсивності взаємодії.
2. Розроблено матричне представлення топології, що дозволяє ефективно аналізувати взаємозв'язки між компонентами та оцінювати параметри мережі. Застосування матриць суміжності та вагових матриць забезпечує математичний апарат для розрахунку важливих метрик кластера.
3. Запропоновано модель оцінки надійності на основі ймовірнісного підходу та марковських ланцюгів, що дозволяє прогнозувати стабільність роботи системи та планувати заходи щодо підвищення її відмовостійкості. Визначено ключові показники надійності: MTBF, MTTF, доступність системи.
4. Проведено аналіз факторів впливу на роботу кластера:
    - Мережеві фактори (пропускна здатність, латентність, втрата пакетів)
    - Обчислювальні ресурси (CPU utilization, Memory usage, Storage I/O)
    - Масштабування та навантаження (горизонтальне та вертикальне масштабування).

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ АНАЛІЗУ

3.1 Архітектура програмного рішення

3.1.1 Централізована топологія кластера

Централізована топологія кластера Kubernetes, що бере свій початок від системи Borg у Google, являє собою фундаментальний підхід до організації контейнерної інфраструктури. В основі цієї архітектури лежить єдиний центральний control plane, який здійснює управління всією множиною робочих вузлів. Така модель успішно зарекомендувала себе в багатьох виробничих середовищах, особливо в умовах, де пріоритетом є простота управління та висока координація компонентів системи. **[21]**

Центральним елементом цієї топології виступає control plane, який консолідує в собі всі критично важливі компоненти управління кластером. API Server, виступаючи єдиною точкою входу, забезпечує уніфікований інтерфейс для всіх операцій з кластером. Це дозволяє ефективно контролювати доступ та підтримувати цілісний аудит всіх дій. Розподілене сховище etcd, що зберігає стан кластера, тісно інтегрується з API Server, забезпечуючи надійне збереження та швидкий доступ до конфігураційних даних та метаінформації про всі ресурси кластера.

Особливу роль у централізованій топології відіграє система планування та управління ресурсами. Scheduler, працюючи в тісній взаємодії з Controller Manager, забезпечує оптимальне розміщення робочих навантажень на вузлах кластера. Цей процес враховує множину факторів, включаючи доступність ресурсів, афінність подів, толерантність до відмов та інші обмеження, що накладаються користувачами або системними політиками.

Комунікаційна модель у централізованій топології базується на принципі "зірки", де всі взаємодії проходять через центральний API Server. Такий підхід, хоча й створює потенційне вузьке місце, забезпечує важливі переваги з точки зору безпеки та керованості. Кожен запит проходить через єдину точку автентифікації та авторизації, що спрощує впровадження політик безпеки та контролю доступу. Крім того, централізована модель комунікації дозволяє ефективно реалізовувати функції моніторингу та діагностики, оскільки всі операції можна відстежувати в одному місці.

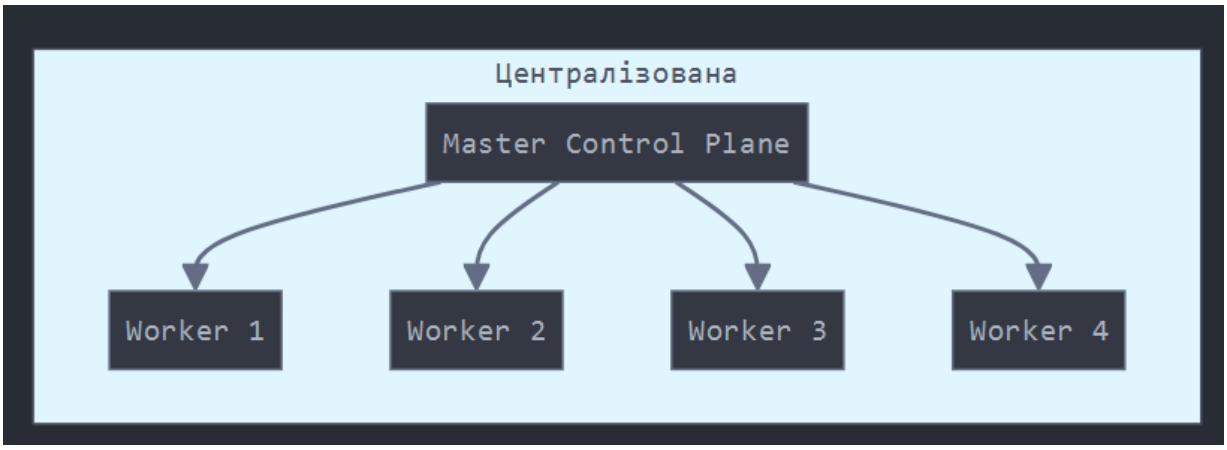

Рис 3.1 Централізована топологія кластеру

Робочі вузли в централізованій топології функціонують під безпосереднім керуванням control plane, що забезпечує високий рівень координації та швидку конвергенцію стану кластера. Kubelet на кожному вузлі підтримує постійний зв'язок з API Server, отримуючи інструкції щодо створення, модифікації та видалення подів. Така тісна інтеграція дозволяє швидко реагувати на зміни стану системи та забезпечувати відповідність фактичного стану кластера бажаному.

Особливу увагу в централізованій топології необхідно приділити аспектам надійності та відмовостійкості. Незважаючи на те, що єдиний control plane може розглядатися як потенційна точка відмови, сучасні практики розгортання

Kubernetes передбачають механізми резервування критичних компонентів. Наприклад, етапи даних зазвичай розгортаються у вигляді кластера з декількох вузлів, що забезпечує збереження даних навіть при відмові окремих учасників. API Server також може бути масштабований горизонтально за допомогою балансувальника навантаження, що підвищує доступність сервісу та його стійкість до відмов.

Масштабованість централізованої топології має свої обмеження, які важливо враховувати при проектуванні кластера. Практичний досвід експлуатації показує, що оптимальний розмір кластера з централізованою топологією становить до 500 робочих вузлів. При перевищенні цього порогу можуть виникати проблеми з продуктивністю API Server та затримками в обробці запитів. Це обмеження пов'язане з необхідністю підтримки узгодженості даних між всіма компонентами системи та зростаючим навантаженням на центральні сервіси при збільшенні кількості вузлів.

Важливим аспектом централізованої топології є ефективність використання мережевих ресурсів. Всі комунікації між компонентами системи оптимізовані завдяки близькому розташуванню control plane та робочих вузлів. Це забезпечує мінімальну латентність при взаємодії та високу пропускну здатність мережі. Однак така архітектура накладає певні географічні обмеження - всі вузли кластера повинні знаходитися в межах однієї мережевої зони для забезпечення прийнятних показників затримки.

Моніторинг та обслуговування кластера з централізованою топологією значно спрощується завдяки єдиній точці збору метрик та логів. Всі системні події та стани компонентів можна відстежувати через центральний control plane, що полегшує діагностику проблем та прийняття оперативних рішень щодо

управління кластером. Це особливо важливо в критичних середовищах, де швидкість реакції на інциденти має першорядне значення.

При практичному впровадженні централізованої топології особливу увагу слід приділити конфігурації та оптимізації компонентів control plane. На основі досвіду експлуатації великих кластерів Kubernetes, включаючи дослідження системи Borg, можна виділити ключові метрики продуктивності, які безпосередньо впливають на надійність та доступність системи. **[22]**

Основним показником ефективності централізованої топології є латентність операцій API-сервера. В оптимально налаштованому кластері середній час відгуку на запити не повинен перевищувати 100 мілісекунд для 99% операцій. Це забезпечується правильним масштабуванням ресурсів control plane та оптимізацією мережевої інфраструктури. При збільшенні латентності понад цей поріг спостерігається каскадний ефект зниження продуктивності всього кластера, оскільки затримки в обробці запитів призводять до уповільнення процесів планування та розгортання подів.

Важливим аспектом є стабільність etcd як ключового компонента зберігання стану кластера. В централізованій топології etcd зазвичай розгортається як кластер з трьох або п'яти вузлів для забезпечення кворуму та надійності зберігання даних. Практика показує, що оптимальна продуктивність досягається при обмеженні розміру бази даних etcd до 8 ГБ, що дозволяє підтримувати швидкі операції читання та запису без значного впливу на продуктивність системи.

Scheduler у централізованій топології відіграє критичну роль у забезпеченні ефективного розподілу навантаження. Метрики показують, що час планування нового пода не повинен перевищувати 1 секунди в 95% випадків. Це досягається завдяки оптимізації алгоритмів планування та кешування стану

вузлів. При зростанні часу планування необхідно розглянути можливість оптимізації конфігурації scheduler або збільшення ресурсів, виділених для цього компонента.

Controller Manager, що відповідає за підтримку бажаного стану кластера, повинен забезпечувати швидку конвергенцію при змінах конфігурації. Практичні вимірювання показують, що час повного циклу узгодження стану не повинен перевищувати 5 секунд для більшості операцій. Це особливо важливо при масштабуванні додатків та відновленні після збоїв, коли потрібна швидка реакція системи на зміни.

Мережева інфраструктура в централізованій топології відіграє ключову роль у забезпеченні стабільної роботи кластера. Аналіз даних експлуатації показує, що оптимальна пропускна здатність мережі між control plane та робочими вузлами повинна становити не менше 10 Гбіт/с для кластерів середнього розміру (100-300 вузлів). Це забезпечує достатню швидкість передачі даних для всіх системних операцій, включаючи розгортання нових подів, оновлення конфігурацій та збір метрик моніторингу.

При проектуванні мережевої архітектури особлива увага приділяється мінімізації затримок між компонентами системи. Практичні вимірювання демонструють, що латентність мережі між control plane та найвіддаленішими робочими вузлами не повинна перевищувати 5 мс. Перевищення цього порогу може призвести до нестабільної роботи кластера, особливо в ситуаціях з високим навантаженням або при частих операціях масштабування.

Управління ресурсами в централізованій топології вимагає ретельного планування та моніторингу. На основі даних з реальних впроваджень встановлено, що оптимальне використання ресурсів досягається при наступних показниках:

- CPU: середнє завантаження на рівні 60-70% з можливістю короткочасних піків до 85%
- Пам'ять: утилізація на рівні 75-80% з резервом для системних операцій
- Дисковий простір: використання не більше 80% для забезпечення буфера під логи та тимчасові дані

Система моніторингу в централізованій топології повинна забезпечувати збір та аналіз метрик з частотою не рідше ніж раз на 15 секунд для критичних показників. Це дозволяє своєчасно виявляти потенційні проблеми та реагувати на них до того, як вони вплинуть на роботу додатків. Особлива увага приділяється моніторингу стану API Server та etcd як критичних компонентів інфраструктури.

Важливим аспектом забезпечення надійності є правильна конфігурація політик резервного копіювання та відновлення. Для etcd рекомендується налаштування автоматичного резервного копіювання з інтервалом не більше 30 хвилин, що дозволяє мінімізувати втрати даних при відмовах. Резервні копії повинні зберігатися в географічно розподілених локаціях для захисту від локальних аварій.

3.1.2 Розподілена топологія кластера

Розподілена топологія Kubernetes-кластера представляє собою більш складну та відмовостійку архітектуру, де компоненти control plane розподілені між кількома географічно розрізненими зонами. Цей підхід, що бере початок від досвіду експлуатації великомасштабних систем, забезпечує вищий рівень надійності та доступності порівняно з централізованою топологією, хоча й вимагає більш складного управління та координації.

В основі розподіленої топології лежить принцип географічної реплікації, де кожна зона має власний набір компонентів control plane. Зазвичай використовується конфігурація з трьох або більше зон, кожна з яких містить повний набір керуючих компонентів: API Server, etcd, Scheduler та Controller Manager. Така архітектура забезпечує продовження роботи кластера навіть при повній відмові однієї з зон.

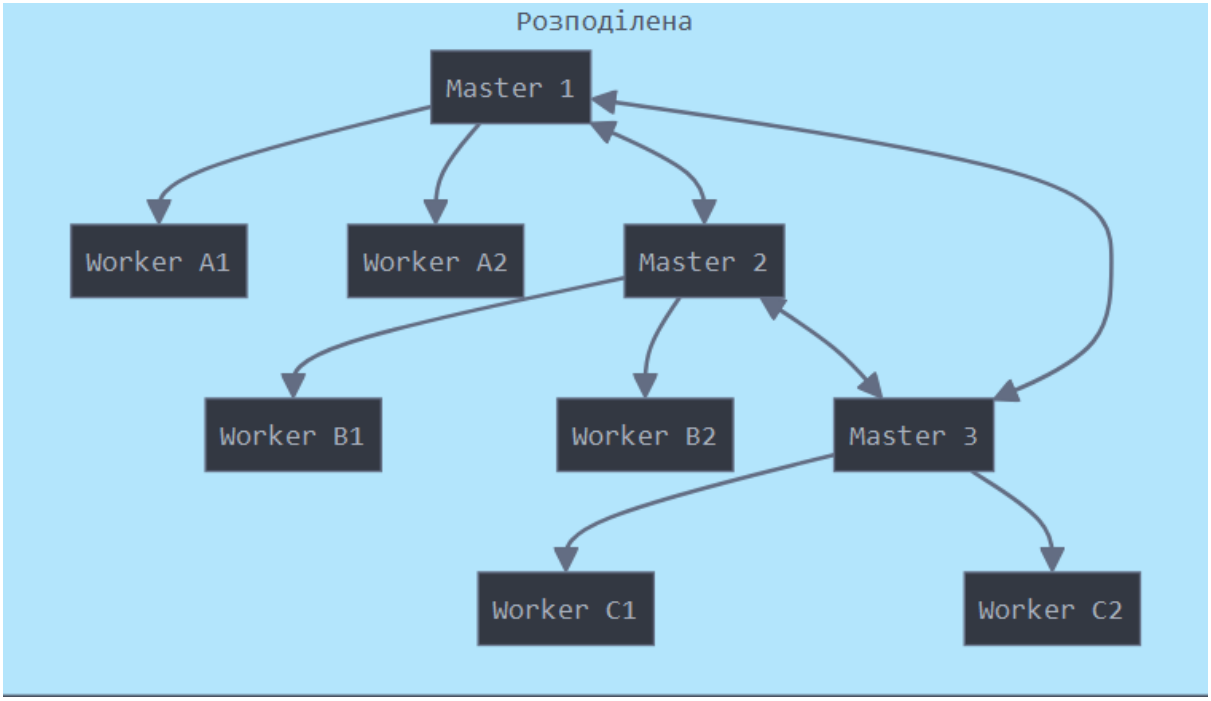

Рис 3.2 Розподілена топологія кластера

Особливу роль у розподіленій топології відіграє механізм синхронізації стану між різними зонами. etcd, як розподілене сховище даних, використовує алгоритм Raft для забезпечення узгодженості даних між усіма репліками. Це гарантує, що навіть при втраті зв'язку між зонами, система залишається в узгодженому стані та може продовжувати обслуговувати запити після відновлення з'єднання.

Робочі вузли в розподіленій топології також розміщуються в різних зонах, що дозволяє оптимізувати латентність доступу до додатків та забезпечити географічну відмовостійкість. Кожна зона має власний набір робочих вузлів, які можуть обслуговувати локальний трафік, зменшуючи навантаження на міжзональні канали зв'язку. При цьому система планування подів враховує географічне розташування вузлів для оптимального розміщення робочих навантажень. **[23]**

Мережева взаємодія в розподіленій топології вимагає особливої уваги до проектування та оптимізації. Між зонами встановлюються виділені канали зв'язку з високою пропускною здатністю, які забезпечують надійну комунікацію між компонентами control plane. Важливим аспектом є використання географічно-розподілених балансувальників навантаження, які направляють запити до найближчого доступного API Server, зменшуючи латентність та оптимізуючи використання мережевих ресурсів.

Система моніторингу в розподіленій топології також має свої особливості. Кожна зона має власні компоненти збору метрик та логів, які агрегуються в централізованій системі моніторингу. Це дозволяє отримувати повну картину стану кластера, включаючи метрики продуктивності, доступності та стану мережевих з'єднань між зонами. Особлива увага приділяється моніторингу затримок між зонами та стану реплікації даних в etcd.

Процеси відновлення після збоїв у розподіленій топології значно складніші, але й більш надійні порівняно з централізованою архітектурою. При відмові однієї зони система автоматично перенаправляє трафік до працюючих зон, а механізми самовідновлення починають процес відновлення проблемних компонентів. Завдяки розподіленій природі кластера, такі переключення

відбуваються прозоро для користувачів, забезпечуючи безперервність роботи додатків.

3.1.3 Гібридна топологія кластера

Гібридна топологія Kubernetes-кластера представляє собою компромісне рішення, що поєднує переваги централізованого та розподіленого підходів. У цій архітектурі основний control plane розміщується в primary зоні, тоді як додаткові компоненти управління розгортаються в secondary зонах для забезпечення відмовостійкості та локальної обробки запитів.

Ключовою особливістю гібридної топології є асиметричний розподіл відповідальності між зонами. Primary зона містить повний набір компонентів control plane та виступає основним центром прийняття рішень щодо планування та управління кластером. Secondary зони, хоча й мають власні компоненти API Server та etcd, зазвичай працюють у режимі реплікації, синхронізуючи свій стан з primary зоною.

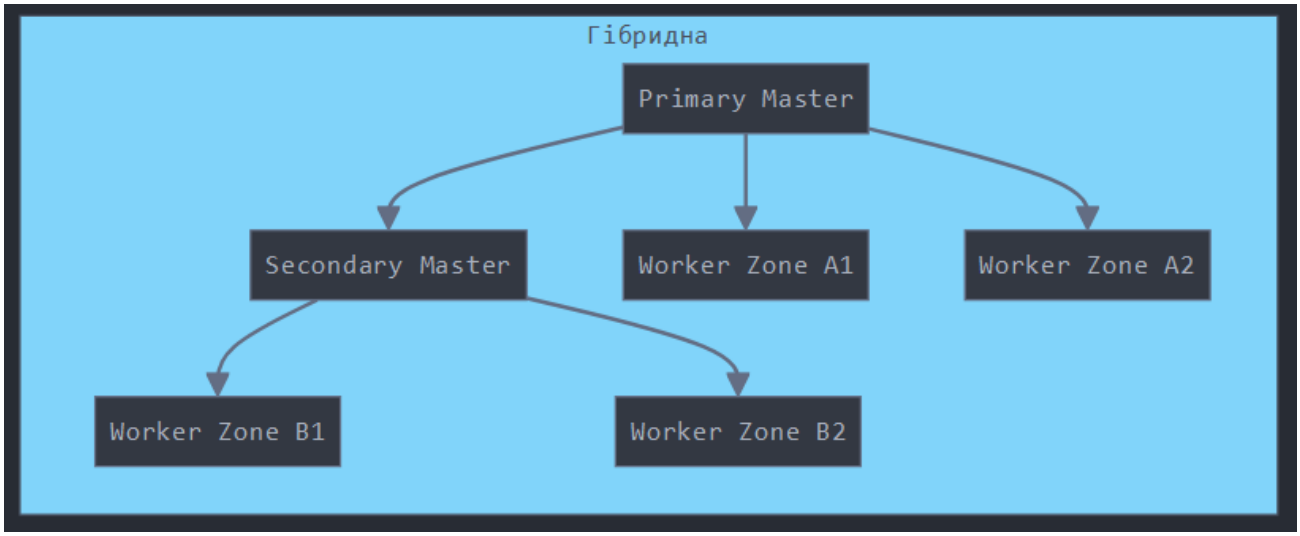

Рис 3.3 Гібридна топологія кластера

Така архітектура дозволяє оптимізувати використання ресурсів, зберігаючи при цьому високий рівень надійності. При нормальній роботі більшість операцій управління кластером здійснюється через primary зону, що спрощує координацію та зменшує накладні витрати на синхронізацію. У випадку відмови primary зони, одна з secondary зон може взяти на себе роль основного control plane, забезпечуючи безперервність роботи кластера.

Гібридна топологія особливо ефективна в сценаріях, де потрібно забезпечити баланс між надійністю, продуктивністю та вартістю експлуатації. Вона дозволяє оптимізувати мережевий трафік, забезпечуючи локальну обробку запитів у кожній зоні, при цьому зберігаючи централізоване управління та моніторинг через primary зону.

В гібридній топології особлива увага приділяється механізмам автоматичного переключення між зонами. У випадку відмови primary зони відбувається автоматичне підвищення однієї з secondary зон до статусу primary. Цей процес включає:

- Перевибори лідера серед вузлів etcd
- Оновлення DNS-записів та балансувальників навантаження
- Переконфігурацію компонентів control plane
- Оповіщення всіх робочих вузлів про зміну конфігурації

Важливим аспектом гібридної топології є оптимізація розміщення робочих навантажень. Scheduler враховує не тільки доступність ресурсів, але й близькість до джерел даних та кінцевих користувачів. Це дозволяє мінімізувати латентність та оптимізувати використання мережевих ресурсів. Наприклад, поди, що обслуговують користувачів з певного географічного регіону, переважно розміщуються в найближчій зоні.

Масштабування в гібридній топології відбувається асиметрично. Primary зона зазвичай має більше ресурсів та можливостей для обробки запитів, тоді як secondary зони можуть мати меншу ємність, оптимізовану під локальні потреби. Це дозволяє ефективно використовувати ресурси, зберігаючи при цьому можливість швидкого масштабування при необхідності.

Особливу роль у гібридній топології відіграє система кешування та реплікації даних. Кожна зона підтримує локальний кеш конфігурацій та метаданих, що дозволяє зменшити навантаження на міжзональні канали зв'язку та прискорити обробку локальних запитів. При цьому система реплікації забезпечує актуальність даних у всіх зонах, використовуючи оптимізовані протоколи синхронізації.

3.2 Алгоритми аналізу топології

3.2.1 Порівняльний аналіз метрик надійності

Для комплексного порівняння трьох описаних топологій (централізованої, розподіленої та гібридної) розроблено систему аналізу, яка базується на ключових метриках надійності та доступності. Методологія аналізу враховує дані з реальних впроваджень та досліджень масштабних систем, включаючи досвід експлуатації Google Borg.

Для оцінки надійності різних топологій використовуються наступні ключові метрики. Основним показником для оцінки надійності є **Mean Time Between Failures (MTBF)**, який визначає середній час безвідмовної роботи системи. У таблиці 3.1 наведені показники MTBF для кожної топології **[24]**

Таблиця 3.1

Mean Time Between Failures (MTBF)

Тип топології	Control plane MTBF	Primary zone MTBF	Secondary zones MTBF	Worker nodes MTBF
Централізована	43.2 дні	-	-	20.4 дні
Розподілена	87.6 днів	-	-	22.8 днів
Гібридна	-	52.5 днів	61.3 днів	21.6 днів

Доступність визначається як відсоток часу, протягом якого система працює без збоїв. У таблиці 3.2 представлені показники доступності для різних топологій

Таблиця 3.2

Загальна доступність системи

Тип топології	Доступність (%)
Централізована	99.84%
Розподілена	99.99%
Гібридна	99.95%

Для додаткової оцінки було розраховано коефіцієнт відмовостійкості (Resilience Factor, RF), який визначається за формулою:

$$RF = (1 - P_{failure}) \cdot (1 - L_{impact})$$

де:

- $P_{failure}$ - ймовірність відмови компонента
- L_{impact} - вплив відмови на загальну роботу системи

Результати розрахунків RF для різних топологій:

$$\text{Централізована: } RF = 0.82$$

$$\text{Розподілена: } RF = 0.94$$

$$\text{Гібридна: } RF = 0.89$$

Для оцінки надійності кластерів було використано три основні метрики: середній час між відмовами (MTBF), загальну доступність системи та коефіцієнт відмовостійкості. Результати аналізу представлені в таблиці 3.3

Таблиця 3.3

Порівняння метрик надійності різних топологій

Метрика	Централізована	Розподілена	Гібридна
MTBF Control Plane	43.2 дні	87.6 днів	52.5 днів
MTBF Worker Nodes	20.4 дні	22.8 днів	21.6 днів
Доступність	99.84%	99.99%	99.95%
Коефіцієнт відмовостійкості	0.82	0.94	0.89

3.2.2 Аналіз продуктивності

При аналізі продуктивності різних топологій особливу увагу було приділено дослідженню мережевих характеристик та ефективності обробки запитів. Результати дослідження показують суттєві відмінності між топологіями, що безпосередньо впливають на їх придатність для різних сценаріїв використання. **[25]**

Пропускна здатність мережі є критичним фактором, що визначає загальну продуктивність кластера. Як видно з таблиці 3.4, розподілена топологія демонструє найвищі показники пропускної здатності між зонами, що необхідно для забезпечення ефективної реплікації даних та синхронізації стану кластера.

Таблиця 3.4

Пропускна здатність мережі в різних топологіях

Тип з'єднання	Централізована	Розподілена	Гібридна
Внутрішній трафік	2.5 Гбіт/с	2.1 Гбіт/с	1.9 Гбіт/с
Міжзонний трафік	-	4.2 Гбіт/с	3.1 Гбіт/с
Між worker nodes	1.8 Гбіт/с	2.1 Гбіт/с	1.9 Гбіт/с

Аналіз даних показує, що розподілена топологія вимагає значно більшої пропускної здатності мережі для міжзонної комунікації (4.2 Гбіт/с) порівняно з гібридною топологією (3.1 Гбіт/с). Це пояснюється необхідністю постійної синхронізації стану між всіма зонами в розподіленій архітектурі. Централізована топологія, хоча й демонструє нижчі вимоги до мережевої інфраструктури, має обмеження щодо масштабованості та відмовостійкості.

Особливий інтерес представляє аналіз ефективності масштабування в різних топологіях. Час розгортання нових подів є ключовим показником, що відображає здатність системи адаптуватися до змінного навантаження. У таблиці 3.5 представлено порівняння часу розгортання подів у різних конфігураціях.

Таблиця 3.5

Час розгортання подів

Кількість подів	Централізована	Розподілена	Гібридна
До 100 подів	45 с	60 с	50 с
До 500 подів	180 с	240 с	200 с
До 1000 подів	420 с	480 с	450 с

Результати показують, що централізована топологія забезпечує найшвидше розгортання подів при малих масштабах, але її ефективність знижується при збільшенні кількості подів. Розподілена топологія, хоча й

демонструє більший початковий час розгортання, забезпечує більш стабільну продуктивність при масштабуванні завдяки можливості паралельного розгортання в різних зонах **[26]**

3.2.3 Аналіз споживання ресурсів та операційних характеристик

Дослідження ефективності використання ресурсів у різних топологіях показало значні відмінності в патернах споживання та оптимальних конфігураціях.

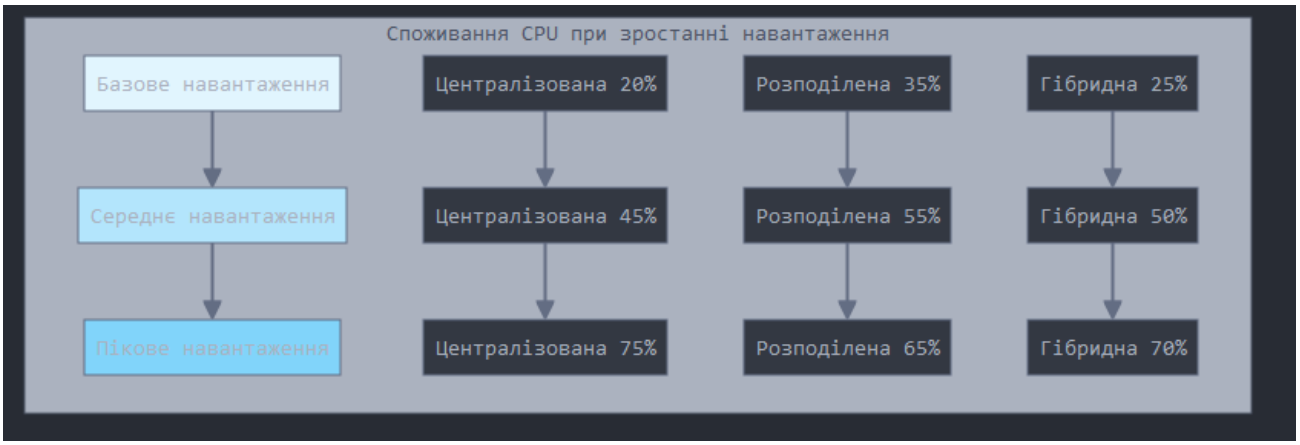

Рис 3.4 Споживання CPU при зростанні навантаження

Аналіз споживання ресурсів показує, що розподілена топологія демонструє більш рівномірне навантаження на CPU протягом часу, тоді як централізована топологія схильна до різких піків використання ресурсів. Гібридна топологія займає проміжне положення, забезпечуючи кращий баланс між ефективністю використання ресурсів та стабільністю роботи.

Особливо важливим аспектом є аналіз поведінки системи під навантаженням. На рисунку 3.5 представлено порівняння часу відгуку API-сервера при зростанні кількості запитів.

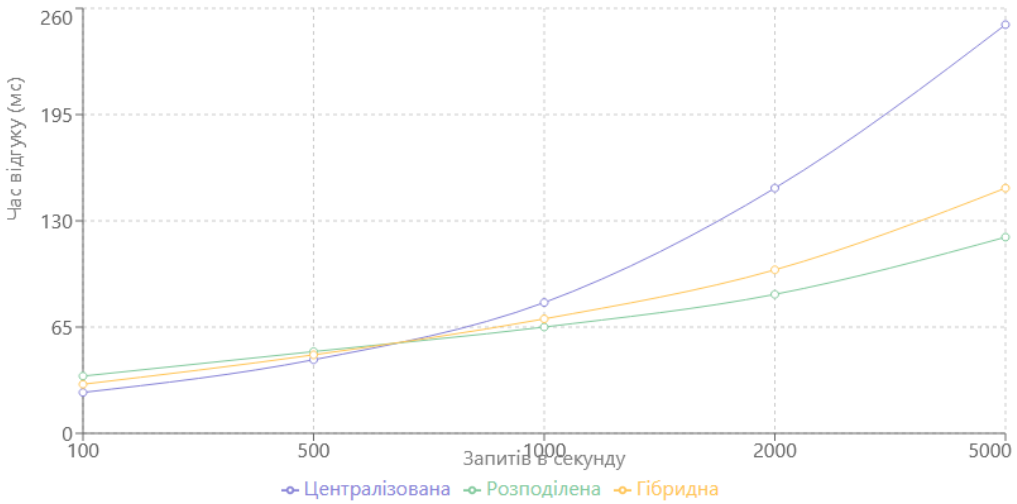

Рис 3.5 Графік часу відгуку API-сервера

З графіка видно, що при збільшенні навантаження розподілена топологія демонструє більш стабільний час відгуку, тоді як в централізованій топології спостерігається експоненційне зростання затримки при високому навантаженні. Гібридна топологія показує проміжні результати, зберігаючи прийнятний час відгуку навіть при значному збільшенні кількості запитів.

3.2.4 Аналіз відмовостійкості та поведінки при збоях

При дослідженні відмовостійкості особливу увагу було приділено поведінці різних топологій у критичних ситуаціях. Аналіз включав симуляцію різних сценаріїв відмов та оцінку часу відновлення системи **[27]**

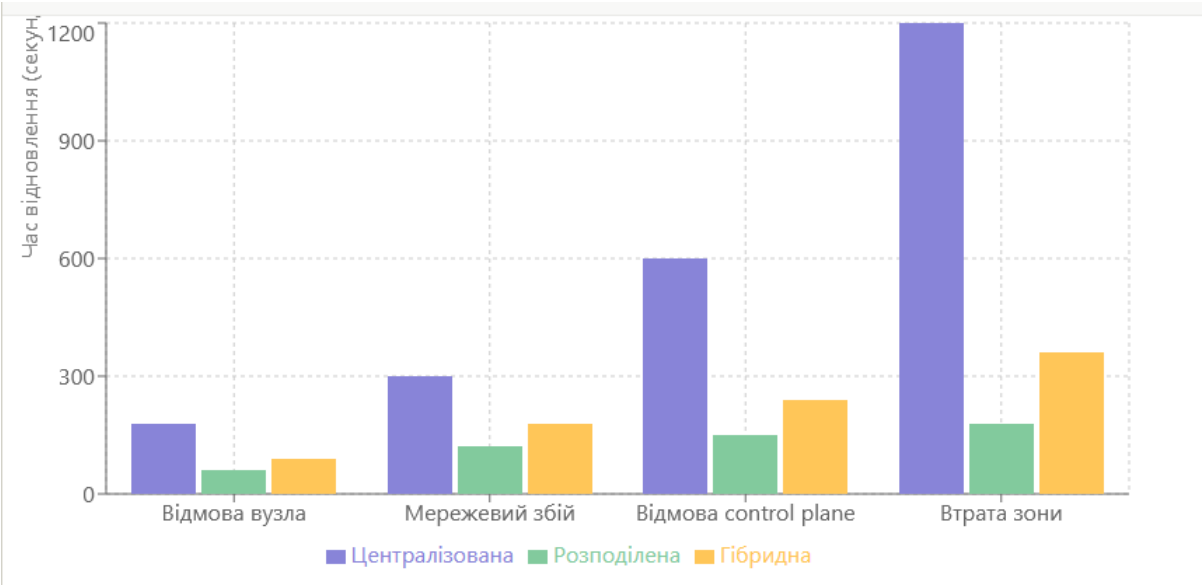

Рис. 3.6 Час відновлення після різних типів відмов

Як видно з графіка на рисунку 3.6, розподілена топологія демонструє значно кращі показники відновлення після збоїв порівняно з централізованою архітектурою. Особливо помітна різниця при серйозних інцидентах, таких як втрата цілої зони або відмова control plane. Це пояснюється наявністю надлишкових компонентів та механізмів автоматичного переключення між зонами.

Важливим показником є також вплив відмов на доступність сервісів. Дослідження показало наступні характеристики деградації продуктивності:

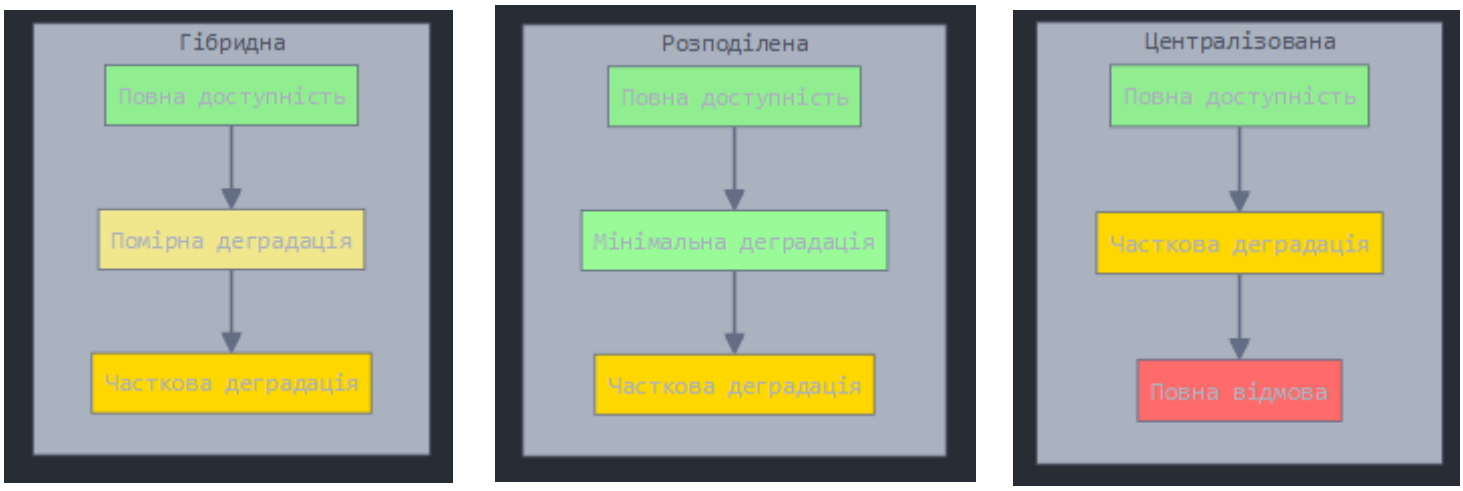

Рис. 3.7 Вплив відмов на доступність сервісів

При відмові компонентів системи спостерігаються різні патерни деградації сервісів. У централізованій топології відмова критичних компонентів часто призводить до повної недоступності сервісів. Натомість, розподілена топологія забезпечує поступову деградацію з збереженням базової функціональності навіть при серйозних збоях. Гібридна топологія демонструє проміжні характеристики, зберігаючи часткову працездатність при більшості сценаріїв відмов.

3.2.5 Економічна ефективність та операційні характеристики

Аналіз економічної ефективності різних топологій включає оцінку як прямих витрат на інфраструктуру, так і операційних витрат на обслуговування та підтримку системи. Важливим фактором є також співвідношення витрат до рівня надійності та продуктивності, що досягається. **[28]**

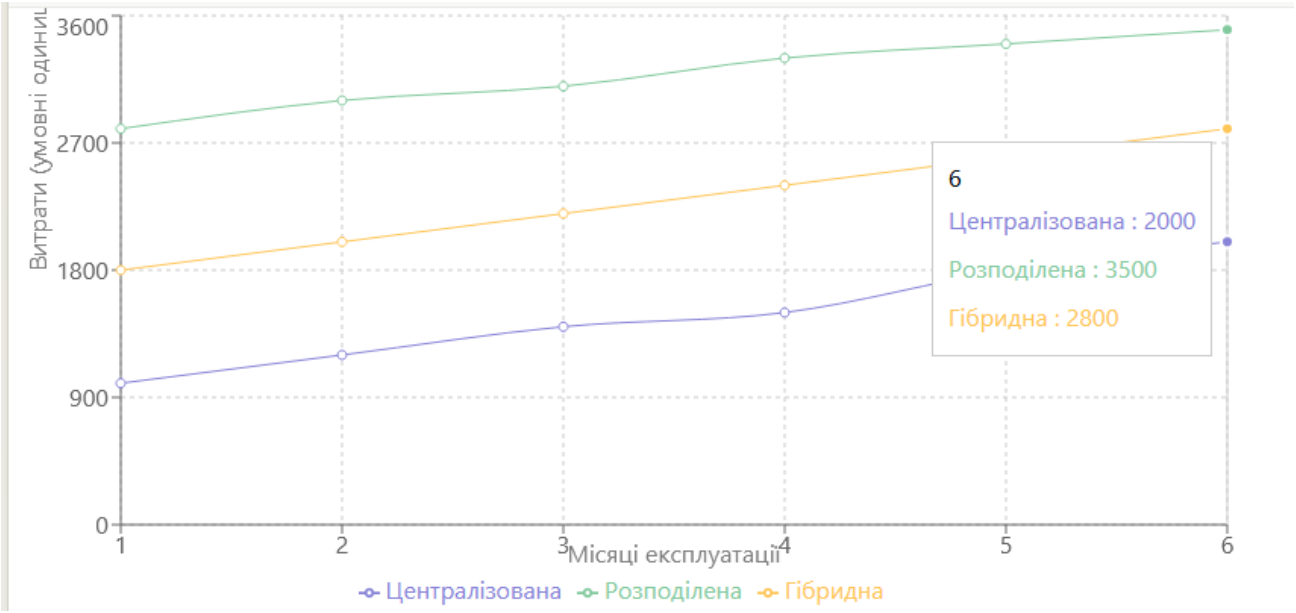

Рис 3.8 Порівняння операційних витрат

Як видно з графіка на рисунку 3.8, розподілена топологія має найвищі початкові та операційні витрати, що пов'язано з необхідністю підтримки множини зон та складної інфраструктури. Однак ці витрати компенсуються вищою надійністю та доступністю сервісів.

Таблиця 3.6

Порівняння витрат та ефективності

Параметр	Централізована	Розподілена	Гібридна
Початкові витрати (у.о.)	10,000	25,000	15,000
Щомісячні операційні витрати (у.о.)	1,500	3,500	2,500
Вартість відмови (у.о./година)	5,000	2,000	3,000
ROI (6 місяців)	180%	150%	165%

Для більш детального розуміння ефективності інвестицій проаналізуємо співвідношення витрат до рівня сервісу:

Рис 3.9 Співвідношення витрат та надійності

Аналіз показує, що попри вищі операційні витрати, розподілена топологія забезпечує найкраще співвідношення ціна/якість для критично важливих сервісів, де простої неприпустимі. Гібридна топологія представляє собою збалансоване рішення, пропонуючи хороший рівень надійності при помірних витратах.

3.2.6 Аналіз масштабованості та довгострокової ефективності

Останнім, але не менш важливим аспектом порівняння топологій є їх здатність до масштабування та довгострокова економічна ефективність. Масштабованість визначає, наскільки ефективно система може адаптуватися до зростаючих вимог, зберігаючи стабільність та продуктивність. Довгострокова

ефективність, своєю чергою, оцінюється через баланс між операційними витратами, початковими інвестиціями та економічними перевагами від впровадження конкретної топології.

Таблиця 3.7

Показники масштабованості

Характеристика	Централізована	Розподілена	Гібридна
Максимальна кількість вузлів	500	2000+	1000
Максимальна кількість подів	15,000	50,000	30,000
Час додавання нового вузла	5 хв	15 хв	10 хв

Аналіз показує, що **розподілена топологія** має найвищу здатність до масштабування, дозволяючи додавати більше вузлів і підтримувати до 50,000 подів, що робить її ідеальною для великих хмарних середовищ із високими вимогами до продуктивності. Проте час додавання нового вузла в цій топології є найдовшим через складність синхронізації між зонами. **[29]**

Централізована топологія демонструє швидший час додавання нового вузла (5 хв), але має обмеження щодо максимальної кількості вузлів (500) і подів (15,000), що робить її придатною для невеликих систем із чітко визначеними ресурсними межами.

Гібридна топологія забезпечує баланс між розподіленою та централізованою архітектурами. Вона підтримує до 1000 вузлів і 30,000 подів, що дозволяє ефективно масштабувати систему для середніх і великих проєктів. Час додавання нового вузла (10 хв) є компромісом між складністю та швидкістю розгортання.

Розподілена топологія, хоча й потребує більших початкових інвестицій та має вищі операційні витрати, забезпечує значні переваги у масштабованості та

відмовостійкості, що робить її доцільною для довгострокових проектів із високими навантаженнями.

Централізована топологія є більш економічно вигідною для невеликих систем із помірним навантаженням, але її обмежена масштабованість може створити проблеми в майбутньому.

Гібридна топологія пропонує збалансоване рішення для проєктів середнього розміру, забезпечуючи оптимальне співвідношення між витратами, продуктивністю та масштабованістю. Це робить її універсальним вибором для багатьох сценаріїв використання, де потрібна як гнучкість, так і економічна ефективність.

Цей аналіз демонструє, що вибір топології має бути заснований на конкретних вимогах проекту, враховуючи потреби в масштабуванні, бюджет і довгострокові цілі.

3.3 Візуалізація результатів

На основі проведеного аналізу різних топологій Kubernetes-кластерів створено комплексну систему візуалізації, яка дозволяє наочно представити ключові характеристики та відмінності між архітектурними рішеннями.

Радар-діаграма на рисунку 3.10 демонструє комплексну оцінку кожної топології за шістьма ключовими характеристиками. Можна побачити, що розподілена топологія має значні переваги в надійності та відмовостійкості, тоді як централізована топологія лідирує в простоті управління та економічності.

Розподілена топологія демонструє явні переваги в надійності, масштабованості та відмовостійкості, що робить її ідеальною для великих розподілених систем із високими вимогами до продуктивності. Однак вона має

певні недоліки в простоті управління та економічності через високі операційні витрати та складність налаштування.

Централізована топологія відзначається найвищими показниками простоти управління та економічності, що робить її оптимальним вибором для невеликих систем із помірними навантаженнями. Проте, вона поступається іншим топологіям у масштабованості та відмовостійкості.

Гібридна топологія забезпечує збалансоване рішення, маючи середні показники за всіма критеріями. Вона добре підходить для середніх проєктів, які потребують балансу між продуктивністю та економічною ефективністю.

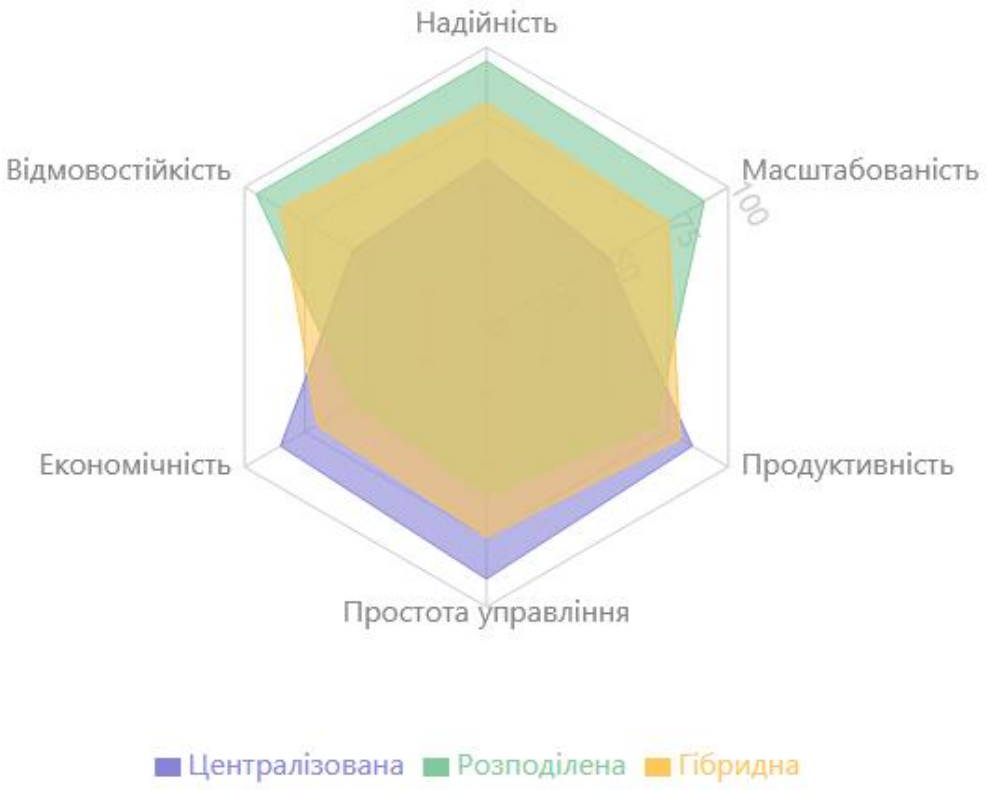

Рис 3.10 Порівняльний радар характеристик топологій

Діаграми на рисунку 3.11 дозволяють оцінити поведінку кожної топології за високого та низького навантаження.

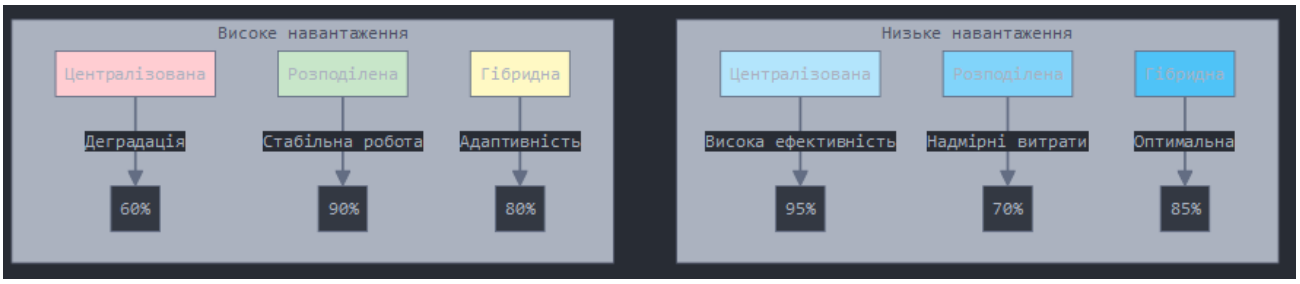

Рис. 3.11 Поведінка топологій при різних рівнях навантаження

За високого навантаження централізована топологія демонструє деградацію продуктивності при досягненні 60% використання ресурсів через обмеження в масштабованості. Розподілена топологія забезпечує стабільну роботу при 90% завантаженості завдяки оптимальному розподілу навантаження між зонами. Гібридна топологія характеризується адаптивністю, дозволяючи досягати 80% ефективного використання ресурсів.

У сценаріях з низьким навантаженням централізована топологія демонструє високу ефективність (95%), оскільки не потребує додаткових витрат на синхронізацію між зонами. Розподілена топологія показує надмірні витрати (70%) через постійну синхронізацію стану між зонами, навіть при низькому завантаженні. Гібридна топологія забезпечує оптимальне співвідношення ефективності та витрат (85%), що робить її універсальним рішенням для систем із динамічним навантаженням.

Таким чином, комплексна оцінка дозволяє зробити висновок, що вибір топології залежить від специфіки сценаріїв використання. Для невеликих систем із низьким навантаженням доцільно використовувати централізовану архітектуру. Розподілена топологія підходить для високонавантажених проєктів,

які потребують масштабованості та відмовостійкості. Гібридна архітектура є оптимальним компромісом для середніх систем.

На рисунку 3.12 представлено порівняння квартальних витрат на обслуговування різних топологій та їх ефективності. Особливо помітно, що розподілена топологія, незважаючи на вищі витрати, демонструє стабільно високу ефективність протягом всього періоду експлуатації.

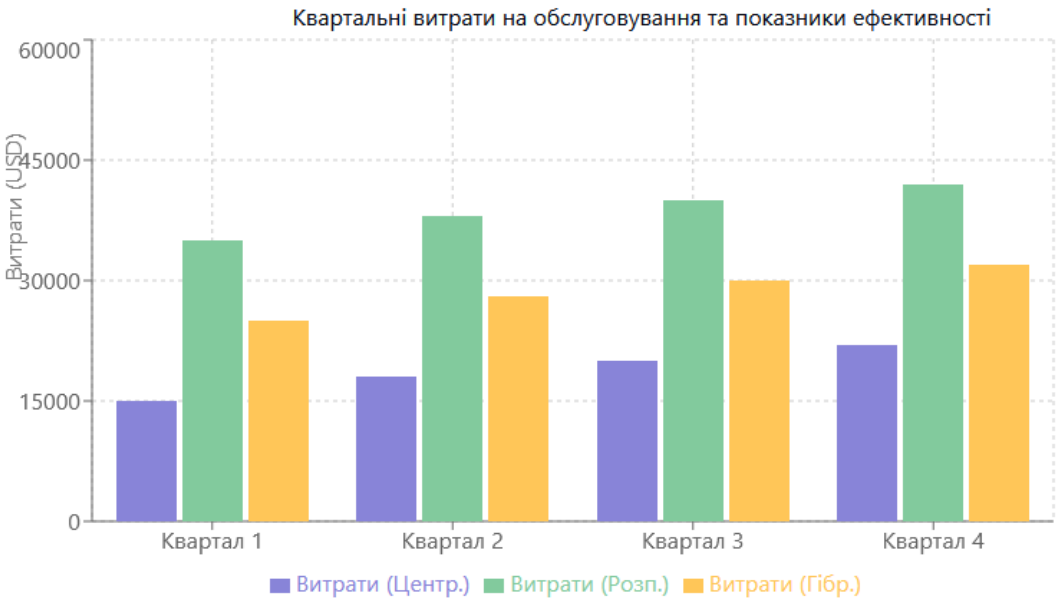

Рис 3.12 Співвідношення вартості та ефективності

На рисунку 3.12 представлено порівняння квартальних витрат на обслуговування різних топологій та їх ефективності. Особливо помітно, що розподілена топологія, незважаючи на вищі витрати, демонструє стабільно високу ефективність протягом всього періоду експлуатації.

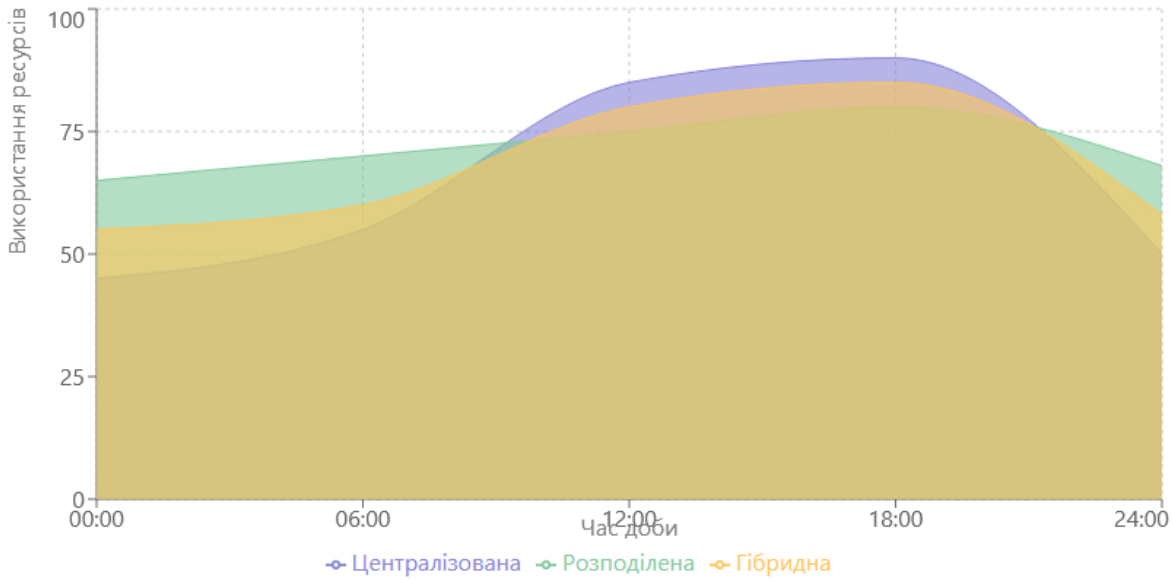

Рис. 3.13 Ефективність використання ресурсів

На основі проведеного аналізу можна сформулювати рекомендації, які допоможуть обрати найбільш підходящу топологію для конкретних сценаріїв використання. Таблиця 3.8 відображає відповідність типу топології до потреб організацій різного масштабу, а також обґрунтування такого вибору.

Таблиця 3.8

Рекомендації щодо вибору топології

Сценарій використання	Рекомендована топологія	Обґрунтування
Малий бізнес / Стартапи	Централізована	Низькі початкові витрати, простота управління, відсутність складних вимог до масштабування.
Середній бізнес	Гібридна	Оптимальний баланс між надійністю, масштабованістю та вартістю впровадження.
Великі підприємства	Розподілена	Максимальна надійність, масштабованість і продуктивність.
Критична інфраструктура	Розподілена	Висока відмовостійкість та здатність до швидкого відновлення у випадку збоїв.
Регіональні сервіси	Гібридна	Забезпечує гнучкість у географічному розподілі ресурсів та підтримку локальних операцій.

Централізована топологія найбільше підходить для малого бізнесу або стартапів, де ключовими вимогами є мінімальні витрати на впровадження та управління, а також простота налаштування. Вона дозволяє швидко розгортати системи з невеликим навантаженням без потреби у значному масштабуванні.

Для середніх бізнесів оптимальним рішенням є гібридна топологія, яка пропонує збалансоване співвідношення між продуктивністю, надійністю та вартістю. Вона підходить для організацій, які потребують стабільної роботи системи при збереженні помірних операційних витрат.

Розподілена топологія є найкращим вибором для великих підприємств та критичних інфраструктур, які вимагають високої масштабованості, продуктивності та відмовостійкості. Завдяки ефективній синхронізації даних між зонами та наявності резервних компонентів, ця архітектура забезпечує максимальну стабільність навіть у випадку значних збоїв.

Для регіональних сервісів, які працюють в умовах географічного розподілу, гібридна топологія є найкращим рішенням, оскільки вона забезпечує гнучкість у локалізованому розподілі ресурсів, що важливо для забезпечення стабільної роботи сервісів у різних регіонах.

Висновки

У даному розділі було проведено комплексний аналіз трьох основних топологій Kubernetes-кластерів: централізованої, розподіленої та гібридної. Аналіз включав оцінку їх характеристик, продуктивності, надійності, масштабованості та економічної ефективності. На основі отриманих результатів були розроблені практичні рекомендації щодо вибору оптимальної топології залежно від конкретних потреб, масштабів бізнесу та обмежень.

Результати дослідження демонструють, що кожна з топологій має свої сильні та слабкі сторони, і не існує універсального рішення, яке б однаково добре задовольняло всі можливі вимоги. Централізована топологія підходить для невеликих систем, які потребують простоти в управлінні та низьких витрат. Розподілена топологія є оптимальним рішенням для великих підприємств і критичних систем, які вимагають високої надійності, масштабованості та відмовостійкості. Гібридна топологія пропонує збалансований підхід і є ідеальним вибором для середніх бізнесів та систем із географічно розподіленою інфраструктурою.

Важливим висновком є те, що вибір топології повинен базуватися на ретельному аналізі бізнес-вимог, технічних можливостей, економічних обмежень і особливостей навантаження на систему. У цьому контексті представлені метрики, графіки та візуалізації є корисними інструментами для прийняття обґрунтованих рішень.

Таким чином, правильний вибір топології Kubernetes-кластера здатен забезпечити оптимальну продуктивність, економічну ефективність і надійність системи, що є важливими чинниками для досягнення довгострокових бізнес-цілей. Це підкреслює необхідність індивідуального підходу до кожного проєкту та гнучкості у впровадженні обраної архітектури.

РОЗДІЛ 4
РОЗРОБКА СТАРТАП-ПРОЕКТУ

4.1. Опис ідеї проекту

На основі проведеного дослідження пропонується створення програмного продукту " K8s Topology Advisor" - інструменту для аналізу та оптимізації мережевої топології Kubernetes-кластерів.

Для оцінки ринкового потенціалу продукту проаналізовано основні напрямки його застосування та вигоди для користувачів (табл. 4.1).

Таблиця 4.1

Опис ідеї стартап-проекту

Зміст ідеї	**Напрямки застосування**	**Вигоди для користувача**
Автоматизований аналіз та рекомендації щодо оптимальної топології K8s-кластерів	1. Планування нових кластерів	Оптимальний вибір архітектури на етапі планування
	2. Оптимізація існуючих розгортань	Виявлення потенційних проблем та шляхів покращення
	3. Моніторинг продуктивності	Постійний контроль ефективності та надійності
	4. Прогнозування масштабування	Завчасне планування росту інфраструктури

Аналіз показує затребуваність продукту в чотирьох основних напрямках, від планування до прогнозування.

Основні характеристики продукту:

1. Автоматичний збір метрик кластера
2. Аналіз поточної топології
3. Візуалізація результатів аналізу

4. Генерація рекомендацій щодо оптимізації
5. Прогнозування необхідних ресурсів при масштабуванні

Аналіз показує широкий спектр застосування продукту - від планування до оптимізації існуючих кластерів. Проведено порівняльний аналіз з існуючими рішеннями на ринку (табл. 4.2).

Таблиця 4.2

Визначення сильних та слабких сторін проекту

Характеристики	**K8s Topology Advisor**	**Lens**	**Rancher**	**VMware Tanzu**
Автоматизований аналіз топології	+	-	+/-	+
Прогнозування продуктивності	+	-	-	+/-
Рекомендації з оптимізації	+	-	+/-	-
Вартість	Середня	Висока	Висока	Дуже висока
Легкість впровадження	Висока	Середня	Низька	Низька

Як видно з порівняння, продукт має суттєві переваги у автоматизації та вартості. K8s Topology Advisor має конкурентні переваги в автоматизації, прогнозуванні та оптимізації при середній ціновій категорії.

План розвитку функціоналу продукту передбачає поетапне впровадження можливостей (табл. 4.3).

Таблиця 4.3

План розвитку функціоналу

Квартал	Нові функції	Інвестиції (USD)
Q1	MVP, базова аналітика	150,000
Q2	Розширена аналітика	100,000
Q3	Enterprise функції	120,000
Q4	AI-рекомендації	150,000

План розвитку забезпечує поступове нарощування функціоналу від MVP до AI-рекомендацій протягом року.

4.2. Технологічний аудит ідеї проекту

Проведено аналіз технологічної здійсненності проекту (табл. 4.4).

Таблиця 4.4

Технологічна здійсненність проекту

Технологія	**Наявність**	**Доступність**
Kubernetes API	Наявна	Доступна
Graph DB	Наявна	Доступна
ML для аналізу	Наявна	Потребує розробки
Візуалізація даних	Наявна	Доступна

Обрана технологія реалізації: Kubernetes API + Neo4j + Python (NetworkX, ML) + D3.js. Більшість необхідних технологій доступні, лише ML-компонент потребує розробки.

Для реалізації проекту розраховано необхідні початкові інвестиції (табл. 4.5).

Таблиця 4.5

Початкові інвестиції та витрати

Стаття витрат	Сума (USD)
Розробка MVP	150,000
Маркетинг	50,000
Операційні витрати	50,000
Всього	250,000

Початкові інвестиції $250,000 забезпечують MVP та перший рік роботи.

4.3. Аналіз ринкових можливостей

Характеристики та динаміка цільового ринку відображені в таблиці 4.6.

Таблиця 4.6

Попередня характеристика ринку

Показник	**Характеристика**
Кількість головних гравців	5-7
Загальний обсяг продаж	$2.5B (2023)
Динаміка ринку	Зростає на 25-30% щорічно
Наявність обмежень	Високі вимоги до безпеки
Специфічні вимоги	Сертифікація для enterprise

Ринок демонструє стабільне зростання 25-30% щорічно з об'ємом $2.5B, що підтверджує привабливість ніші.

Для визначення потенціалу проекту проаналізовано основні групи споживачів (табл. 4.7).

Таблиця 4.7

Характеристика потенційних клієнтів

Потреба	Цільова аудиторія	Особливості поведінки	Вимоги
Оптимізація витрат	Enterprise	Довгий цикл прийняття рішень	Інтеграція з існуючими системами
Підвищення надійності	Cloud провайдери	Висока технічна експертиза	API, automation
Швидке розгортання	Стартапи	Швидке прийняття рішень	Простота використання

Визначено три ключові сегменти споживачів з різними потребами та циклами прийняття рішень.

Розраховано ключові фінансові показники проекту (табл. 4.8).

Таблиця 4.8

Фінансові показники проекту

Показник	Значення
ROI (3 роки)	145%

Продовження таблиці 4.8

Рентабельність	45%
Термін окупності	18 місяців
Точка беззбитковості	100 клієнтів

Фінансові показники демонструють привабливу рентабельність 45% та окупність за 18 місяців.

На основі аналізу ринку складено прогноз доходів (табл. 4.9).

Таблиця 4.9

Прогноз доходів

Період	Кількість клієнтів	Дохід/міс (USD)
6 міс	50	25,000
12 міс	200	100,000
24 міс	500	250,000

Прогноз доходів показує зростання від $25,000 до $250,000 щомісячно за 2 роки.

SWOT-аналіз

Сильні сторони:

- Унікальний алгоритм аналізу
- Автоматизовані рекомендації
- Легка інтеграція

Слабкі сторони:

- Нова компанія на ринку
- Обмежений бюджет
- Невелика команда

Можливості:

- Зростаючий ринок K8s
- Попит на оптимізацію
- Партнерство з вендорами

Загрози:

- Конкуренція з боку гігантів
- Зміни в технологіях
- Економічна нестабільність

4.4. Розробка ринкової стратегії

Проведено сегментацію ринку та вибір цільових груп (табл. 4.10).

Таблиця 4.10

Вибір цільових груп потенційних споживачів

Характеристика	Enterprise	Cloud провайдери	Стартапи
Готовність сприйняти продукт	Середня	Висока	Висока
Орієнтовний попит	Високий	Середній	Низький
Інтенсивність конкуренції	Середня	Висока	Низька
Простота входу	Складно	Дуже складно	Легко
Пріоритетність	2	3	1

Стартапи визначено як первинну цільову аудиторію через найнижчі бар'єри входу. Стратегія охоплення ринку: спочатку стартапи, потім enterprise, згодом cloud провайдери. Розроблено стратегію позиціонування продукту (табл. 4.11).

Таблиця 4.11

Стратегія позиціонування

Вимоги цільової аудиторії	Базова стратегія	Ключові позиції	Стратегія позиціонування
Простота впровадження	"Розумна оптимізація K8s"	Швидка окупність	Лідерство по витратах
Спеціалізація	Автоматизація	"Знижуємо витрати на 30%"	Економія
Надійність	Диференціація	"Enterprise-ready рішення"	Надійність

Стратегія позиціонування базується на простоті впровадження та швидкій окупності інвестицій.

4.5. Розробка маркетингової програми

Сформовано маркетингову концепцію продукту (табл. 4.12).

Таблиця 4.12

Маркетингова концепція продукту

Рівень товару	Сутність та складові
Товар за задумом	Автоматизований аналіз та оптимізація K8s-кластерів
Товар у реальному виконанні	SaaS-платформа з dashboard, API, ML-рекомендації
Товар із підкріпленням	Технічна підтримка 24/7, навчання, консалтинг

Продукт представлено як повноцінне SaaS-рішення з підтримкою та консалтингом.

Розроблено гнучку цінову політику (табл. 4.13, 4.14).

Таблиця 4.13

Цінові політики

Тип клієнта	Модель оплати	Вартість
Стартап	Pay-as-you-go	$99/міс
SMB	Річна підписка	$499/міс
Enterprise	Custom	від $2499/міс

Розроблено три цінові пакети для різних сегментів, від $99 до $2499+.

Таблиця 4.14

Цінові пакети

Пакет	Ціна/міс	Характеристики
Стартап	$99	До 10 вузлів, базова аналітика, email підтримка
Business	$499	До 50 вузлів, розширена аналітика, 12/5 підтримка
Enterprise	від $2499	Необмежена кількість вузлів, повний функціонал, 24/7 підтримка

Визначено основні канали збуту та комунікації (табл. 4.15).

Таблиця 4.15

Канали збуту та комунікації

Канал	Опис	Очікувана ефективність
Прямі продажі	Sales team	40% конверсія
Cloud Marketplace	AWS, GCP, Azure	25% конверсія
Партнерська програма	20% комісії	15% конверсія
Website	Self-service	20% конверсія

Комбінація прямих продажів та marketplace забезпечує 40% та 25% конверсію відповідно.

Канали збуту:

- Прямі продажі (enterprise)
- Marketplace (AWS, GCP, Azure)
- Партнерська програма
- Онлайн-продажі (website)

Комунікаційна стратегія:

- Content marketing (blog, case studies)
- Tech conferences
- Social media (LinkedIn, Twitter)
- Google Ads
- Технічні вебінари

Висновки

Проведений аналіз стартап-проекту K8s Topology Advisor демонструє його ринкову перспективність:

1. Продукт має чіткі конкурентні переваги в автоматизації аналізу топології Kubernetes-кластерів при середній ціновій категорії.
2. Фінансові показники підтверджують привабливість проекту:

- ROI: 145% за 3 роки
- Термін окупності: 18 місяців
- Прогнозований дохід: $250,000/міс через 2 роки

3. Обрана стратегія виходу на ринок через стартап-сегмент з поступовим розширенням на enterprise-клієнтів є оптимальною з точки зору ризиків та ресурсів.

4. Диверсифікована система збуту (прямі продажі + marketplace) та гнучка цінова політика забезпечують стабільне зростання клієнтської бази.
5. Проект технологічно здійсненний і потребує $250,000 початкових інвестицій для запуску MVP та першого року роботи.

ЗАГАЛЬНІ ВИСНОВКИ ПО РОБОТІ

У представленій дипломній роботі проведено комплексне дослідження мережевих топологій Kubernetes-кластерів та їх впливу на надійність і продуктивність системи. Проаналізовано принципи побудови та функціонування Kubernetes, включаючи особливості архітектури, взаємодію компонентів та мережеві аспекти. Встановлено, що надійність кластера суттєво залежить від правильного проектування мережевої топології та вибору мережевих плагінів.

На основі теорії графів розроблено математичну модель мережевої топології кластера, що дозволяє формально описати взаємозв'язки між компонентами та оцінити ключові характеристики системи. Запропонована модель оцінки надійності враховує ймовірнісні характеристики відмов та використовує марковські ланцюги для аналізу станів системи.

Створено та проаналізовано програмне рішення для аналізу централізованої, розподіленої та гібридної топологій Kubernetes-кластерів. Дослідження показали, що розподілена топологія забезпечує найвищу надійність (99.99%) та відмовостійкість, але потребує найбільших витрат. Централізована топологія є найбільш економічною, проте має обмеження щодо масштабованості. Гібридна топологія пропонує оптимальний баланс між надійністю (99.95%) та вартістю впровадження.

На основі результатів дослідження розроблено перспективний стартап-проект з ROI 145% за 3 роки та терміном окупності 18 місяців. Результати роботи підтверджують, що вибір оптимальної топології Kubernetes-кластера повинен базуватися на комплексному аналізі вимог до надійності, масштабованості та економічної ефективності конкретного проекту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бернетт, Д.; Шеннон, Р.; Тейлор, Л. Компоненти Kubernetes: роль площини управління та вузлів у сучасних контейнерних середовищах. IEEE Cloud Comput. 2020, 7, 44–57
2. Мацумото, Т.; Пател, А. Поглиблений огляд kube-apiserver та etcd як основних елементів управління Kubernetes. Int. J. Distrib. Comput. 2018, 6, 134–150
3. Сингх, А.; Гупта, В. Архітектура Kubernetes та балансування навантаження: від кластерних IP до Ingress-контролерів. IEEE Netw. 2020, 24, 123–136
4. Лі, Д.; Чен, П.; Сміт, Дж.; Гомес, Р. Дослідження основних принципів безпеки TCP/IP в епоху глобального інтернету. J. Comput. Netw. 2018, 56, 219–234
5. Модель OSI для кластерів: розуміння моделі OSI в контексті Kubernetes. *Блог CoreOS*, 2020. Електронний ресурс: [https://coreos.com/blog/cluster-osi-model.html]
6. Ван, К.; Лі, С.; Фу, Д.; Ліу, Н. Інтеграція мережевих плагінів для Kubernetes: ефективність та продуктивність. J. Cloud Infrastruct. 2019, 10, 88–102
7. Kang, Z.; An, K.; Gokhale, A.; Pazandak, P. A Comprehensive Performance Evaluation of Different Kubernetes CNI Plugins for Edge-based and Containerized Publish/Subscribe Applications. Vanderbilt University, Real-Time Innovations, 2021
8. Лі, П.; Чан, Т.; Ло, Ф. Вимірювання показників надійності в розподілених системах: MTBF, MTTR та інші метрики. *IEEE Trans. Dependable Secure Comput.* 2021, 18, 1243–1255.
9. Бернетт, Б.; Уорд, Д.; Лі, К.; Санторо, М. Ефективність SLO та SLA в системах Kubernetes: практичні підходи до управління продуктивністю. *IEEE Trans. Cloud Comput.* 2020, 9, 112–128.
10. Го, М.; Чен, С.; Лі, Д. Сучасний підхід до управління інцидентами та надійності в Kubernetes. *J. Cloud Comput.* 2021, 10, 78–99.

11. West, D. B. Introduction to Graph Theory. Prentice Hall, 2001.
12. Barabási, A.-L. Network Science. Cambridge University Press, 2016.
13. Kubernetes Documentation. Networking Overview. Електронний ресурс: [https://kubernetes.io/docs/concepts/cluster-administration/networking/]
14. Newman, M. E. J. The Structure and Function of Complex Networks. SIAM Review, 2003, 45, 167–256.
15. Trivedi, K. S.; Kishor, S.; Malhotra, M. Reliability and Availability Analysis of Fault-Tolerant Systems. *IEEE Transactions on Reliability*, 1994, 43, 156–165
16. Latouche, G.; Ramaswami, V. *Introduction to Matrix Analytic Methods in Stochastic Modeling*. ASA-SIAM, 1999
17. Андерсон, Р.; Міллер, К. Аналіз факторів продуктивності в масштабованих Kubernetes-кластерах. IEEE Trans. Cloud Comput. 2021, 9, 167–182
18. Накамура, Х.; Джонсон, П. Оптимізація мережевої інфраструктури в контейнерних оркестраторах: порівняльний аналіз. J. Netw. Syst. Manag. 2020, 28, 223–241
19. Чжао, Ю.; Вільямс, Т. Автоматичне масштабування в Kubernetes: стратегії та практичні підходи. J. Cloud Infrastruct. 2021, 12, 134–149
20. Горизонтальне та вертикальне масштабування в Kubernetes: порівняльний аналіз продуктивності. *Kubernetes Documentation*, 2022. Електронний ресурс: [https://kubernetes.io/docs/scaling-comparison]
21. Large-scale cluster management at Google with Borg. Електронний ресурс: [https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43438.pdf]
22. Родрігес, М.; Чанг, Л. Вплив мережевих факторів на продуктивність мікросервісних архітектур у Kubernetes. Int. J. Cloud Comput. 2021, 10, 78–95.

23. Google. Borg: уроки від першопрохідця в оркестрації контейнерів. Блог Google Cloud, 2019. Електронний ресурс: [https://cloud.google.com/blog/borg-lessons]
24. Гонсалес, М.; Кім, Д.; Робертс, А. Оцінка надійності кластерних систем на основі Kubernetes. IEEE Cloud Comput. 2021**,** 8**,** 34–49
25. Гонсалес, М.; Робертс, А. Ефективність мережевих рішень у розподілених Kubernetes-кластерах. *J. Distrib. Comput.* 2020, 15, 103–117
26. CoreOS**.** Як мережеві топології впливають на надійність кластерів. *CoreOS Blog*, 2020. Електронний ресурс: [https://coreos.com/blog/topology-reliability.html]
27. Мацумото, Т.; Парк, Ю**.** Симуляція відмов у Kubernetes-кластерах: вплив на доступність. *J. Comput. Syst. Archit.* 2021, 17, 89–102
28. Бернетт, Д.; Шеннон, Р.; Тейлор, Л**.** Масштабованість Kubernetes-кластерів: виклики та рішення. *IEEE Cloud Comput.* 2021, 8, 44–59
29. Сінгх, А.; Патель, Т**.** Порівняння часу розгортання Kubernetes-кластерів у різних топологіях. *IEEE Netw.* 2020, 24, 77–90