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

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

«На правах рукопису»
УДК 004.9

До захисту допущено:
Завідувач кафедри
____Віталій РОМАНКЕВИЧ
«___»____________2023 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-науковою програмою

«Системне програмування і спеціалізовані комп‘ютерні системи»

зі спеціальності 123 «Комп‘ютерна інженерія»

на тему: «Засоби керування обчислювальними ресурсами у Kubernetes кластері»

Виконав:
студент ІІ курсу, групи КВ-11мн
Гнідий Павло Олександрович__________
Науковий керівник:
професор кафедри СПСКС, д.т.н., професор
Яценко Віталій Олексійович __________

Рецензент: __________

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

Київ – 2023 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

Рівень вищої освіти – другий (магістерський)

Спеціальність – 123 «Комп‘ютерна інженерія»

Освітньо-професійна програма «Системне програмування і спеціалізовані комп‘ютерні системи»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____Віталій РОМАНКЕВИЧ

«___»_____________2023 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Гнідому Павлу Олександровичу

1. Тема дисертації «Засоби керування обчислювальними ресурсами у Kubernetes кластері», науковий керівник дисертації професор кафедри СПСКС, д.т.н., професор Яценко Віталій Олексійович, затверджені наказом по університету МД № 1359-с від 30.03.2023 р.

2. Термін подання студентом дисертації ___________________________________

3. Об’єкт дослідження:
   - засоби керування обчислювальними ресурсами у Kubernetes кластері

4. Вихідні дані:
   - спосіб керування обчислювальними ресурсами у Kubernetes кластері, програмне рішення для розгортання середовища на базі описаного способу

5. Перелік завдань, які потрібно розробити:

   - аналіз існуючих способів управління обчислювальними ресурсами
   - опис способу управління обчислювальними ресурсами у Kubernetes кластері та розробка програмного забезпечення для розгортання середовища на базі описаного способу

6. Орієнтовний перелік графічного (ілюстративного) матеріалу - презентація

7. Орієнтовний перелік публікацій:

- IV Міжнародна студентська наукова конференція «Теоретичне та практичне застосування результатів сучасної науки»
- III Міжнародна наукова конференція «Розвиток наукової думки постіндустріального суспільства: сучасний дискурс».

8. Дата видачі завдання ___________

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Затвердження теми	10.01.2023	
2	Збір та дослідження літератури	20.01.2023	
3	Аналіз існуючих рішень	05.02.2023	
4	Зміст та вступ	25.02.2023	
5	Розробка програмної моделі рішення	10.03.2023	
6	Реферат	21.03.2023	
7	Перший розділ	27.03.2023	
8	Другий розділ	01.04.2023	
9	Третій розділ	22.04.2023	
10	Четвертий розділ	03.05.2023	
11	Попередній захист магістерської дисертації	04.05.2023	

Студент Павло ГНІДИЙ

Науковий керівник Віталій ЯЦЕНКО

Реферат

Актуальність теми. Хмарні обчислення — це доставка обчислювальних ресурсів, включаючи сховище, обчислювальну потужність, бази даних, мережу, аналітику, штучний інтелект і програмне забезпечення — через Інтернет (хмару). Завдяки аутсорсингу цих ресурсів компанії можуть отримати доступ до обчислювальних ресурсів, які їм потрібні, коли вони їм потрібні, без необхідності купувати та підтримувати фізичну локальну ІТ-інфраструктуру. Це забезпечує гнучкі ресурси, швидші інновації та економію на масштабі. Для багатьох компаній хмарна міграція безпосередньо пов'язана з модернізацією даних та ІТ.

Розробники створюють програми в локальній системі, потім доставляють код у середовище розробки команди. Це часто може викликати складність через великі витрати на системне адміністрування, пов'язані з наданням кожному розробнику кластеру машин. Тестування також є важливою частиною процесу розробки, яка потребує значної уваги. Розробники, які працюють над програмами розподілених мікросервісів, залишилися без ефективного способу створення, запуску тестів і усунення несправностей у середовищі, схожому на робоче. Локальні середовища розробки є важливими під час створення та розробки додатків.

Об'єктом дослідження є технологія управління обчислювальними ресурсами в Kubernetes кластері.

Предметом дослідження є засоби керування обчислювальними ресурсами у Kubernetes кластері.

Мета роботи: запропонувати стандартизований і адаптований метод керування обчислювальними ресурсами у Kubernetes кластері та розгортання програм у великих і складних системах, який підтримує широкий спектр мов і фреймворків, простий у використанні, гнучкий.

Наукова новизна полягає в наступному: запропоновано засіб керування ресурсами у Kubernetes кластері, що полегшує розгортання середовищ тестування та попереднього перегляду результатів інтеграційних тестів, ручної перевірки якості та розробки.

Практична цінність отриманих в роботі результатів полягає в тому, що запропонований метод дозволяє легко та швидко розгорнути локальне середовище схоже на робоче, що дозволяє протестувати і налагодити функціонал самостійно та миттєво отримати результати інтеграційних тестів, що значно підвищує ефективність роботи розробника. Таким чином сам розробник не повинен чекати результати роботи CI/CD, що на великих проектах може займати години.

Апробація роботи. Основні положення були представлені та обговорювались на IV Міжнародній студентській науковій конференції «Теоретичне та практичне застосування результатів сучасної науки» та III Міжнародній науковій конференції «Розвиток наукової думки постіндустріального суспільства: сучасний дискурс».

Структура та обсяг роботи. Магістерську дисертація складається з вступу, чотирьох розділів та висновків.

У вступі подано загальну характеристику роботи, зроблено оцінку сучасного стану проблеми, обґрунтовано актуальність напрямку досліджень, сформульовано мету і задачі досліджень, показано наукову новизну отриманих результатів і практичну цінність роботи, наведено відомості про апробацію результатів і їхнє впровадження.

У першому розділі розглянуто існуючі програмі рішення для розгортання багатокомпонентних систем в локальній системі та проаналізовано їх переваги та недоліки.

У другому розділі описано принципи роботи IaaC інструментів для створення середовища багатокомпонентних систем.

У третьому розділі та четвертому розділах наведено структуру та опис роботи програмного забезпечення, а також проведено тестування та проаналізовано результати виконаного дослідження.

У висновках представлені результати проведеної роботи. Робота представлена на 80 аркушах, містить посилання на список використаних літературних джерел.

Ключові слова: Cloud Computing, Docker, Docker Compose, Kubernetes, Terraform, end-to-end testing.

Abstract

Actuality of topic. Cloud computing is the delivery of computing resources, including storage, computing power, databases, networking, analytics, artificial intelligence, and software, over the Internet (the cloud). By outsourcing these resources, businesses can access the computing resources without having to purchase and maintain a physical IT infrastructure on-premises. This allows for flexible resources, faster innovation, and economies of scale. For many companies, migrating to the cloud is directly related to modernizing data and IT.

Developers often build applications on a local system and then, when they reach a working stage, deliver the code to the team's development environment. Most developers want to be able to develop on a daily basis with an infrastructure that is as close to production as possible. This goal can often be difficult due to the large system administration overhead associated with providing each developer with a group of machines. Testing is also an important part of the development process that requires a lot of attention. Developers working on distributed microservices applications are left without an efficient way to build, test, and troubleshoot in a production-like environment. Local development environments are important when creating and developing applications. It is in such environments that we develop code securely without concern for the performance of the underlying software available to users.

The object of the study is the deployment technology of complex multi-component systems.

The subject of the research is the software to define and execute the multi-component program environment on the local system and in the cloud.

The purpose of the work: to develop a standardized and adapted method for managing and deploying applications, especially in large and complex environments, that supports a wide range of languages and frameworks, is easy to use and flexible.

The scientific novelty is the following: a decentralized tool is proposed that facilitates the deployment of ephemeral test environments and the preview of the

results of integration tests, manual QA and development. Thanks to this, developers get fast feedback exactly when they need it. Have the ability to integrate with various distributed IaaC tools, such as Kubernetes.

The practical value of the results obtained in the work is that the proposed method allows you to easily and quickly deploy a production-like local environment, allowing you to test and debug the functionality yourself and instantly get the results of the integration tests, which significantly increases the efficiency of the developer's work. In this way, the developer himself does not have to wait for the results of CI/CD work, which can take hours on large projects.

Approbation of work. The main provisions were presented and discussed at the XIV Scientific Conference of Master and Graduate Students "Applied Mathematics and Computing" PMK-2022 (Kiev, November 2022) and the International Scientific and Technical Internet Conference at the National University of Food Technologies .

Structure and scope of work. The master's thesis consists of an introduction, four chapters and conclusions.

The introduction provides a general description of the work, evaluates the current state of the problem, substantiates the relevance of the direction of the research, formulates the purpose and tasks of the research, shows the scientific novelty of the results obtained and the practical value of the work provides information on the approval of the results and their implementation.

In the first section, existing software solutions for implementing multi-component systems on the local system are considered and their advantages and disadvantages are discussed.

The second chapter describes the principles of operation of IaaC tools to create a multi-component systems environment.

The third chapter and the fourth chapter give the structure and description of the software, as well as testing and analyzing the research results.

The results of the work are presented in the conclusions. The work is presented in 86 pages, contains links to the list of literary sources used.

Keywords: Cloud computing, Docker, Docker Compose, Kubernetes, Terraform, end-to-end testin.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ.............. 4

ВСТУП.. 6

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБҐРУНТУВАННЯ ТЕМИ МАГІСТЕРСЬКОЇ ДИСЕРТАЦІЇ.. 7

1.1. Визначення хмарної інфраструктури.. 7

1.2. Способи опису хмарної інфраструктури.. 9

1.2.1 Ansible.. 10

1.2.2 Terraform.. 11

1.2.3 CloudFormation 13

1.3. Інструменти управління контейнерезованими програмами......... 14

1.3.1. Kubernetes... 16

1.3.2. AWS Fargate.. 18

1.3.3. Docker Swarm... 19

1.3.4. Docker Compose... 20

1.3.5. Nomad 21

1.4. Постановка задачі магістерської дисертації 22

ВИСНОВКИ ДО РОЗДІЛУ 1.. 24

2. ЗАСОБИ ОПИСУ ТА КЕРУВАННЯ ОБЧИСЛЮВАЛЬНИМИ РЕСУРСАМИ В ХМАРІ 25

2.1. Формат опису хмарних ресурсів.. 27

2.1.1 YAML 27

2.1.2 JSON... 29

2.1.3 HCL 31

2.1.4 Порівняння форматів опису хмарних ресурсів.............................. 32

2.2. Kubernetes – інструмент управляння контейнерезованими робочими навантаженнями ... 33
2.3. Docker – інструмент контейнерезації ... 38
2.4. Способи взаємодії з комп'ютерною системою ... 40
2.5. Способи візуалізацій залежностей між компонентами системи .. 42
ВИСНОВКИ ДО РОЗДІЛУ 2 ... 47
3. СТРУКТУРА ТА ОПИС РОБОТИ МОДУЛІВ ПРОГРАМНОГО ЗАБЕСПЕЧЕННЯ ... 49
3.1. Мова програмування для імплементації обраного програмного забезпечення ... 49
3.2. Принцип роботи та архітектура реалізованої платформи ... 56
3.3. Опис архітектури реалізованої платформи ... 56
3.3.1 CLI ... 58
3.3.2 Server ... 60
3.3.3 Agent ... 62
ВИСНОВКИ ДО РОЗДІЛУ 3 ... 65
4. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕСПЕЧЕННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ ... 66
4.1. Тестування програмного забезпечення ... 66
4.2. Аналіз та порівняння результатів ... 74
ВИСНОВКИ ДО РОЗДІЛУ 4 ... 76
ВИСНОВКИ ... 77
СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ ... 79

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

AKS (англ. Azure Kubernetes Service) - сервіс кластерів Azure Kubernetes.

API (англ. Application Programming Interface) – це набір правил, протоколів та інструментів, які використовуються для розробки програмного забезпечення та взаємодії між різними програмами.

AWS (англ. Amazon Web Services) - веб-сервіси Amazon.

CI/CD (англ. Continuous Integration/Continuous Delivery) – це методологія розробки програмного забезпечення, що передбачає автоматичну інтеграцію, тестування, збірку та розгортання коду для забезпечення швидкої та безперервної доставки якісного програмного забезпечення виробничому середовищу.

DFD (англ. Data Flow Diagram) – діаграма потоку даних, яка використовується для моделювання та аналізу потоків даних у системі.

DNS (англ. Domain Name System) - система доменних імен.

ECS (англ. Elastic Container Service) - сервіс контейнерів.

ER (англ. Entity-Relationship) - сутність-зв'язок.

GKE (англ. Google Kubernetes Engine) - двигун Kubernetes Google.

GUI (англ. Graphical User Interface) - графічний інтерфейс користувача.

HCL (англ. HashiCorp Configuration Language) - мова конфігурації HashiCorp.

IaC (англ. Infrastructure as Code) - інфраструктура як код.

IT (англ. Information Technology) - інформаційні технології.

NPM (англ. Node Package Manager) - менеджер пакетів Node.

RDS (англ. Relational Database Service) - реляційний сервіс баз даних.

REST (англ. Representational State Transfer) - передача стану представлення.

S3 (англ. Simple Storage Service) - простий сервіс зберігання.

SOAP (англ. Simple Object Access Protocol) - це протокол обміну повідомленнями, який використовується для взаємодії між різними компонентами програмного забезпечення через мережу Інтернет.

SSH (англ. Secure Shell) - це криптографічний протокол мережевого зв'язку, який використовується для забезпечення захищеного доступу до мережевих пристроїв, передачі даних та виконання команд на віддалених комп'ютерах.

СУБД – система управління базами даних

UML (англ. Unified Modeling Language) - це стандартна мова моделювання програмного забезпечення.

XML (англ. Extensible Markup Language) - це розширювана мова розмітки, що використовується для зберігання та обміну даними у вигляді текстових файлів.

YAML (англ. YAML Ain't Markup Language) - це мова розмітки, яка використовується для представлення даних у вигляді структурованого тексту.

ВСТУП

Концепція «хмарних обчислень» зародилася в 1960 році, коли Джон Маккарті висловив припущення, що колись комп'ютерні обчислення будуть проводитися за допомогою «загальнонародних утиліт». Ідеологія хмарних обчислень набула популярності в 2007 році завдяки швидкому розвитку каналів зв'язку та потреби як бізнесу, так і приватних користувачів, що зростає в геометричній прогресії, в горизонтальному масштабуванні своїх інформаційних систем. Це призвело до виникнення такого поняття, як Infrastructure-as-a-Service (IaaS), що забезпечує доступ до основних ресурсів, таких як фізичні машини, віртуальні машини, віртуальне сховище тощо.

Infrastructure-as-Code (IaC) — це процес керування та надання комп’ютерних центрів обробки даних за допомогою тестових файлів визначення, а не фізичної конфігурації апаратного забезпечення чи інтерактивних інструментів конфігурації. IT-інфраструктура, якою керує цей процес, включає як фізичне обладнання, такі як сервери, віртуальні машини, так і відповідні ресурси конфігурації. Визначення можуть бути в системі контролю версій. Код у файлах визначення може використовувати або сценарії, або декларативні визначення, а не підтримувати код за допомогою ручних процесів.

Метою роботи є розробка інструменту, який автоматизує розгортання середовищ, яке складається з багатьох компонентів як в локальній системі там і в хмарі, використовуючи підхід IaC та має можливість інтеграції до різних розповсюджених інструментів як Kubernetes.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБҐРУНТУВАННЯ ТЕМИ МАГІСТЕРСЬКОЇ ДИСЕРТАЦІЇ

1.1.Визначення хмарної інфраструктури

Хмарні обчислення — це модель забезпечення зручного мережевого доступу на вимогу до загального пулу конфігуруємих обчислювальних ресурсів (наприклад, мереж передачі даних, серверів, пристроїв зберігання даних, додатків і сервісів - як разом, так і окремо) , які можуть бути оперативно надані і звільнені з мінімальними експлуатаційними витратами або зверненнями до провайдера [1].

З хмарними обчисленнями клієнти можуть отримати доступ до п'яти важливих аспектів:

1) Управління даними
2) Зберігання даних
3) Пристрої
4) Мережа
5) Інтернет та інші онлайн-сервіси

Постачальники хмарних сервісів розвивають свій бізнес та продовжують удосконалювати пропозиції, дозволяючи вирішувати з їх допомогою ще складніші завдання. Нинішня хвиля інновацій, що впроваджуються провідними постачальниками хмарних обчислень, передбачає подальше збільшення кількості сервісів, причому постачальник управляє їх масштабами та безпекою та знижує вартість, складність та ризики для клієнта. Один із способів досягнення цього – реорганізація існуючих технологічних платформ таким чином, щоб вони не просто вирішували конкретні технічні проблеми, а й робили це з урахуванням переваг хмарних обчислень.

Наприклад, в AWS є сервіс керованих СУБД Amazon Aurora, побудований на повністю розподіленому та самовідновлюваному сховищі, призначеному для забезпечення безпеки даних та їх розподілу за кількома зонами доступності. Цей сервіс підвищує корисність пропозицій керованих сервісів, специфічних для СУБД, дозволяючи також створювати масив сховищ, який збільшується на вимогу і налаштовується для хмари, щоб забезпечити продуктивність, вп'ятеро більшу, ніж дають аналогічні двигуни СУБД.

Не всі сучасні віддалено керовані послуги є переробкою існуючих технологій. З використанням безсерверних обчислень постачальники хмарних технологій прибирають недиференційовану рутинну роботу не тільки з процесу експлуатації, а й із розробки. Оскільки хмара може запропонувати практично безмежні ресурси, поділ великих додатків на окремі функції є наступною великою хвилею проектування розподілених систем та веде безпосередньо до формування архітектури cloud native.

Мікросервіси та хмарні обчислення добре поєднуються один з одним, і найчастіше мікросервіси вважаються найбільш розвиненим типом архітектури (Рисунок 1.1).

На даний момент причина такої гарної комбінації полягає в тому, що постачальники хмарних послуг часто розробляють їх у вигляді окремих будівельних блоків, які можна використовувати різними способами для досягнення бізнес-результату. Підхід, заснований на стандартних блоках, дає командам розробників додатків можливість комбінувати та зіставляти сервіси вирішення своїх проблем, замість того щоб застосовувати конкретний тип сховища даних або мову програмування. Це призвело до збільшення кількості інновацій та шаблонів проектування, що використовують переваги хмари, таких як безсерверні обчислювальні сервіси, що приховують управління ресурсами від команд розробників, що дозволяє їм зосередитись на бізнес-логіці.

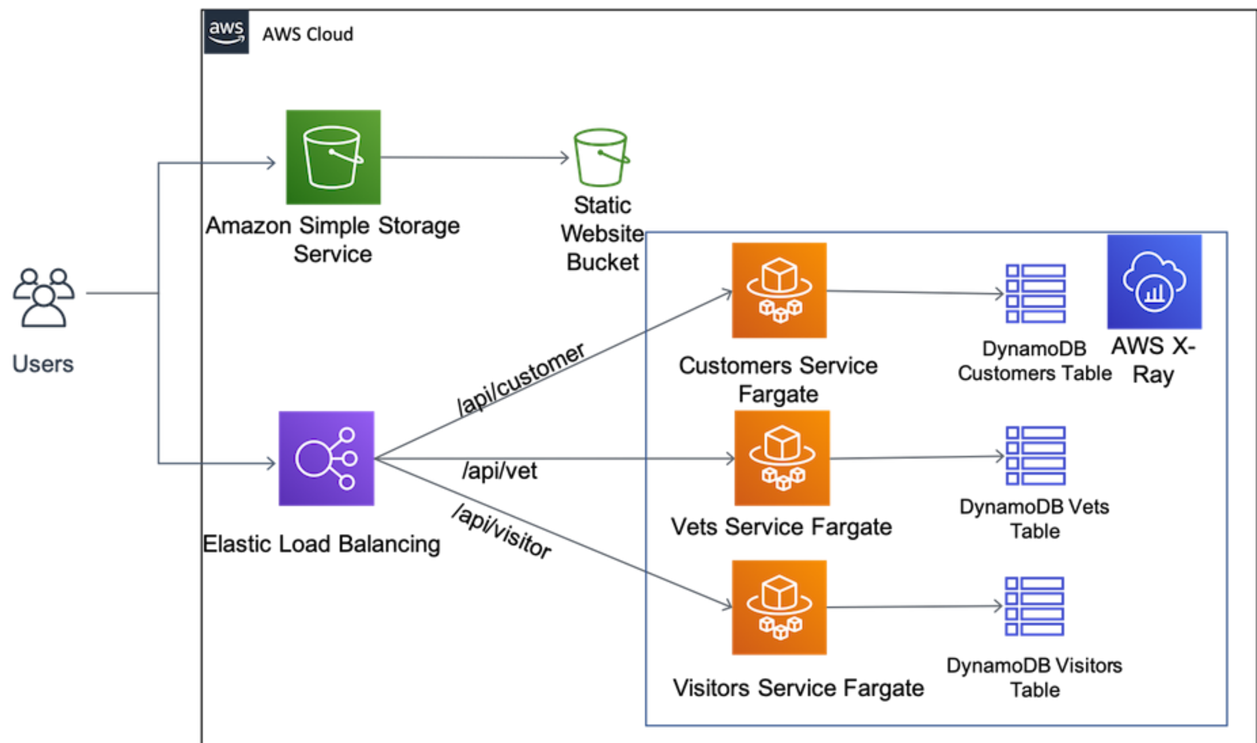

Рисунок 1.1 – Мікросервісна архітектура на базі AWS Cloud

1.2. Способи опису хмарної інфраструктури

Інфраструктура як код – це процес керування ІТ-інфраструктурою, який застосовує найкращі практики розробки програмного забезпечення DevOps для керування ресурсами хмарної інфраструктури. Застосовними інфраструктурними ресурсами є віртуальні машини, мережі, балансувальники навантаження, бази даних та інші мережеві програми.

IaC — це спосіб постачання та керування обчислювальними та мережевими ресурсами методом їх опису у вигляді програмного коду, на відміну від налаштовування необхідного обладнання власноруч чи з допомогою інтерактивних інструментів. Інфраструктура, керована таким чином, охоплює як фізичні сервери, так і віртуальні машини, а також пов'язані з ними ресурси. У підході використовуються як виконувані скрипти, так і декларативні визначення,

шаблони, які можуть перебувати в системі контролю версій. Термін найчастіше використовується для позначення декларативного опису IT інфраструктури [2].

Інфраструктура як код стала можливою завдяки поширенню платформ розміщення хмарної інфраструктури, зокрема платформ IaaS. IaaS дозволяє надавати за вимогою та вимагати хмарні ресурси через віддалені API, які встановлюють шаблон для властивостей, закріплених у файлах конфігурації інфраструктури. Функції автоматизації IaC можуть приймати файли конфігурації та запускати їх із віддаленими API IaaS.

Після того, як команда закріпила конфігурацію інфраструктури для контролю версій, вони можуть застосувати методи CI/CD до змін інфраструктури. Оновлення інфраструктури можна виконувати за робочим процесом DevOps. Якщо член команди відредагував один із текстових файлів конфігурації, для аудиту та перевірки правильності змін можна використовувати запити на отримання та робочі процеси перегляду коду. Крім того, інфраструктура як кодова система з підтримкою DevOps використовуватиме автоматичне розгортання інфраструктури та відкат.

1.2.1 Ansible

Ansible — це програмне забезпечення, що надає засоби для управління конфігурацією, оркестровки, централізованої установки застосунків і паралельного виконання типових завдань на групі систем. Початковий код Ansible написаний мовою Python [3], це простий механізм IT-автоматизації з відкритим вихідним кодом, який автоматизує хмарне забезпечення, внутрішньосервісну оркестровку, керування конфігурацією, розгортання додатків і багато інших потреб IT у повсякденному житті програмного інженера.

Він розроблений для багаторівневого розгортання. Таким чином, він точно узгоджується з базовою структурою хмари. Ansible моделює вашу IT-інфраструктуру, описуючи, як усі ваші системи взаємопов'язані, а не просто керує однією системою за раз.

Автоматизація спрощує складні завдання, не лише роблячи роботу розробників більш керованою, але й дозволяючи їм зосередити увагу на інших завданнях, які додають цінності організації. Іншими словами, це звільняє час і підвищує ефективність.

1.2.2 Terraform

Terraform — це програмне забезпечення з відкритим вихідним кодом, створене HashiCorp. Користувачі визначають і надають інфраструктуру центру обробки даних за допомогою декларативної мови конфігурації, відомої як HashiCorp Configuration Language (HCL) [4] (Рисунок 1.2).

Terraform — це інструмент для безпечного й ефективного створення, зміни та керування версіями інфраструктури. Terraform може керувати популярними постачальниками послуг, а також індивідуальними власними рішеннями. Він використовується для визначення та надання повної інфраструктури за допомогою легкої для вивчення декларативної мови під назвою HashiCorp Configuration Language (HCL).

Terraform може допомогти з кількома хмарами, маючи один робочий процес для всіх хмар. Інфраструктура, якою керує Terraform, може бути розміщена в загальнодоступних хмарах, таких як Amazon Web Services, Microsoft Azure та Google Cloud Platform. Багатохмарне розгортання: багатохмарне розгортання може бути дуже складним, оскільки багато існуючих інструментів

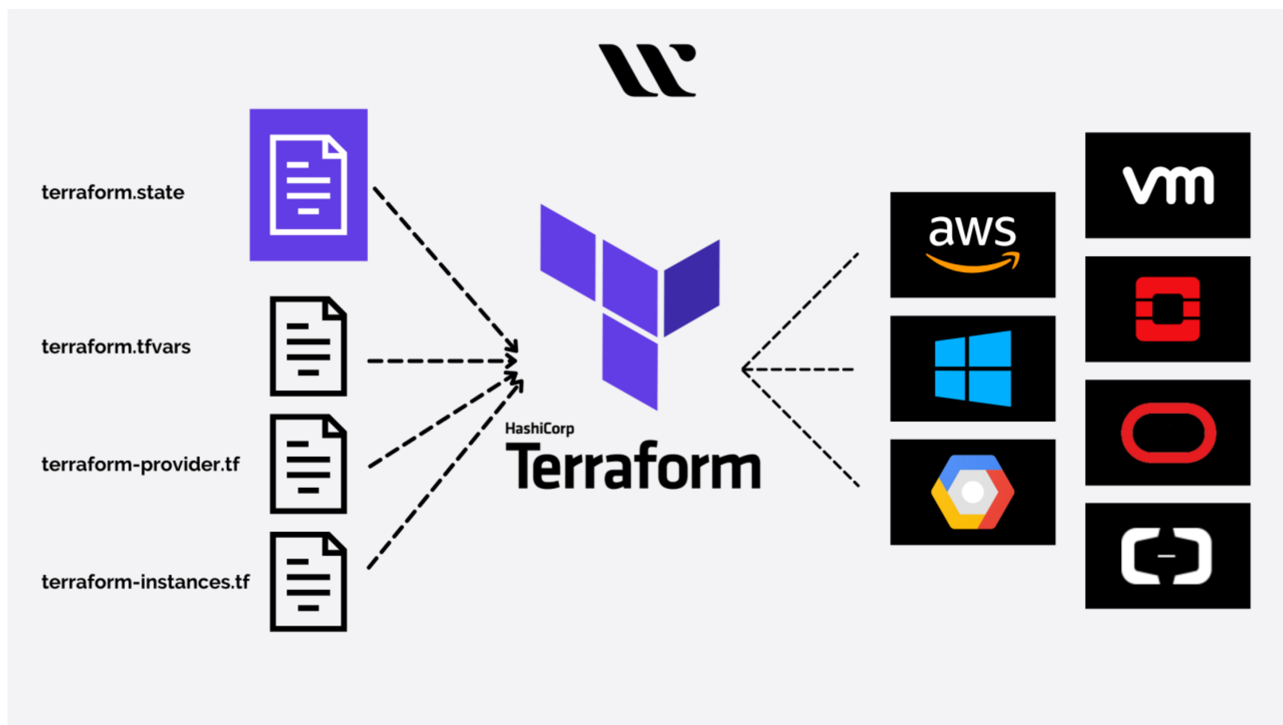

Рисунок 1.2 – Схема основних компонентів Terraform

для керування інфраструктурою є специфічними для хмари. Terraform не залежить від хмари та дозволяє використовувати одну конфігурацію для керування декількома провайдерами та навіть для обробки міжхмарних залежностей. Це спрощує керування та оркестровку, допомагаючи операторам створювати великомасштабні мультихмарні інфраструктури.

Багаторівневі програми — кожен рівень можна описати як набір ресурсів, і залежності між кожним рівнем обробляються автоматично; Terraform забезпечить доступність рівня бази даних до запуску веб-серверів і що балансувальники навантаження знають про веб-вузли. Потім кожен рівень можна легко масштабувати за допомогою Terraform, змінюючи одне значення конфігурації підрахунку. Оскільки створення та надання ресурсу кодифіковано та автоматизовано, еластичне масштабування з навантаженням стає тривіальним.

1.2.3 CloudFormation

Це послуга, надана AWS, яка допомагає вам створювати ресурси та керувати ними, щоб ви могли витрачати менше часу на керування цими ресурсами та більше часу, зосереджуючись на програмах, які працюють в AWS. Вам просто потрібно створити шаблон, який описує всі ресурси, які вам потрібні, тоді AWS Cloudformation подбає про керування та надання всіх ресурсів. AWS надає конструктор Cloudformation для розробки шаблону, у який можна розмістити всі ресурси. Ви також можете визначити залежності всіх необхідних ресурсів. Ви також можете повторно використовувати свої шаблони для копіювання своєї інфраструктури в багатьох середовищах і регіонах.

Переваги AWS Cloudformation:

1. Автоматизація: AWS CloudFormation допомагає автоматизувати процес створення, налаштування та керування ресурсами AWS. Це дозволяє швидко, надійно та неодноразово розгортати інфраструктуру.

2. Послідовність і стандартизація: за допомогою AWS CloudFormation можна створювати стандартні шаблони стеків інфраструктури, які можна використовувати для створення ідентичних копій тієї самої інфраструктури. Це забезпечує узгодженість у розгортанні інфраструктури та полегшує її обслуговування.

3. Економія: AWS CloudFormation допомагає зменшити витрати, дозволяючи клієнтам використовувати існуючі шаблони інфраструктури та повторно використовувати їх у кількох середовищах. Це зменшує витрати на проектування та розгортання нової інфраструктури.

4. Безпека: AWS CloudFormation допомагає переконатися, що всі ресурси AWS налаштовані безпечним способом за допомогою політик і правил безпеки. Це допомагає захистити інфраструктуру від потенційних загроз безпеці.

5. Масштабованість: AWS CloudFormation дозволяє швидко та легко масштабувати ресурси за вимогою. Це означає, що клієнти можуть швидко та легко додавати ресурси для задоволення своїх мінливих потреб.

1.3.Інструменти управління контейнерезованими програмами

Оркестровка контейнерів автоматизує розгортання, керування, масштабування та роботу в мережі. Підприємства, яким необхідно розгортати та керувати сотнями або тисячами контейнерів і хостів Linux®, можуть отримати вигоду від оркестровки контейнерів.

Оркестровку контейнерів можна використовувати в будь-якому середовищі, де ви використовуєте контейнери. Це може допомогти вам розгорнути ту саму програму в різних середовищах без необхідності її перепроектування. А мікросервіси в контейнерах спрощують організацію служб, зокрема зберігання, мережі та безпеки.

Контейнери надають вашим додаткам на основі мікросервісів ідеальну одиницю розгортання додатків і автономне середовище виконання. Вони дають змогу запускати кілька частин програми незалежно в мікросервісах на одному апаратному забезпеченні з набагато більшим контролем над окремими частинами та життєвими циклами.

Управління життєвим циклом контейнерів за допомогою оркестровки також підтримує команди DevOps, які інтегрують це в робочі процеси CI/CD. Разом з інтерфейсами програмування додатків (API) і командами DevOps контейнерні мікросервіси є основою для хмарних додатків.

Використовуйте оркестрування контейнерів для автоматизації та керування такими завданнями, як:

- надання та розгортання;
- конфігурація та планування;
- розподіл ресурсів;
- наявність контейнера;
- масштабування або видалення контейнерів на основі балансування робочих навантажень у вашій інфраструктурі;
- балансування навантаження та маршрутизація трафіку;
- контроль стану контейнера;
- налаштування додатків на основі контейнера, в якому вони будуть запускатися;
- захист взаємодії між контейнерами.

Коли ви використовуєте інструмент оркестровки контейнера, наприклад Kubernetes, ви описуєте конфігурацію програми за допомогою файлу YAML або JSON. Файл конфігурації повідомляє інструменту керування конфігурацією, де знайти зображення контейнерів, як встановити мережу та де зберігати журнали.

Під час розгортання нового контейнера інструмент керування контейнером автоматично планує розгортання в кластері та знаходить правильний хост, враховуючи будь-які визначені вимоги чи обмеження. Потім інструмент оркестровки керує життєвим циклом контейнера на основі специфікацій, визначених у файлі створення.

Ви можете використовувати шаблони Kubernetes для керування конфігурацією, життєвим циклом і масштабом контейнерних програм і служб. Ці шаблони, що повторюються, є інструментами, які потрібні розробнику Kubernetes для створення повних систем.

Оркестровку контейнерів можна використовувати в будь-якому середовищі, яке запускає контейнери, включно з локальними серверами та публічними або приватними хмарними середовищами.

1.3.1. Kubernetes

Як зазначалося раніше (але варто повторити), Kubernetes - це платформа з відкритим вихідним кодом для управління контейнеризованими робочими навантаженнями та супутніми службами. Її основні характеристики - кросплатформенність, розширюваність, успішне використання декларативної конфігурації та автоматизації. Вона має гігантську, швидкопрогресуючу екосистему [5]. Він забезпечує середовище виконання контейнера, оркестровку контейнерів, оркестровку інфраструктури, орієнтовану на контейнери, механізми самовідновлення, виявлення служб і балансування навантаження. Він використовується для розгортання, масштабування, керування та композиції контейнерів додатків у кластерах хостів.

Але Kubernetes — це більше, ніж просто оркестровка контейнерів. Її можна розглядати як операційну систему для хмарних додатків у тому сенсі, що це платформа, на якій працюють додатки, так само як додатки для настільних комп'ютерів працюють у MacOS, Windows або Linux.

Він має на меті зменшити навантаження на організацію основної обчислювальної, мережевої інфраструктури та інфраструктури зберігання даних і дати можливість операторам додатків і розробникам повністю зосередитися на робочих процесах, орієнтованих на контейнери, для роботи в режимі самообслуговування. Це дозволяє розробникам створювати індивідуальні робочі процеси та автоматизувати вищий рівень для розгортання програм, що складаються з кількох контейнерів, і керування ними.

Незважаючи на те, що Kubernetes запускає всі основні категорії робочих навантажень, як-от моноліти, додатки без стану або збереження стану, мікросервіси, служби, пакетні завдання та все між ними, він зазвичай використовується для категорії робочих навантажень мікросервісів.

У перші роки проекту він здебільшого запускав додатки без стану, але в міру того, як платформа набула популярності, було розроблено все більше інтеграцій сховища для підтримки програм із збереженням стану.

Kubernetes — це дуже гнучка та розширювана платформа. Це дозволяє вам використовувати його функціональні можливості за бажанням або використовувати власне рішення замість вбудованих функцій. З іншого боку, ви також можете інтегрувати Kubernetes у своє середовище та додати додаткові можливості.

На високому рівні середовище Kubernetes складається з площини керування (master), розподіленої системи зберігання для підтримки узгодженого стану кластера (etcd) і кількох вузлів кластера (Kubelets).

Площина керування — це система, яка підтримує записи всіх об’єктів Kubernetes. Він постійно керує станами об'єктів, реагуючи на зміни в кластері; він також працює, щоб фактичний стан системних об'єктів відповідав бажаному стану. Як показано на зображенні вище, площина керування складається з трьох основних компонентів: kube-apiserver, kube-controller-manager і kube-scheduler. Усі вони можуть працювати на одному головному вузлі або можуть бути відтворені на кількох головних вузлах для високої доступності.

Сервер API надає API для підтримки оркестрації життєвого циклу (масштабування, оновлення тощо) для різних типів програм. Він також діє як шлюз до кластера, тому сервер API повинен бути доступний клієнтам поза кластером. Клієнти автентифікуються через сервер API, а також використовують його як проксі/тунель для вузлів і модулів (і служб).

Більшість ресурсів містять метадані, такі як мітки та анотації, бажаний стан (специфікація) і спостережуваний стан (поточний статус). Контролери працюють, щоб привести фактичний стан до бажаного.

Існують різні контролери для управління станом вузлів, реплікації (автомасштабування), кінцевих точок (служб і модулів), облікових записів служб

і токенів (просторів імен). Менеджер контролера — це демон, який запускає основні цикли керування, спостерігає за станом кластера та змінює стан приводу до бажаного стану. Cloud Controller Manager інтегрується в кожну публічну хмару для оптимальної підтримки зон доступності, екземплярів віртуальних машин, служб зберігання та мережевих служб для DNS, маршрутизації та балансування навантаження.

Планувальник відповідає за планування контейнерів між вузлами в кластері; він враховує різні обмеження, такі як обмеження ресурсів або гарантії, а також специфікації спорідненості та антиспорідненості.

Вузли кластера — це машини, які запускають контейнери та керуються головними вузлами. Kubelet є основним і найважливішим контролером у Kubernetes. Він відповідає за керування рівнем виконання контейнера, як правило, Docker.

1.3.2. AWS Fargate

AWS Fargate — це технологія, яку можна використовувати з Amazon ECS і Amazon EKS [6]. Ви можете використовувати його для запуску контейнерів без керування серверами чи кластерами примірників Amazon EC2. Fargate має гнучку обчислювальну модель, яка не вимагає вибору типу екземпляра або безпосереднього налаштування кластера. Він прозоро масштабується, і ви платите відповідно до ресурсів ЦП і пам’яті, які фактично використовуєте.

Одна з особливостей AWS Fargate - це відсутність можливостей оркестрування, таких як Docker Swarm або Kubernetes. Це робить його ідеальним варіантом для автономних служб, простих мікросервісних додатків або розподілених систем, які не вимагають складної взаємодії між різними

компонентами. Fargate також інтегрується з Amazon Elastic Container Service (ECS), що дозволяє вам використовувати його разом з іншими сервісами Amazon Web Services для розгортання та керування вашими контейнерами в хмарному середовищі.

1.3.3. Docker Swarm

Docker — це програмне забезпечення з відкритим кодом, найпопулярніша платформа для управління контейнерами [7]. Docker Swarm це функція, яка дозволяє кластеризувати та планувати Docker Engines. Ви можете використовувати цю функцію, увімкнувши режим Swarm у Docker.

Режим Docker Swarm пропонує децентралізований дизайн, який гарантує, що Docker Engines можуть обробляти спеціалізації під час виконання. Це дозволяє розгортати як менеджер, так і робочі вузли, і, по суті, створювати рій, використовуючи один образ диска.

Для використання Swarm, потрібно спочатку налаштувати кластер. Кожен кластер має дві типові ролі вузлів: менеджер та робочий. Менеджер відповідає за керування та моніторинг кластеру, в той час як робочий вузол виконує навантаження.

Після налаштування кластеру, ви можете використовувати команди Docker CLI для створення сервісів, що мають бути запущені на кластері. Swarm автоматично розподілить сервіси між робочими вузлами, щоб забезпечити більшу доступність та надійність.

Крім того, Swarm підтримує автоматичне масштабування сервісів. Якщо навантаження збільшується, Swarm може автоматично створювати нові робочі вузли та розподіляти сервіси між ними.

Використання Docker Swarm дозволяє забезпечити високу доступність, надійність та масштабованість для ваших контейнерних програм.

1.3.4. Docker Compose

Docker — найпопулярніший інструмент контейнеризації у світі DevOps. Docker Compose використовується для запуску програм, які мають кілька контейнерів за допомогою файлу YAML. Може бути кілька випадків, коли програма докерів повинна запускати кілька контейнерів для різних стеків технологій. Тепер створення, запуск, підключення окремих докер-файлів для кожного контейнера може бути важким завданням; тут вам допоможе docker-compose. Використовуючи єдиний і простий файл docker-compose.yml, ви можете створити, підключити та запустити всі контейнери, виконавши одну команду. Це дуже корисно для корпоративних програм у виробництві, де кілька програм працюють усередині контейнерів. Це економить багато часу, легко запускаючи 100 програм у контейнерах докерів.

Docker Compose є потужним інструментом для оркестрування контейнерів, зокрема для використання в розробці та виробництві, коли потрібно запускати кілька контейнерів, виконуючи різні функції та технології. Використовуючи файл docker-compose.yml, ви можете описати конфігурацію всіх контейнерів, що потрібні для вашої програми, і запустити їх однією командою, забезпечуючи однакові умови розгортання для всього стеку додатків.

Деякі з переваг Docker Compose включають:

1. Простоту використання: Завдяки файлу YAML, Docker Compose дозволяє описати всю конфігурацію контейнерів у зрозумілому форматі, що полегшує розгортання та управління контейнерами.

2. Однакове середовище розгортання: Docker Compose дозволяє вам описати всі залежності між контейнерами, такі як мережі, об'ємні томи, зовнішні порти, тощо, забезпечуючи однакове середовище розгортання для всього стеку додатків.
3. Швидке розгортання: Одна команда запуску docker-compose дозволяє автоматично створювати та запускати всі контейнери, описані в файлі docker-compose.yml, забезпечуючи швидке розгортання та тестування комплексних додатків.
4. Масштабованість: Docker Compose дозволяє масштабувати контейнери горизонтально, створюючи декілька реплік одного контейнера, що забезпечує високу доступність та надійність додатків.

Загалом, Docker Compose є важливим інструментом в екосистемі Docker, що дозволяє спростити розгортання та управління багатоконтейнерними додатками, забезпечуючи ефективну розроб

1.3.5. Nomad

Nomad, розроблений HashiCorp, є гнучким оркеструвальником контейнерів, який дозволяє організаціям розгортати контейнери та застарілі додатки та керувати ними з однаковим робочим процесом для обох. Він приділяє велику увагу простоті використання.

Nomad дозволяє розробникам розгортати програми за декларативним підходом, використовуючи інфраструктуру як код (IaC), і запускати робочі навантаження, такі як Docker, неконтейнерні програми, мікросервіси та пакетні програми поруч.

Додатково до цього, Nomad має декілька переваг:

- Інтеграція з іншими продуктами HashiCorp, такими як Consul та Vault, що дозволяє покращити безпеку, моніторинг та управління сертифікатами.

- Гнучкі настройки для керування ресурсами, включаючи можливість налаштовувати ресурси для кожної задачі окремо.

- Можливість автоматичного масштабування, що дозволяє автоматично змінювати розмір кластера для забезпечення оптимальної продуктивності.

- Проста інтеграція з іншими інструментами та технологіями, такими як Docker, Kubernetes та AWS.

- Широкі можливості налаштування, що дозволяє налаштовувати різноманітні параметри, включаючи мережеву топологію, диск та обмеження ресурсів.

1.4.Постановка задачі магістерської дисертації

Задача програмного забезпечення для запуску середовища та керування обчислювальними ресурсами у Kubernetes кластері полягає в розробці інструменту, який дозволить легко керувати процесом запуску та управління ресурсами у Kubernetes кластері. Kubernetes - це популярна система контейнеризації, яка дозволяє ефективно керувати розгорнутими контейнерами та ресурсами, що використовуються для їх роботи.

Програмне забезпечення для запуску середовища та керування обчислювальними ресурсами у Kubernetes кластері повинно забезпечувати зручний та простий спосіб налаштування та керування додатками та сервісами, які запускаються у кластері. Це може включати в себе створення та налаштування ресурсів, які необхідні для роботи додатків, такі як обчислювальні та мережеві ресурси, налаштування моніторингу та журналювання,

налаштування механізмів масштабування та балансування навантаження, а також багато іншого.

Додатково, програмне забезпечення повинно забезпечувати автоматизацію процесу встановлення та налаштування Kubernetes кластеру, забезпечувати можливість резервування та відновлення даних та налаштувань, а також забезпечувати можливість розгортання та управління додатками на кластері у автоматичному режимі.

Загалом, завдання програмного забезпечення для запуску середовища та керування обчислювальними ресурсами у Kubernetes кластері полягає в забезпеченні легкого та ефективного керування додатками та сервісами в середовищі Kubernetes, що дозволить розробникам та адміністраторам більш ефективно тестувати написані програми.

ВИСНОВКИ ДО РОЗДІЛУ 1

Під час аналізу існуючих рішень, було проведено визначення хмарної інфраструктури та способів її опису. Було детально розглянуто такі інструменти, як Ansible, Terraform та CloudFormation. Також було проаналізовано інструменти управління контейнерезованими програмами, зокрема Kubernetes, AWS Fargate, Docker Swarm, Docker Compose та Nomad.

На основі проведеного дослідження, можна зробити висновок про те, що інфраструктура хмарного середовища є важливою складовою в сучасних інформаційних технологіях. Для її ефективного управління та розгортання можна використовувати різноманітні інструменти, зокрема Ansible, Terraform та CloudFormation. Також існують спеціалізовані інструменти управління контейнерами, такі як Kubernetes, AWS Fargate, Docker Swarm, Docker Compose та Nomad, які дозволяють ефективно керувати процесом розгортання та масштабування додатків у хмарному середовищі.

У магістерській дисертації, ми будемо досліджувати питання оптимізації використання хмарної інфраструктури та інструментів управління контейнерезованими програмами на прикладі розгортання веб-додатку в локальному та хмарному середовищі. За результатами дослідження будуть запропоновані практичні рекомендації щодо оптимізації використання інфраструктури та інструментів управління контейнерезованими програмами для розгортання веб-додатку в локальному та хмарному середовищі.

2. ЗАСОБИ ОПИСУ ТА КЕРУВАННЯ ОБЧИСЛЮВАЛЬНИМИ РЕСУРСАМИ В ХМАРІ

Для опису середовища та ресурсів у хмарі зазвичай використовуються спеціальні формати, які дозволяють описати необхідні параметри, налаштування та взаємозв'язки між різними компонентами.

Один з найпоширеніших форматів для опису середовища в хмарі - це YAML. YAML - це мова, яка може використовуватися для опису конфігурацій, структурованих даних та інших даних, які можна зберігати у вигляді текстових файлів. В контексті хмарних технологій, YAML-файли можуть містити описи конфігурацій та ресурсів, які потрібно створити або налаштувати у хмарі.

Одним з популярних способів опису ресурсів у хмарі є використання інфраструктури як коду (Infrastructure as Code, або IaC). IaC описує ресурси, які потрібно створити, змінити або видалити, у вигляді коду, зазвичай використовуючи якусь з доступних мов програмування, таких як Python, Ruby, або Go. Такі описи можуть бути збережені у системах контролю версій, таких як Git, та використовуватися для автоматизації процесу налаштування та розгортання середовища у хмарі.

Також існують спеціалізовані інструменти для опису та керування ресурсами у хмарі, такі як Terraform, CloudFormation, або Ansible. Ці інструменти дозволяють описувати і керувати різними хмарними сервісами та ресурсами з використанням конфігураційних файлів або скриптів, що значно спрощує процес керування середовищем у хмарі.

IaC є скороченням від інфраструктури як код (Infrastructure as Code). Це підхід до автоматизованого управління IT-інфраструктурою з використанням кодування. Замість того, щоб ручним чином налаштовувати окремі компоненти інфраструктури, усі налаштування, конфігурації та ресурси визначаються у

вигляді коду. Цей код можна зберігати в репозиторії контролю версій, рецензувати, тестувати та використовувати для автоматичного впровадження та розгортання інфраструктури.

Це дозволяє швидше, безпечніше та більш ефективно управляти інфраструктурою, а також знижує ризик помилок людського фактору, який може стати причиною непередбачуваних проблем. За допомогою IaC можна створювати, масштабувати та управляти інфраструктурою в автоматичному режимі.

Інфраструктура як код (IaC) - це підхід до управління IT-інфраструктурою, при якому налаштування і ресурси інфраструктури описуються за допомогою програмного коду, що може бути збережений та керований в системах контролю версій.

Переваги IaC:

- простота та швидкість;
- одноразовість та передбачуваність;
- масштабованість;
- безпека.

IaC дозволяє швидко та легко налаштовувати та керувати інфраструктурою, можна легко повторювати налаштування та ресурси інфраструктури, що дозволяє запобігати непередбачуваним помилкам та забезпечує однаковість налаштувань в різних середовищах. IaC дозволяє легко масштабувати інфраструктуру, додавати та видаляти ресурси.

Недоліки IaC:

- вимоги до навичок та знань;
- вартість;
- складність.

Інфраструктура як код вимагає від команди розробників додатків та операторів володіння навичками написання програмного коду, що може бути

викликом для тих, хто не має досвіду розробки. Інфраструктура як код може вимагати великих витрат на інструменти, сервіси та ресурси для тестування та розгортання інфраструктури. Інфраструктура як код може бути досить складною та потребувати досвіду в розробці та налаштуванні IT-інфраструктури.

2.1. Формат опису хмарних ресурсів

Формат опису хмарних ресурсів - це стандартизований спосіб опису хмарної інфраструктури та її ресурсів, таких як віртуальні машини, сховища, мережі, сервіси, бази даних тощо. Цей формат дозволяє описувати всі необхідні характеристики цих ресурсів, такі як конфігураційні параметри, розміри, доступні опції, права доступу і т.д.

Найпоширеніші формати опису хмарних ресурсів, які використовуються в інфраструктурному програмуванні (IaC), це YAML, JSON та HCL. Кожен з них має свої переваги та недоліки і використовується залежно від потреб користувача та конкретної ситуації.

Формати опису хмарних ресурсів дозволяють автоматизувати процес створення, налаштування та керування хмарною інфраструктурою, забезпечуючи швидкість, надійність та повторюваність цих операцій.

2.1.1 YAML

YAML (або "YAML Ain't Markup Language") – зручний для читання людиною формат серіалізації даних, концептуально близький до мов розмітки,

але орієнтований на зручність введення-виведення типових структур даних багатьох мов програмування [8]. У контексті кластерів та хмарної інфраструктури, YAML широко використовується як формат опису ресурсів кластера (Рисунок 2.1).

```yaml
stages:
  - build
  - test

build-code-job:
  stage: build
  script:
    - echo "Check the ruby version, then build some Ruby project files:"
    - ruby -v
    - rake

test-code-job1:
  stage: test
  script:
    - echo "If the files are built successfully, test some files with one command:"
    - rake test1

test-code-job2:
  stage: test
  script:
    - echo "If the files are built successfully, test other files with a different command:"
    - rake test2
```

Рисунок 2.1 – Приклад опису ресурсів за допомогою YAML

Формат YAML дозволяє описувати структуровані дані, використовуючи різноманітні типи даних, такі як рядки, числа, булеві значення та масиви. Це дозволяє описувати складні структури даних у зручному для людей форматі, що легко інтерпретується комп'ютером.

У контексті кластерів та хмарної інфраструктури, YAML широко використовується для опису ресурсів кластера, таких як ролі, сервіси, мережі та обмеження ресурсів. Файли YAML можуть бути використані для опису кластера в цілому, або для окремих складових кластера.

Наприклад, YAML-файл може містити опис сервісу, який повинен бути запущений на кластері, включаючи його назву, порт, який повинен бути відкритий, та інші налаштування. Файл також може містити опис мережі, яку повинен використовувати сервіс, або обмеження ресурсів, які повинні бути надані для його роботи.

Крім того, формат YAML підтримується багатьма інструментами управління кластером, такими як Kubernetes та Docker Compose. Це дозволяє розробникам та адміністраторам кластерів легко описувати та керувати ресурсами кластера, використовуючи зрозумілий та зручний формат файлів YAML.

Для опису середовища та ресурсів у хмарі зазвичай використовуються спеціальні формати, які дозволяють описати необхідні параметри, налаштування та взаємозв'язки між різними компонентами.

Один з найпоширеніших форматів для опису середовища в хмарі - це YAML (або "YAML Ain't Markup Language"). YAML - це мова, яка може використовуватися для опису конфігурацій, структурованих даних та інших даних, які можна зберігати у вигляді текстових файлів. В контексті хмарних технологій, YAML-файли можуть містити описи конфігурацій та ресурсів, які потрібно створити або налаштувати у хмарі.

2.1.2 JSON

JSON (JavaScript Object Notation) є одним з форматів опису хмарної інфраструктури. Це текстовий формат обміну даними, який використовується для представлення структурованих даних. JSON використовується для збереження конфігураційних файлів, які описують інфраструктуру в області IT.

```
{
  "io_mode": "async",
  "service": {
    "http": {
      "web_proxy": {
        "listen_addr": "127.0.0.1:8080",
        "process": {
          "main": {
            "command": ["/usr/local/bin/awesome-app", "server"]
          },
          "mgmt": {
            "command": ["/usr/local/bin/awesome-app", "mgmt"]
          },
        }
      }
    }
  }
}
```

Рисунок 2.2 – Приклад опису інфраструктури за допомогою JSON

Формат JSON був створений для зручності зберігання даних у процесі їх обміну між веб-браузером та сайтом або між різними сайтами. Це текстовий формат даних, основною одиницею якого є пара «ключ-значення», працювати з ним можна не тільки через JS, а й через будь-яку іншу популярну мову програмування. JSON має ряд переваг у порівнянні з іншим подібним форматом XML, а його поєднання з AJAX дозволяє вносити зміни до сайтів та веб-додатків без оновлення сторінки [9].

2.1.3 HCL

HCL (HashiCorp Configuration Language) - це мова конфігурації з відкритим кодом, розроблена компанією HashiCorp [10]. HCL розроблений для того, щоб бути зрозумілим для людини і зручним для машини, що дозволяє розробникам та операторам легко писати та підтримувати код інфраструктури (Рисунок 2.3).

HCL використовується для визначення інфраструктури як код в інструментах HashiCorp, включаючи Terraform для керування інфраструктурою

```
io_mode = "async"

service "http" "web_proxy" {
  listen_addr = "127.0.0.1:8080"

  process "main" {
    command = ["/usr/local/bin/awesome-app", "server"]
  }

  process "mgmt" {
    command = ["/usr/local/bin/awesome-app", "mgmt"]
  }
}
```

Рисунок 2.3 – Приклад опису інфраструктури за допомогою HCL

хмар, Vault для керування секретами, Consul для виявлення служб та мережі та Nomad для оркестрування навантаження. Вона використовує простий і

зрозумілий синтаксис з підтримкою коментарів, інтерполяції рядків та типів даних, таких як карти та списки.

Одна з ключових переваг HCL полягає в здатності до модульності коду за допомогою модулів, що дозволяє використовувати код у різних проектах та командах. HCL також підтримує умовну логіку, що дозволяє створювати динамічні конфігурації на основі вхідних змінних або умов середовища.

Загалом, HCL забезпечує потужний та гнучкий спосіб визначення інфраструктури як код, що дозволяє організаціям легко керувати та масштабувати їх інфраструктуру за допомогою сучасних інструментів та практик.

2.1.4 Порівняння форматів опису хмарних ресурсів

YAML, JSON та HCL - це три різних формати для опису конфігурації та інфраструктури в програмному забезпеченні. Кожен з них має свої переваги та недоліки залежно від конкретних вимог та потреб. Нижче наведено порівняння між YAML, JSON та HCL.

YAML:

- читабельний та легкий у використанні;
- добре підтримується для більшості мов програмування;
- підтримує багато типів даних, включаючи об'єкти, масиви, рядки, числа та булеві значення;
- підтримує коментарі, що дозволяє додатково пояснити конфігурацію.

JSON:

- має стандартний формат, який підтримується багатьма мовами програмування;

- легкий у використанні та зрозумілому для більшості розробників;
- має більш обмежений набір типів даних, які включають об'єкти, масиви, рядки, числа та булеві значення;
- не підтримує коментарі.

HCL:

- спеціально розроблений для інструментів інфраструктури HashiCorp, таких як Terraform та Packer;
- легкий у використанні та зрозумілому для більшості розробників;
- підтримує багато типів даних, включаючи об'єкти, масиви, рядки, числа та булеві значення;
- підтримує коментарі;
- має деякі додаткові функції, такі як можливість використовувати змінні та функції.

Загалом, вибір формату залежить від конкретної ситуації та вимог проекту. YAML та JSON зазвичай використовуються для загального використання, тоді як HCL є більш спеціалізованим форматом.

2.2. Kubernetes – інструмент управляння контейнерезованими робочими навантаженнями

Kubernetes є інструментом управління контейнерезованими робочими навантаженнями, що дозволяє розгортати та керувати додатками у контейнерах на різних хмарних платформах та фізичних серверах.

Цей інструмент забезпечує автоматизоване керування контейнерами та додатками, що використовують ці контейнери. Kubernetes дозволяє розгортати додатки в контейнерах, забезпечуючи їх ізоляцію від зовнішнього середовища та

забезпечуючи їхню безпеку. Крім того, він дозволяє автоматизувати процеси масштабування, розгортання та керування контейнерами, що значно спрощує роботу розробників та адміністраторів.

Kubernetes використовує концепцію "контролюючого підходу" до керування додатками в контейнерах. Це означає, що Kubernetes не просто запускає контейнери з додатками, але стежить за їх станом та забезпечує, що вони працюють належним чином, навіть у разі помилок. Крім того, Kubernetes дозволяє масштабувати додатки горизонтально та вертикально, що дозволяє розгортати більше копій додатків для забезпечення високої доступності та продуктивності.

Kubernetes також підтримує різноманітні формати опису ресурсів кластера, включаючи YAML та JSON. Це дозволяє розробникам та адміністраторам кластерів легко описувати та керувати ресурсами кластера, використовуючи зрозумілий та зручний формат файлів. Крім того, Kubernetes підтримує різноманітні інструменти для роботи з контейнерами, такі як Docker.

Архітектура Kubernetes - це система, що складається з різноманітних компонентів, які взаємодіють між собою для управління та розгортанням контейнерів (Рисунок 2.4).

Основними компонентами Kubernetes є:

1) Master-компоненти:
   a) kube-apiserver: компонент, який надає API-інтерфейс для взаємодії з Kubernetes;
   b) etcd: розподілена база даних, використовується для зберігання конфігураційних даних Kubernetes;
   c) kube-scheduler: компонент, який відповідає за розподіл робочих навантажень;

   d) kube-controller-manager: компонент, який відповідає за контроль над класичними ресурсами (наприклад, ReplicaSets, Deployments, StatefulSets тощо) та відповідає за підтримку їх поточного стану.
2) Node-компоненти:
   a) kubelet: компонент, який діє на кожному вузлі і відповідає за запуск контейнерів, їх стан та взаємодію з мастер-компонентами.
   b) kube-proxy: компонент, який відповідає за маршрутизацію мережевого трафіку до контейнерів.
3) Додаткові компоненти:
   a) kubectl: клієнтський інтерфейс командного рядка для взаємодії з API Kubernetes.
   b) dashboard: веб-інтерфейс для моніторингу та управління Kubernetes-кластером.
   c) ingress controller: компонент, який надає зовнішній доступ до вузлів кластеру за допомогою HTTP-запитів.

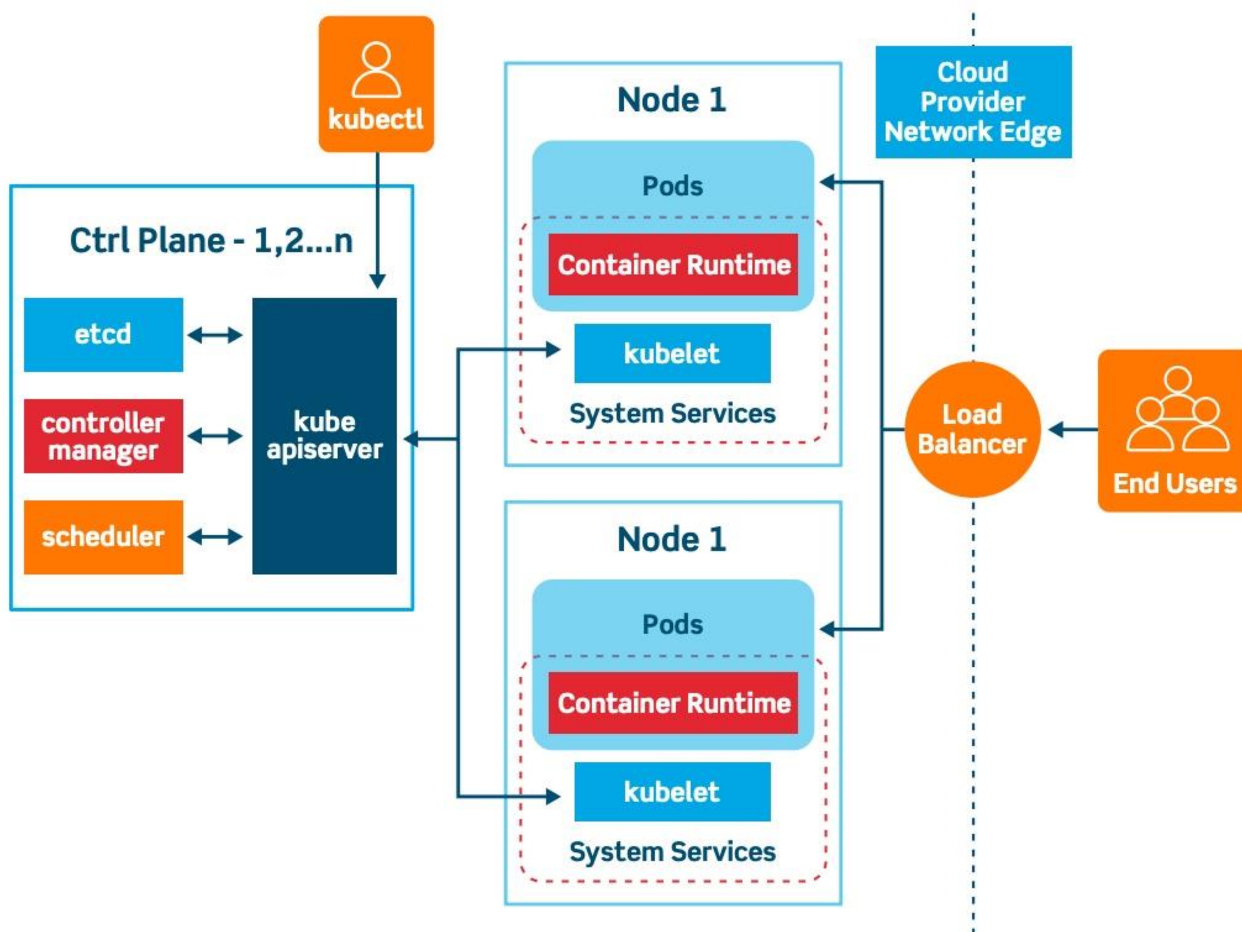

Рисунок 2.4 – Схема компонентів архітектури Kubernetes

Кожен компонент відповідає за свої функції та взаємодіє з іншими компонентами через API-інтерфейс. Разом вони утворюють платформу, яка надає масштабоване та автоматизоване управління контейнерами та додатками.

Розгортання Kubernetes у хмарі можна здійснювати різними способами, залежно від хмарної платформи, на якій буде розгортуватися кластер. Деякі хмарні платформи, наприклад, AWS, Google Cloud і Microsoft Azure, надають свої власні рішення для розгортання Kubernetes, які дозволяють легко налаштувати і керувати кластером.

Використання керованої платформи Kubernetes, наприклад, Google Kubernetes Engine (GKE), Azure Kubernetes Service (AKS) або Amazon Elastic Kubernetes Service (EKS). Ці платформи надають готовий кластер Kubernetes з налаштуваними машинами, мережею та зберіганням.

Використання інструментів для розгортання Kubernetes у хмарі, наприклад, kops, kubespray або kubeadm. Ці інструменти дозволяють розгортати кластер Kubernetes на інфраструктурі хмари, налаштовуючи його за допомогою скриптів або конфігураційних файлів.

Ручне розгортання Kubernetes на інфраструктурі хмари, створюючи власні машини, мережі та зберігання та розгортаючи на них кластер Kubernetes.

У будь-якому випадку, для ефективної роботи з Kubernetes у хмарі необхідно забезпечити належну конфігурацію кластера, включаючи машини, мережі та зберігання, а також налаштування безпеки та моніторингу.

Розгортання Kubernetes dev-like кластеру в локальному середовищі може також бути складним процесом, оскільки вимагає наявності певного обладнання та програмного забезпечення. Для розгортання dev-like кластеру можна використовувати мінімальні налаштування, але все ж таки потрібно виконати деякі кроки:

- встановити Docker для запуску контейнерів;
- встановити Kubernetes, використовуючи механізмі встановлення, що найбільше підходить для вашої ОС;
- налаштувати Kubernetes з допомогою kubeadm або іншого інструменту для керування кластером;
- встановити і налаштувати kubectl, інтерфейс командного рядка Kubernetes для взаємодії з кластером;
- запустити інші компоненти, такі як мережу та балансувальник навантаження, які можуть бути необхідні для вашого додатку.

Однак, найскладнішою частиною може бути налаштування мережі в кластері, особливо якщо використовується багато машин. Необхідно налаштувати мережу таким чином, щоб кожен вузол кластеру міг взаємодіяти з іншими вузлами та зовнішніми сервісами. Крім того, важливо встановити правильні правила безпеки та аутентифікації для кластеру, щоб уникнути можливих загроз безпеці.

2.3. Docker – інструмент контейнерезації

Docker є інструментом для контейнерезації програмного забезпечення, який дозволяє пакувати додатки та їх залежності від іншого програмного забезпечення у зручний для розгортання та управління пакет (Рисунок 2.5). Кожен контейнер Docker містить всі необхідні компоненти для роботи додатку, включаючи код, бібліотеки та системне програмне забезпечення. Це забезпечує відокремленість додатків та їх залежностей від хост-системи та інших контейнерів на тому ж хості.

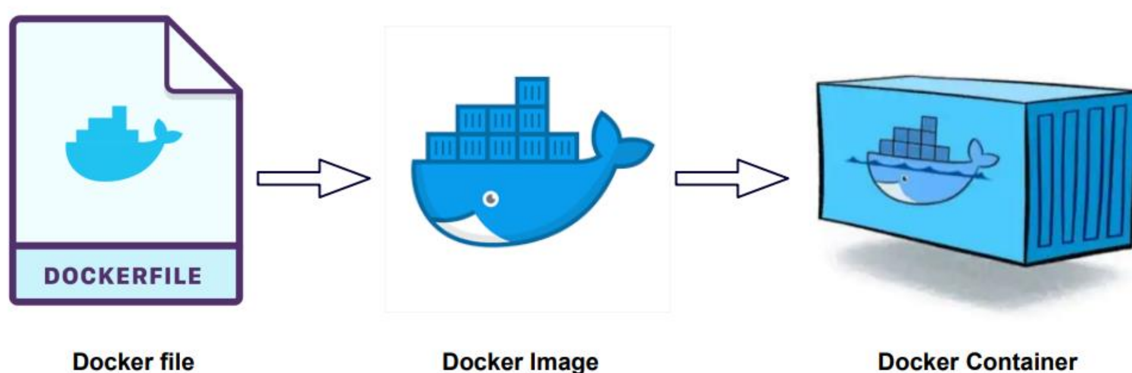

Рисунок 2.5 – Docker як інструмент контейнерезації програмного забезпечення

Docker забезпечує зручний спосіб для пакування та розгортання додатків, що дозволяє зменшити час розгортання та зменшити кількість помилок,

пов'язаних з різницею між середовищами розробки та виробництва. Docker також забезпечує простий спосіб для масштабування додатків та автоматичного розгортання на різних середовищах, включаючи хмарні платформи та локальні сервери.

Одним з найбільш важливих аспектів Docker є його стандартизація. Контейнери Docker можуть бути створені та запущені на будь-якому хості, що підтримує Docker, незалежно від ОС або хмарної платформи. Крім того, Docker забезпечує багато інструментів та сервісів для розробки, розгортання та управління контейнерами, включаючи Docker Compose та Docker Swarm.

Узагальнюючи, Docker є потужним інструментом для контейнерезації додатків та забезпечення їхнього незалежного виконання від інших додатків на тому ж хості. Він також забезпечує зручність розгортання та масштабування додатків на різних середовищах

Docker - це відкрите програмне забезпечення для контейнеризації додатків. Контейнеризація - це метод віртуалізації ОС, в якому різні додатки запускаються відокремлено від інших процесів на системі та залежностей, використовуючи одну ОС-гостьову систему. За допомогою Docker можна пакувати додатки та їх залежності в ізольовані контейнери, які можуть бути розгорнуті на будь-якій системі, що підтримує Docker. Це дозволяє забезпечувати стабільність та переносимість додатків, а також спрощує їх розгортання та керування.

Docker складається зі двох основних компонентів: Docker Engine та Docker Hub. Docker Engine - це програмне забезпечення, яке забезпечує контейнеризацію та керування контейнерами на системі. Docker Hub - це хмарна платформа, що дозволяє зберігати, керувати та ділитися контейнерами та їх образами.

Docker можна використовувати для розгортання та керування контейнерами на одній системі, або в складніших конфігураціях,

використовувати його разом з іншими інструментами, такими як Kubernetes, для керування контейнерами на різних системах та їх оркестрації.

2.4. Способи взаємодії з комп’ютерною системою

Існує кілька видів взаємодії з комп'ютерною системою.

Графічний інтерфейс користувача (GUI) - це інтерфейс, який дозволяє користувачеві взаємодіяти з системою за допомогою графічних елементів, таких як вікна, кнопки, меню тощо. Графічний інтерфейс користувача (GUI) у контексті управління хмарними ресурсами - це інтерактивна графічна оболонка, яка надає користувачам можливість взаємодіяти з хмарною інфраструктурою за допомогою графічного інтерфейсу.

Використання GUI дозволяє користувачам легко керувати різними аспектами хмарної інфраструктури, включаючи налаштування, моніторинг, управління ресурсами та інше. GUI може відображати стан хмарних ресурсів у реальному часі, дозволяючи користувачам відстежувати стан своїх ресурсів та вчасно виявляти проблеми.

Наприклад, хмарні платформи, такі як Amazon Web Services (AWS) та Microsoft Azure, надають користувачам графічний інтерфейс для керування своїми хмарними ресурсами. Це може включати створення, налаштування та видалення віртуальних машин, зберігання даних, мережі та інші аспекти хмарної інфраструктури.

Командний рядок (CLI) – це інтерфейс командного рядка / інтерфейс користувача. Це в основному інтерфейс, в якому користувач може взаємодіяти з Операційною системою для виконання деяких завдань не за допомогою графічного інтерфейсу, а за допомогою введення деяких

команд. Потім інтерпретатор мови команд інтерпретує ці команди і система виконує відповідні дії [11]. В контексті управління контейнерами та хмарною інфраструктурою, CLI зазвичай надає доступ до різноманітних функцій та операцій через команди, які можуть бути виконані з терміналу або віддалено через SSH або інші протоколи. CLI дозволяє адміністраторам та розробникам більш гнучко та ефективно керувати хмарними ресурсами та контейнерами.

Програмний інтерфейс користувача (API) - це інтерфейс, який дозволяє програмам взаємодіяти з системою через задані протоколи та формати даних. Це може бути корисно для розробників, які хочуть створювати програми, які використовують можливості системи.

Програмний інтерфейс користувача (API) - це спосіб взаємодії з комп'ютерною системою, що дозволяє іншим програмам отримувати доступ до функціональності системи та обмінюватись даними з нею. У контексті управління хмарними ресурсами, API дозволяє розробникам і адміністраторам взаємодіяти з хмарними ресурсами за допомогою програм, що запитують та отримують відповіді на запити через API. Це дозволяє автоматизувати процеси управління хмарними ресурсами та забезпечує більш гнучку та ефективну роботу з хмарними сервісами.

API хмарних сервісів можуть бути розроблені різними постачальниками хмарних послуг та використовувати різні протоколи та формати передачі даних, такі як REST, SOAP, JSON, XML тощо. Крім того, кожен хмарний сервіс може надавати свої власні API, що дозволяє взаємодіяти з конкретними хмарними ресурсами та сервісами.

Наприклад, у контексті Amazon Web Services (AWS), програмний інтерфейс користувача дозволяє розробникам та адміністраторам взаємодіяти з різними сервісами AWS, такими як Amazon S3, Amazon EC2, Amazon RDS тощо, за допомогою API запитів, що передаються через HTTP або HTTPS протоколи.

Для роботи з API AWS необхідно мати обліковий запис у системі та зареєструвати створені за допомогою API ключі та секретні ключі.

2.5. Способи візуалізацій залежностей між компонентами системи

Задача візуалізації залежностей між компонентами системи може бути вирішена за допомогою різних інструментів візуалізації графів та діаграм.Один з популярних способів візуалізації залежностей - це використання DAG (Directed Acyclic Graph) (Рисунок 2.6). Цей графічний підхід представляє собою орієнтований граф, у якому кожен вузол відповідає окремій компоненті системи, а ребра відображають залежності між компонентами. DAG використовується в таких інструментах, як Ansible, Airflow, Kubernetes та інших.

Іншим способом візуалізації залежностей є використання Flowchart (діаграми потоків), які відображають послідовність операцій, які необхідно виконати для досягнення певної мети (Рисунок 2.7). Flowchart можна використовувати для моделювання процесів, але вони менш підходять для візуалізації залежностей між компонентами системи.

Також існують інші способи візуалізації залежностей, такі як UML-діаграми, ER-діаграми, схеми баз даних та інші. Кожен з цих інструментів має

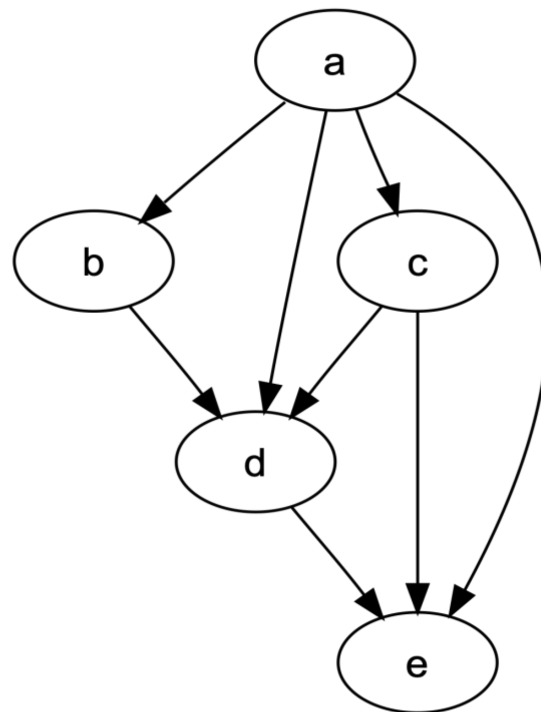

Рисунок 2.6 – Графічне зображення DAG

свої переваги та недоліки, тому вибір способу візуалізації залежностей залежить від конкретної задачі та потреб користувача.

UML-діаграми (Unified Modeling Language) є стандартом для моделювання програмного забезпечення та використовуються для відображення структури, поведінки та взаємодії між різними компонентами програми. UML-діаграми дозволяють відображати класи, об'єкти, відносини між ними та поведінку компонентів.

ER-діаграми (Entity-relationship diagrams) використовуються для відображення структури даних та залежностей між ними. Вони відображають таблиці бази даних та зв'язки між ними, такі як зв'язки "один до багатьох" та "багато до багатьох" (Рисунок 2.8).

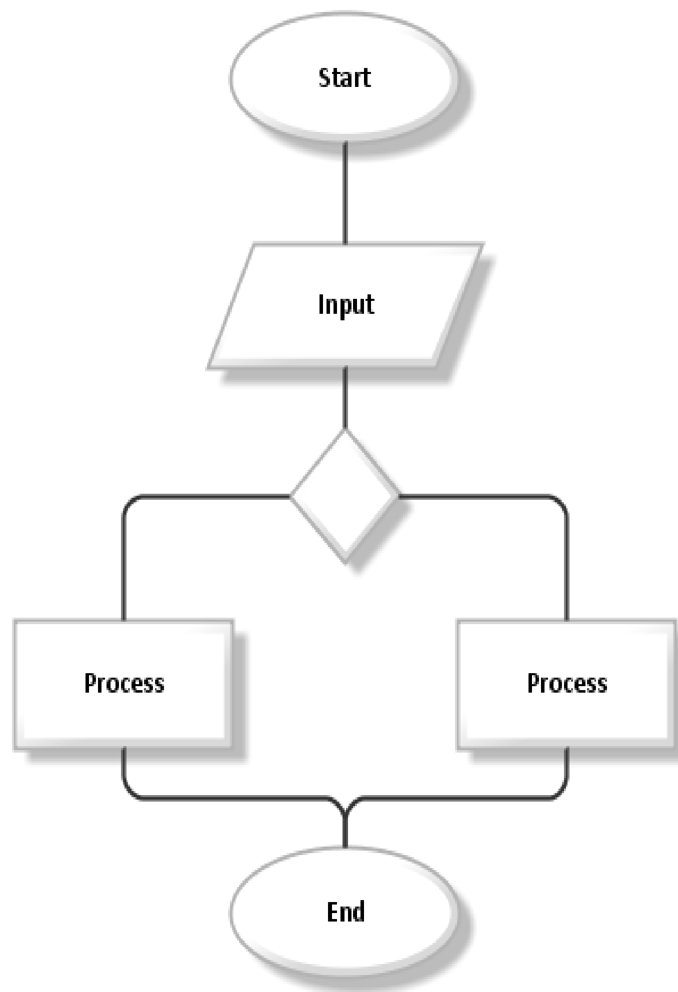

Рисунок 2.7 – Графічне зображення Flowchart

Обидва типи діаграм дозволяють програмістам легше розуміти та відслідковувати залежності між компонентами програмного забезпечення, що робить їх корисними інструментами для розробки та підтримки програмного забезпечення.

Flowchart та DAG (Directed Acyclic Graph) - це дві різні візуалізації, які використовуються для представлення послідовності операцій або процесів. Основна відмінність між ними полягає в тому, що DAG використовується для представлення послідовності операцій, де кожна операція може мати кілька вхідних та вихідних залежностей, тоді як Flowchart використовується для представлення більш складних процесів з великою кількістю різноманітних

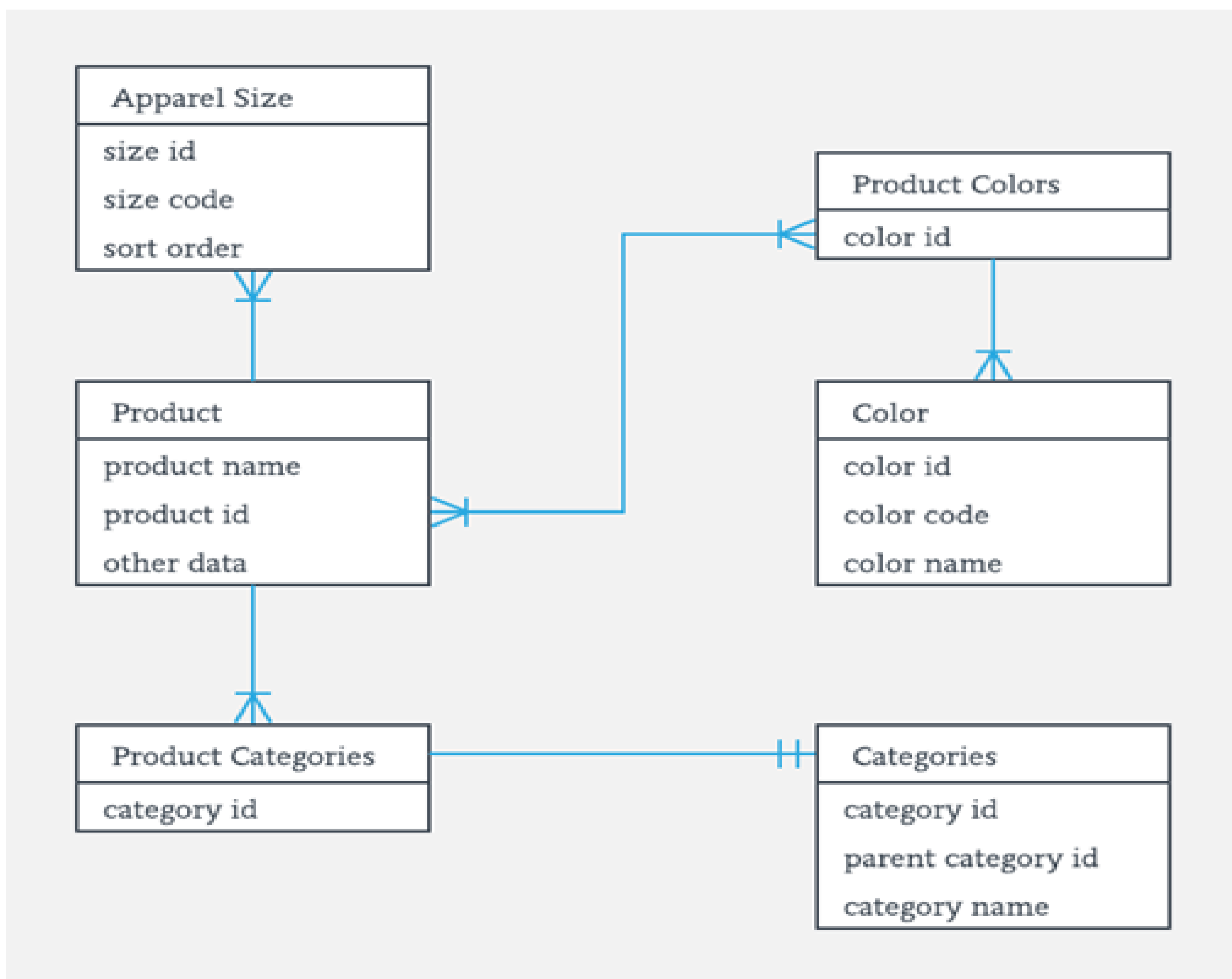

Рисунок 2.8 – Графічне зображення ER діаграми

етапів, які можуть мати не тільки послідовні, але і паралельні або умовні відносини між ними.

Flowchart зазвичай складається з блоків, що представляють окремі етапи процесу та зв'язків між ними, які вказують на послідовність виконання етапів. Діаграми потоку даних (DFD) - це особливий тип flowchart, який показує потік даних між етапами процесу.

DAG зображується як граф, де кожен вузол представляє операцію, а стрілки вказують на залежності між операціями. DAG використовується для моделювання процесів, які можна розділити на частини та виконувати паралельно або відкладати виконання до завершення попередніх етапів. DAG

часто використовується у контексті обробки даних, де потрібно виконувати послідовні операції над великою кількістю даних.

Отже, хоча Flowchart та DAG мають деякі схожість в побудові та візуалізації процесів, вони використовуються для різних типів задач та процесів. Flowchart підходить для більш складних процесів з багатьма етапами та умовними відносинами між ними.

DAG дозволяє легко визначити залежності між компонентами та забезпечує їхню правильну послідовність запуску. У порівнянні з Flowchart, DAG не має зайвої інформації та є більш зручним для організації складних проектів з багатьма компонентами та залежностями між ними. Крім того, DAG дозволяє автоматично визначати потрібні кроки запуску, що спрощує процес розгортання та управління додатком.

Граф залежностей — це спрямований ациклічний граф (DAG), який представляє залежності між компонентами в проекті. Графік організовано за рівнями, де кожен рівень представляє етап у процесі створення та розгортання.

У нижній частині графіка можуть бути базові компоненти, такі як операційна система або середовище виконання. Крім того, можуть існувати такі залежності, як системи баз даних або інші серверні служби. За ними часто йдуть компоненти середнього рівня, такі як бізнес-логіка, а потім інтерфейсні компоненти, такі як компоненти інтерфейсу користувача.

Кожен вузол на графіку представляє компонент, наприклад образ Docker, репозиторій Git або пакет NPM, а кожне ребро представляє залежність між компонентами.

ВИСНОВКИ ДО РОЗДІЛУ 2

Зважаючи на проведене дослідження, можна зробити висновок, що сьогодні веб-розробники стикаються з великою кількістю складних завдань, пов'язаних з розгортанням та управлінням інфраструктурою своїх додатків. Для розв'язання цих задач використовуються різноманітні інструменти та технології.

Одним із таких інструментів є Kubernetes, який є одним з найпопулярніших інструментів для розгортання та управління контейнерезованими додатками. Він забезпечує горизонтальне масштабування, автоматичне відновлення після збоїв, розподіл навантаження та багато іншого.

CLI (Command Line Interface) є досить стандартним та простим інтерфейсом взаємодії, що використовується в більшості інструментів розробки. CLI дає можливість користувачам взаємодіяти з системою з терміналу, що дозволяє більш гнучкий та швидкий доступ до функціональності, особливо для розробників, які часто працюють з терміналом; є більш універсальним інтерфейсом взаємодії, який може бути використаний на будь-якій платформі, що підтримує термінал, незалежно від того, чи це локальний комп'ютер, хмарна платформа або сервер; використання CLI дозволяє легко автоматизувати процеси взаємодії з системою, що робить його більш зручним для використання в CI/CD.

Отже, вибір CLI як інтерфейсу взаємодії є зручним та універсальним рішенням, що дозволяє користувачам взаємодіяти з системою з будь-якого терміналу та легко автоматизувати процеси.

Підхід на базі DAG відповідає вимогам сучасних розподілених систем та є більш гнучким порівняно з іншими підходами. DAG дозволяє визначити залежності між компонентами та розподіляти їх запуск відповідно до цих

залежностей, що дозволяє підтримувати правильну послідовність виконання завдань та ефективніше використовувати ресурси.

Крім того, підхід на базі DAG є більш прозорим та зручним для розуміння та управління, що дозволяє легко визначити та змінювати залежності між компонентами, а також контролювати стан розподіленої системи. Це особливо важливо для складних проектів, де велика кількість компонентів та залежностей можуть ускладнити управління та підтримку системи.

Отже, підхід на базі DAG є оптимальним вибором дозволяє забезпечити ефективну та гнучку роботу розподіленої системи.

3. СТРУКТУРА ТА ОПИС РОБОТИ МОДУЛІВ ПРОГРАМНОГО ЗАБЕСПЕЧЕННЯ

3.1.Мова програмування для імплементації обраного програмного забезпечення

Node.js - це відкрите середовище виконання JavaScript на сервері, що дозволяє розробникам створювати швидкі та масштабовані додатки. Node.js базується на двох ключових компонентах: двигуні V8 JavaScript та бібліотека libuv, що дозволяє виконувати операції вводу/виводу в асинхронному режимі (Рисунок 3.1).

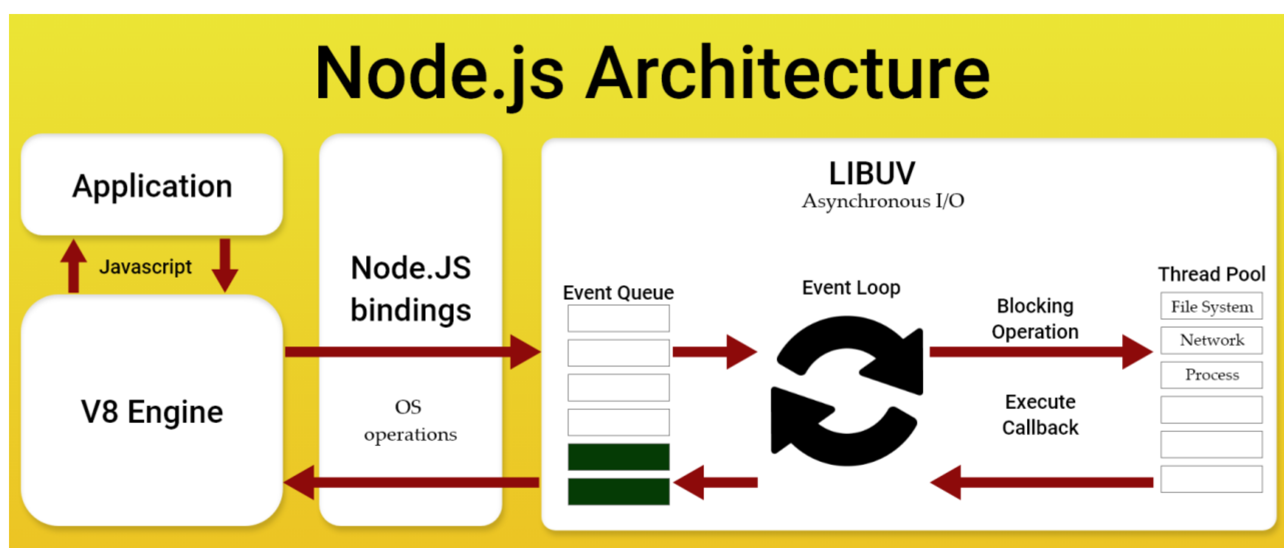

Рисунок 3.1 – Схема Node JS архітектури

Node.js дозволяє створювати серверні додатки, апі, мережеві додатки, інструменти командного рядка, веб-сокети та інші види додатків. Він зручний для використання з базами даних, такими як MongoDB, MySQL та PostgreSQL.

Node.js дозволяє використовувати знайомий синтаксис JavaScript для створення серверної логіки та забезпечує зручний механізм для підтримки

розширень за допомогою модулів Node.js. Він також забезпечує високу продуктивність завдяки своєму асинхронному підходу до виконання коду.

Асинхронний підхід в Node.js забезпечує ефективну обробку багатоопераційних операцій вводу/виводу (I/O), таких як читання файлів або запити до бази даних. Замість того, щоб чекати на завершення кожної операції I/O, Node.js виконує код далі і повертає результати операцій I/O в фоновому режимі.

У Node.js асинхронний підхід реалізується за допомогою колбеків (callbacks), промісів (promises) та асинхронних функцій (async/await). Колбеки - це функції, які передаються в іншу функцію як параметр і викликаються з результатами операції I/O, коли вони стають доступними.

У Node.js також є можливість використовувати проміси та асинхронні функції, що дозволяють писати більш читабельний та легкозрозумілий код. За допомогою промісів, результат операції I/O може бути переданий далі як об'єкт промісу, який може бути вирішений (resolved) або відхилений (rejected) після завершення операції. Асинхронні функції є синтаксичним цукром над промісами, який дозволяє писати асинхронний код так, як він виглядає у синхронному стилі. Незалежно від того, якій техніці асинхронного програмування ви використовуєте в Node.js, важливо забезпечити правильне керування помилками (error handling). Оскільки асинхронний код виконується в фоновому режимі, помилки можуть виникнути пізніше, ніж виклик колбека або вирішення промісу. Тому важливо завжди перевіряти наявність помилок та обробляти їх відповідним чином.

У Node.js є кілька способів керування помилками, включаючи обробку помилок за допомогою try-catch-блоків, передачу помилок у колбек-функцію, використання промісів та обробку помилок на рівні з викликом методів.

Загалом, асинхронний підхід у Node.js дозволяє забезпечити ефективну обробку багатоопераційних операцій вводу/виводу та писати більш ефективний

та продуктивний код. Однак, правильне керування помилками є важливим елементом розробки на Node.js та допомагає забезпечити стійкість та надійність додатку.

Двигун V8 - це відкритий і безкоштовний двигун JavaScript, створений Google для використання у своєму браузері Google Chrome та на серверному підході в Node.js. Двигун V8 є ключовою технологією, яка дозволяє виконувати JavaScript код у браузері та на серверному підході (Рисунок 3.2). Він був розроблений Google у 2008 році та пізніше був опублікований як відкрите програмне забезпечення з ліцензією MIT. V8 є одним з найшвидших двигунів JavaScript і дозволяє виконувати складний JavaScript код в браузері та на сервері з високою швидкістю та ефективністю.

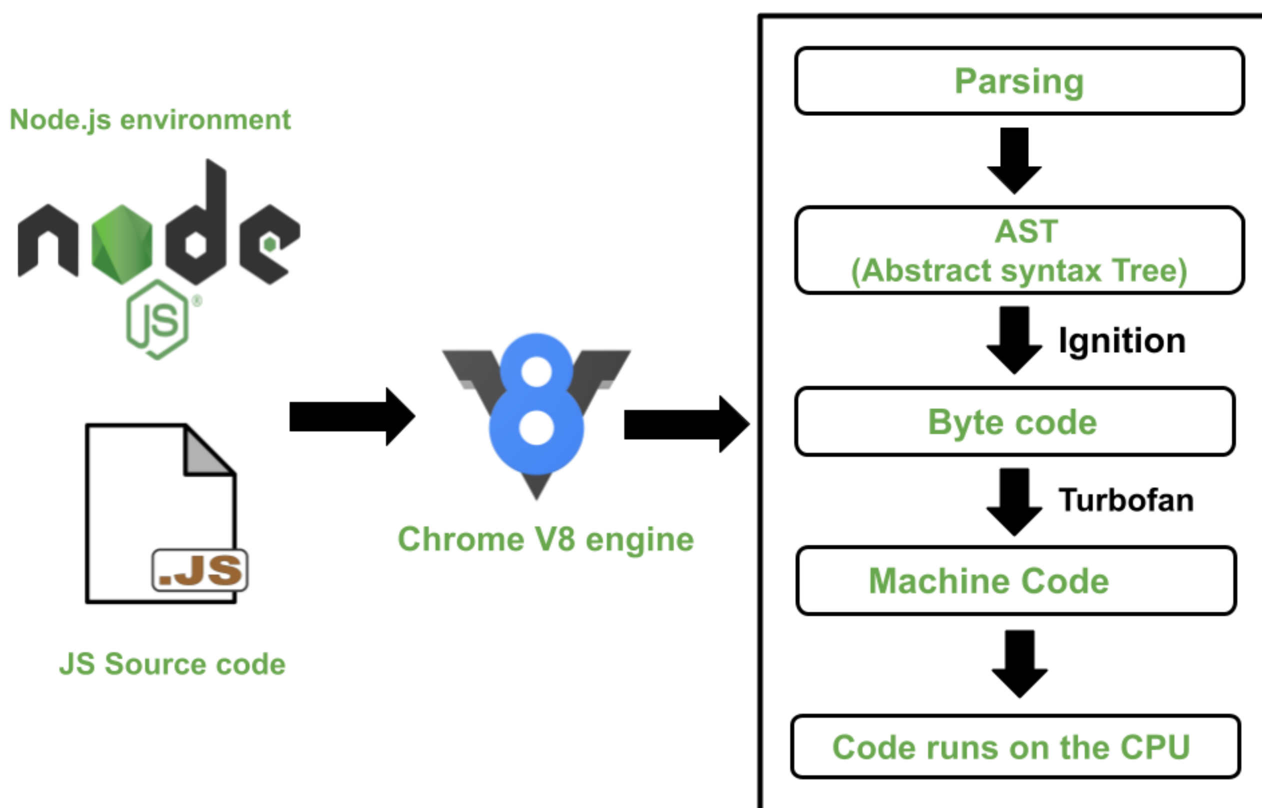

Рисунок 3.2 – Схема роботи двигуна V8

Основним призначенням двигуна V8 є виконання JavaScript-коду в браузері або на сервері, що забезпечує високу швидкодію і продуктивність програм. V8 використовується у багатьох веб-браузерах, таких як Google Chrome, Opera, Vivaldi, а також у Node.js.

Двигун V8 був розроблений з урахуванням вимог до продуктивності та ефективності. Він виконує оптимізацію коду на льоту, щоб забезпечити максимальну продуктивність. Це досягається за допомогою використання динамічної компіляції, яка перетворює JavaScript-код на бінарний код під час виконання програми. Також, V8 використовує ряд технік, таких як inline-кодування та кешування, щоб забезпечити швидкодію виконання JavaScript-коду.

Окрім того, V8 забезпечує підтримку новітніх стандартів JavaScript, таких як ES6, ES7 та ES8, що дозволяє розробникам писати більш сучасний та ефективний код.

Загалом, двигун V8 є важливим компонентом, що забезпечує продуктивність та ефективність виконання JavaScript-коду в браузері та на сервері. Його відкритий та безкоштовний характер дозволяє розробникам використовувати його в своїх проектах безкоштовно та з легкістю розширювати та модифікувати його функціональність.

Libuv (або бібліотека UV) - це кросплатформна бібліотека C, що надає асинхронний ввід/вивід (I/O) та інші базові функції, які використовуються у Node.js для підтримки асинхронного програмування (Рисунок 3.3).

Основною метою Libuv є забезпечення кросплатформеності Node.js та підтримка асинхронного програмування шляхом надання API для взаємодії з операційною системою та апаратними пристроями. Бібліотека підтримує різні платформи, такі як Windows, macOS та Linux.

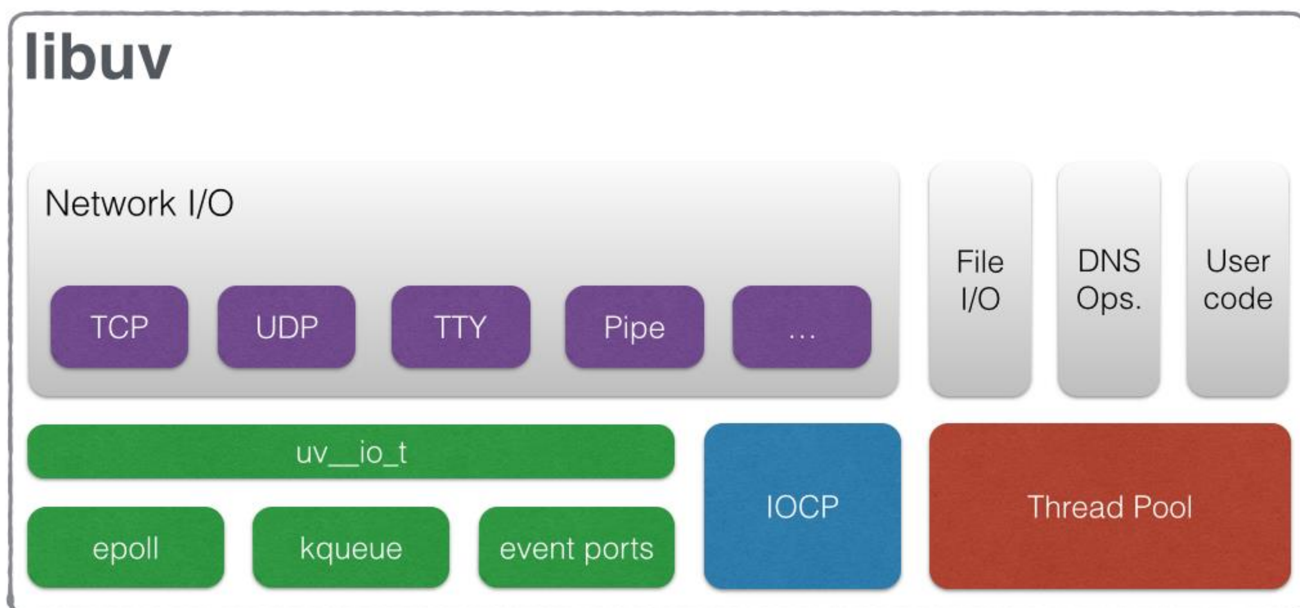

Рисунок 3.3 – Схема компонентів бібліотеки Libuv

Libuv відповідає за керування асинхронним ввідом/виводом (I/O), подіями, таймерами, процесами, асинхронними DNS-запитами та іншими ресурсами системи, що використовуються у Node.js. Завдяки Libuv Node.js може ефективно виконувати багатоопераційні та асинхронні операції на великій кількості з'єднань одночасно.

Загалом, Libuv є важливою складовою у Node.js, що дозволяє забезпечувати асинхронність та ефективність виконання операцій в Node.js. Бібліотека має велику кількість функцій та API, що дозволяють розробникам ефективно взаємодіяти з системою та створювати асинхронні програми.

Node.js має кілька особливостей, які роблять його унікальним та популярним фреймворком для розробки веб-додатків. Ось декілька з них:

Асинхронний та неблокуючий ввод/вивід: Одна з основних особливостей Node.js - це його підхід до вводу/виводу. Node.js використовує асинхронні операції вводу/виводу, що дозволяє багатоопераційним додаткам працювати швидко та ефективно з багатьма з'єднаннями одночасно. Крім того, Node.js не

блокує виконання інших операцій вводу/виводу, що дозволяє додатку працювати безперервно та не впливати на продуктивність системи.

JavaScript на сервері: Node.js дозволяє використовувати JavaScript на сервері. Це дозволяє розробникам використовувати одну мову програмування як на клієнтській, так і на серверній стороні, що спрощує розробку та підтримку веб-додатків.

Модульність: Node.js підтримує модульну архітектуру, що дозволяє розробникам використовувати сторонні модулі та пакети. Це дозволяє спрощувати розробку та підтримку додатків, а також забезпечує більшу повторне використання коду.

Висока продуктивність: Node.js є дуже продуктивним фреймворком, який використовує вбудований двигун V8 для виконання JavaScript. Це дозволяє Node.js працювати швидко та ефективно з великою кількістю з'єднань одночасно.

Масштабованість: Node.js дозволяє легко масштабувати веб-додатки за допомогою горизонтального та вертикального масштабування. Node.js

Розробка інтерфейсу командного рядка (CLI) на Node.js має наступні переваги:

1. Переносимість: Node.js працює на різних операційних системах, тому програми, розроблені на Node.js, можна використовувати на будь-якій платформі. Це особливо важливо для розробки інструментів командного рядка, які часто використовуються на різних операційних системах.
2. Швидкодія: Node.js побудований на двигуні V8, який забезпечує високу швидкодію виконання JavaScript. Це особливо корисно для програм з великою кількістю даних або для програм, які працюють з мережевими запитами.

3. Багатий екосистема: Node.js має багату екосистему пакетів та модулів, які дозволяють розширювати функціональність програми без необхідності написання власного коду. Це дозволяє швидко розробляти програми, що використовуються на командному рядку.
4. Простота: Node.js простий у використанні, тому розробка програм на Node.js дозволяє швидко створювати інструменти командного рядка, що полегшує роботу розробникам та користувачам.
5. Можливість обробки асинхронних операцій: Node.js має підтримку асинхронного програмування, що дозволяє програмам на Node.js працювати з мережевими запитами та іншими операціями без блокування інших частин програми.
6. Можливість інтеграції з іншими інструментами: Node.js можна легко інтегрувати з іншими інструментами командного рядка, такими як Bash або PowerShell, що дозволяє розширювати функціональність програми та її взаємодію з іншими інструментами.

В цілому, розробка інтерфейсу командного рядка на Node.js дозволяє швидко створювати потужні інструменти командного рядка, які можуть використовуватися для автоматизації задач, взаємодії з іншими програмами, роботи з файлами та даними, налаштування системи і багатьох інших сценаріїв. Завдяки високій швидкодії та підтримці асинхронного програмування, програми на Node.js можуть ефективно обробляти великі обсяги даних та мережеві запити, що робить їх особливо корисними для великих проектів з високою навантаженістю. Більш того, Node.js має активну спільноту розробників та багату документацію, що дозволяє легко знайти рішення на більшість проблем, які можуть виникнути при розробці програм на Node.js.

3.2.Принцип роботи та архітектура реалізованої платформи

Програмне забезпечення працює на основі інфраструктури-як-коду (Infrastructure-as-Code), що дозволяє створювати, налаштовувати та управляти інфраструктурою, як будь-який інший код. Це означає, що всі компоненти інфраструктури, включаючи сервери, мережі, сховища даних та інші ресурси, описуються за допомогою коду і зберігаються в системі контролю версій.

Платформа використовує модель DAG (Directed Acyclic Graph) для опису інфраструктури, що дозволяє зручно візуалізувати залежності між компонентами. DAG є графічним представленням об'єктів та зв'язків між ними, де кожен об'єкт представляє окремий елемент інфраструктури, а зв'язки вказують на залежності між ними.

Користувач взаємодіє з платформою через CLI (інтерфейс командного рядка), виконуючи команди, такі як infra deploy для розгортання додатку. Після виконання команди, платформа автоматично стягує необхідні залежності, налаштовує середовище та розгортає додаток в контейнерах Docker.

3.3.Опис архітектури реалізованої платформи

Платформа складається з трьох основних компонентів: Command Line Interface (CLI), Server та Agent (Рисунок 3.3).

CLI – це інтерфейс командного рядка, який дозволяє користувачеві взаємодіяти з платформою та виконувати різні команди, такі як infra deploy, infra status, infra logs тощо. CLI дозволяє користувачам керувати процесом розгортання та керувати модулями проекту.

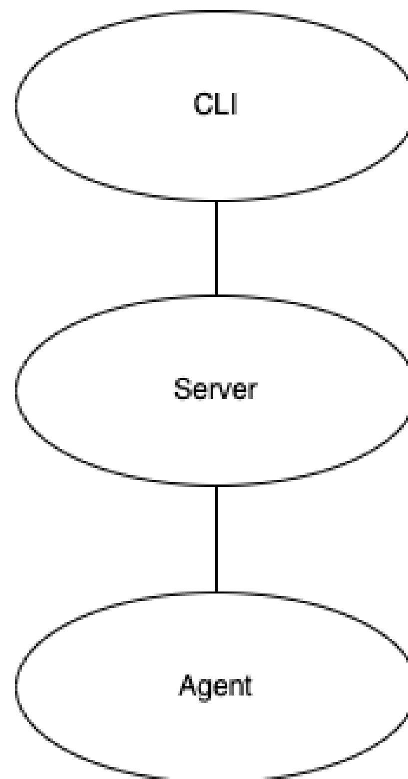

Рисунок 3.3 – Схема основних архітектурних компонентів реалізованого програмного забезпечення

Server – це центральний компонент системи, відповідальний за керування процесом розгортання модулів та координацію між різними компонентами платформи. Server зчитує конфігураційні файли проекту, будує граф залежностей між модулями, відстежує стан системи та взаємодіє з Agent.

Agent – це агент, який виконує розгортання модулів на відповідних вузлах кластера Kubernetes. Agent взаємодіє з Server, приймає від нього команди на розгортання, створює контейнери, налаштовує сервіси, відстежує стан розгорнутих модулів та повертає звіти про виконання.

3.3.1 CLI

CLI - це інтерфейс командного рядка, який забезпечує зручний спосіб взаємодії з платформою. Цей інтерфейс дозволяє налаштовувати та управляти проектами з командного рядка, що дозволяє забезпечити швидкий та зручний доступ до функціоналу (Рисунок 3.3). Інтерфейс командного рядка може бути використаний для створення, редагування, розгортання та керування контейнерами та іншими ресурсами, що використовуються в проектах.

Основні команди CLI включають:

1. **init** - ініціалізує новий проект в поточній директорії.
2. **create** - створює новий сервіс або компонент для проекту.
3. **build** - збирає Docker образи для проекту.
4. **deploy** - розгортає проект на Kubernetes кластері.
5. **delete**- видаляє розгорнутий проект з кластеру.
6. **status** - показує статус розгорнутих проектів на кластері.
7. **logs** - показує журнали (логи) сервісів в розгорнутому проекті.

Виконання команди в командному рядку (CLI) працює наступним чином:

1. Введення команди: Користувач вводить команду в командному рядку свого терміналу або іншому інтерфейсі командного рядка, доступному на його системі.
2. Розпізнавання команди: CLI розпізнає введену команду і визначає, яку дію потрібно виконати. Наприклад, це може бути команда для розгортання проекту, перевірки статусу проекту, оновлення проекту, налаштування параметрів розгортання, тощо.
3. Взаємодія з сервером: CLI взаємодіє з сервером для передачі команди та отримання відповіді від сервера. Це включає передачу конфігураційних

даних, параметрів розгортання та іншої інформації, необхідної для виконання команди.

4. Опрацювання команди на сервері: Сервер отримує команду від CLI і виконує відповідні дії на основі цієї команди. Наприклад, він може створити або оновити конфігураційні файли проекту, взяти на себе відповідні дії з контейнерами на вузлах кластера, перевірити статус розгортання, тощо.
5. Отримання відповіді в CLI: Після виконання команди на сервері, CLI отримує відповідь від сервера, яка може включати результати виконання команди, стан розгортання проекту, повідомлення про помилки або іншу інформацію, яка може бути корисною для користувача.
6. Виведення результатів: CLI виводить результати виконання команди на екран або в інший вихідний потік, доступний користувачу. Це може включати відображення статусу розгортання проекту, відображення даних про контейнери, їх стан та розташування, відображення повідомлень про помилки або статус виконання команди.
7. Обробка помилок: Якщо виконання команди виникають помилки, CLI може обробляти ці помилки і виводити відповідні повідомлення про помилки на екран або в інший вихідний потік. Використання помилкових кодів або повідомлень дозволяє користувачу зрозуміти, що пішло не так та вжити відповідних заходів для виправлення ситуації.
8. Завершення виконання команди: Після виконання команди, CLI повертає контроль користувачу, або може продовжувати очікувати нових команд для виконання.

3.3.2 Server

Server використовується для зберігання та керування конфігурацією проекту, включаючи інформацію про контейнери, сервіси та їх залежності (Рисунок 3.3). Для забезпечення безпеки та контролю доступу, Server підтримує автентифікацію та авторизацію користувачів.

Крім того, Server відповідає за керування процесом розгортання та управління ресурсами. Він автоматично розгортає контейнери та сервіси на вільних ресурсах віртуальної машини або контейнерного кластеру, в залежності від налаштувань.

При створенні нового проекту за допомогою CLI, Server отримує запит на створення проекту та зберігає конфігурацію проекту у базі даних. Після цього Server може взаємодіяти з Agent, щоб розгорнути контейнери проекту на хост-машині.

Server взаємодіє з іншими модулями програмного забезпечення, такими як CLI та Agent, для забезпечення повної функціональності платформи. Він також може бути інтегрований з різними іншими інструментами розробки та управління проектами.

Server зчитує конфігураційні файли з репозиторію, в якому зберігаються проекти. Основний алгоритм зчитування конфігураційних файлів на сервері може бути наступним:

1. Завантаження конфігураційних файлів: Server завантажує конфігураційні файли.
2. Перевірка синтаксису та валідація: Server перевіряє синтаксис та валідність конфігураційних файлів згідно з визначеними правилами та схемою даних. Це включає перевірку правильності формату, наявність

обов'язкових полів, правильність значень та валідацію налаштувань проекту.

3. Побудова DAG: На основі зчитаних конфігураційних файлів, Server будує орієнтований ациклічний граф (DAG), який відображає залежності між компонентами проекту. Це може включати визначення порядку запуску компонентів, розташування контейнерів, налаштування мережевих з'єднань та інші взаємозв'язки між компонентами.
4. Збереження конфігурації: Server зберігає зчитану та оброблену конфігурацію в базі даних.
5. Взаємодія з іншими компонентами: Server взаємодіє з іншими компонентами системи, такими як Agent, для передачі конфігурації та виконання команд. Наприклад, Server може відправляти конфігурацію та завдання на виконання Agent для створення та керування контейнерами на відповідних вузлах.
6. Оновлення конфігурації: Server може відслідковувати зміни в конфігураційних файлах в репозиторії та автоматично оновлювати внутрішню конфігурацію, включаючи перебудову DAG та оновлення інших налаштувань проекту.
7. Обробка помилок: Server відстежує та обробляє можливі помилки при зчитуванні та обробці конфігураційних файлів, включаючи неправильний синтаксис, неправильні значення або відсутність обов'язкових полів. В разі виявлення помилок, Server може повернути відповідні повідомлення про помилку та припинити виконання процесу.

3.3.3 Agent

Agent - це компонент, який встановлюється на хост-машини, де запускаються контейнери (Рисунок 3.3). Його основна функція полягає в управлінні контейнерами, які розгортаються.

При запуску контейнера через Agent виконує декілька дій. Спочатку він отримує інформацію про контейнер з Server, потім встановлює мережеві налаштування та додає контейнер до мережі, якою керує платформа. Після цього він запускає контейнер з допомогою інструментів контейнеризації, таких як Docker або Kubernetes.

Крім того, Agent відповідає за моніторинг та збір логів контейнерів. Він передає цю інформацію до Server, щоб забезпечити моніторинг та аналіз стану контейнерів.

Server та Agent взаємодіють за допомогою REST-протоколу. Server надсилає запити до Agent, щоб відправити конфігурацію проектів та докер-образи, необхідні для їх розгортання. Після отримання необхідних даних, Agent запускає контейнери з використанням Docker API та повідомляє Server про стан контейнерів та ресурсів, які вони використовують.

Server також відповідає за керуванням життєвим циклом контейнерів. Він може відправляти запити Agent на створення, зупинку, перезапуск або видалення контейнерів. Agent слідкує за станом контейнерів та повідомляє про зміни Server. Взаємодія між Server та Agent забезпечує платформу для розгортання та управління додатками в контейнерах.

Коли користувач додає або змінює конфігурацію проекту, Server відправляє цю конфігурацію на Agent на кожній хост-машині, де повинен бути запущений контейнер. Agent розуміє, які контейнери потрібно запустити на основі цієї конфігурації та запускає їх за допомогою відповідних інструментів.

Крім того, Agent періодично відправляє Server звіти про стан контейнерів, що запущені на хост-машині, такі як стан запуску, використання ресурсів та інші метрики. Це дозволяє Server відстежувати стан кожного контейнера та при необхідності змінювати його конфігурацію або зупиняти/перезапускати контейнер.

Агент (Agent) - це компонент, який відповідає за взаємодію з контейнерним середовищем в якому розгортаються компоненти проекту. Основними функціями є виконання команд на вузлі кластера, створення та керування контейнерами, моніторинг стану контейнерів та надання звітності серверу про стан роботи.

Основний алгоритм роботи агента може бути наступним:

1. Очікування команд: Агент очікує команди від Server. Це можуть бути команди на створення, запуск, зупинку, перезапуск, оновлення або інші операції з контейнерами, що розгортаються на вузлі кластера.
2. Виконання команд: При отриманні команди від сервера, агент виконує її на вузлі кластера, де він працює. Це може включати створення або видалення контейнерів, налаштування мережі, виконання інших дій, необхідних для забезпечення розгортання та керування контейнерами.
3. Моніторинг стану контейнерів: Агент моніторить стан контейнерів, що він управляє, включаючи стан запущеності, здоров'я, ресурсів та інших параметрів. Він відправляє цю інформацію на Server для подальшого моніторингу та звітності.
4. Надання звітності серверу: Агент надає звітність Server про стан контейнерів.

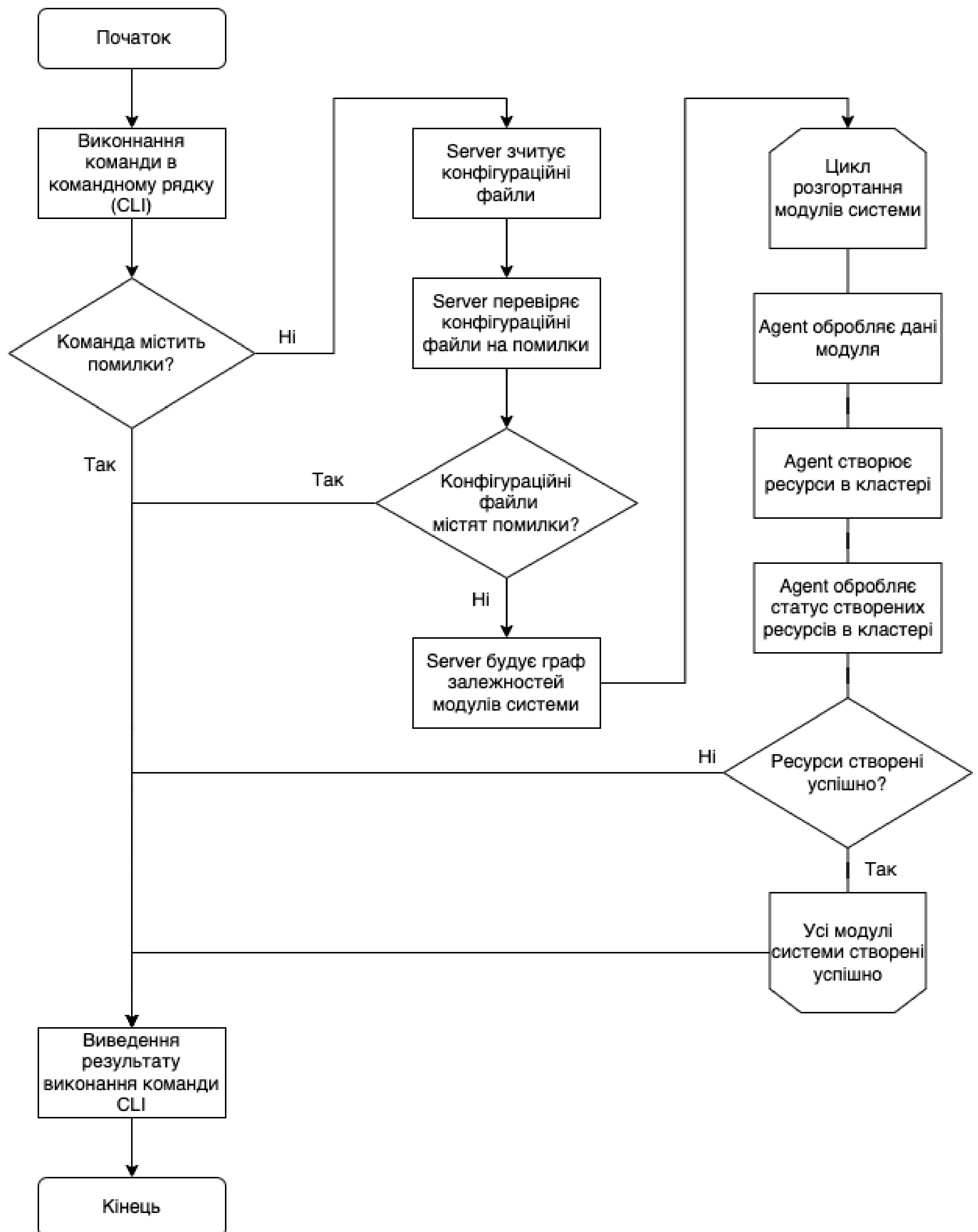

Рисунок 3.3 – Алгоритм роботи реалізованого інструмента

ВИСНОВКИ ДО РОЗДІЛУ 3

В розділі було розглянуто основні характеристики та можливості Node js. Для розробки програмного забезпечення на Node.js використовується мова JavaScript, яка раніше використовувалася переважно на стороні клієнта, а зараз завдяки Node.js використовується для написання серверного коду. Node.js забезпечує підтримку неблокуючого введення/виведення та асинхронного програмування, що дозволяє розробникам створювати швидкі та масштабовані веб-застосунки.

Також було описано архітектуру розробленого програмного забеспечення, яке складається з трьох основних компонентів: CLI (Command Line Interface) - інтерфейсу командного рядка, який дозволяє користувачам взаємодіяти з системою та виконувати різні команди; Server - центрального компонента, відповідального за керування процесом розгортання модулів та координацію між компонентами системи; та Agent - агента, який виконує розгортання модулів на вузлах кластера Kubernetes. Разом ці компоненти дозволяють користувачам ефективно керувати процесом розгортання та керувати модулями проекту.

Було розглянуто розгортання додатків, яке починається зі створення конфігураційного файлу, у якому визначається список модулів, які потрібно розгорнути. Server зчитує конфігураційний файл та будує DAG (Directed Acyclic Graph - орієнтований граф без циклів) модулів, який описує залежності між модулями та порядок їх розгортання. Після побудови графу, сервер передає список модулів Agent - агенту, які потрібно розгорнути.

4. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕСПЕЧЕННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

4.1.Тестування програмного забезпечення

Створюємо каталог demo-project-start з наступною структурою як на рисунку 4.1:

```
➜  demo-project-start: tree
.
├── backend
│ ├── Dockerfile
│ ├── infra.yml
│ └── main.go
├── frontend
│ ├── Dockerfile
│ ├── app.js
│ ├── infra.yml
│ ├── main.js
│ ├── package.json
└── project.infra.yml
```

Рисунок 4.1 – Структура demo-project-start каталогу

Каталог "backend" – містить Dockerfile, infra.yml файл та код бекенд сервісу, написаного на мові програмування Go (main.go).

Каталог "frontend" – містить Dockerfile, infra.yml файл, код та пакетний файл фронтенд сервісу, написаного на мові програмування JavaScript (app.js, main.js) та пакетний файл package.json.

Файл "project.infra.yml" – конфігураційний файл проекту, який використовується для опису ресурсів, що використовуються в цьому проекті, таких як сервіси, інгреси, роути, конфігурації та інші параметри проекту.

Ця структура каталогів використовується для розгортання та управління проектом Kubernetes.

Конфігураційний файл project.infra.yml - визначимо базову конфігурацію для всього проекту. Вказуємо назву проекту, середовище та провайдер (Рисунок 4.2).

```
kind: Project
name: test
environments:
  - name: default
providers:
  - name: local-kubernetes
```

Рисунок 4.2 – Базова конфігурація project.infra.yml

Конфігураційний файл /backend/infra.yml містить налаштування для сервісу з ім'ям "backend" (Рисунок 4.3). Для цього сервісу задаються такі параметри:

1. ports - вказується список портів, які використовує сервіс. У цьому випадку є один порт з іменем "http" і номером контейнерного порту "8080", та номером сервісного порту "80".
2. ingresses - задається список URL-адрес, за якими можна отримати доступ до сервісу. У цьому випадку заданий один URL-адрес зі шляхом "/hello-backend" і портом "http".

```yaml
kind: Module
type: container
name: backend
dockerfile: Dockerfile
services:
  - name: backend
    ports:
      - name: http
        containerPort: 8080
        servicePort: 80
    ingresses:
      - path: /hello-backend
        port: http
```

Рисунок 4.3 – Конфігурація infra.yml бекенд сервісу

Така конфігурація дозволяє іншим сервісам та користувачам звертатись до сервісу backend за допомогою URL-адреси "/hello-backend" та порту "80", сервіс буде слухати на порті "8080".

Для того, щоб запустити модуль в Kubernetes, алгоритм створює Deployment, Service і Ingress ресурси. Ось конфігураційні файли для цього модуля (Рисунок 4.4):

```yaml
// Ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: backend
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
    - http:
        paths:
          - path: /hello-backend
            pathType: Prefix
            backend:
              service:
                name: backend
                port:
                  name: http
```

```yaml
// Deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
        - name: backend
          image: <YOUR_IMAGE_NAME>
          ports:
            - containerPort: 8080
```

```yaml
// Service.yaml
apiVersion: v1
kind: Service
metadata:
  name: backend
spec:
  selector:
    app: backend
  ports:
    - name: http
      port: 80
      targetPort: 8080
```

Рисунок 4.4 – Приклад згенерованих конфігураційних файлі Kubernetes

Наступний конфігураційний файл frontend/infra.yml описує один сервіс з назвою frontend, який має дві точки входу (ingresses): /hello-frontend та /call-backend (Рисунок 4.5). Сервіс слухає трафік на порту 8080 (containerPort).

Ingresses – точки входу, які забезпечують маршрутизацію трафіку до цього сервісу. Кожен вхід (ingress) вказує на шлях (path), по якому трафік буде прийматись. У цьому випадку, трафік, що надходить на шлях /hello-frontend та /call-backend, буде направлятись до сервісу frontend.

Крім того, є залежність від іншого сервісу backend (вказаний як dependencies). Це означає, що сервіс frontend потребує, щоб сервіс backend був запущений та готовий до прийому запитів.

```yaml
kind: Module
type: container
name: frontend
dockerfile: Dockerfile
services:
  - name: frontend
    ports:
      - name: http
        containerPort: 8080
    ingresses:
      - path: /hello-frontend
        port: http
      - path: /call-backend
        port: http
    dependencies:
      - backend
```

Рисунок 4.5 – Конфігурація infra.yml frontend сервісу

Щоб запустити кластер Kubernetes через Docker Desktop, слід діяти за наступними кроками:

1. Встановити Docker Desktop: завантажте та встановіть Docker Desktop на свій комп'ютер, якщо він ще не встановлений.
2. Увімкнути Kubernetes: у налаштуваннях Docker Desktop виберіть пункт "Kubernetes" і переключіть його на "Enable".
3. Перевірити, що кластер запущено: на панелі Docker Desktop має з'явитися значок Kubernetes, після кліку на який можна перевірити стан кластера.

Тепер, коли проект і кластер налаштовані, розпочинаємо розгортання проекту.Результат роботи команди infra deploy. Це вивід з командного рядка (Рисунок 4.6).

```
➔  demo-project-start: infra deploy
Deploy

✔ providers                 → Getting status... → Cached
   ℹ Run with --force-refresh to force a refresh of provider statuses.
✔ graph                     → Resolving 2 modules... → Done
✔ backend                   → Building backend:v-7105f7428a... → Done (took 5.8 sec)
✔ frontend                  → Building frontend:v-278c557118... → Done (took 3.8 sec)
✔ backend                   → Deploying version v-1923036515... → Done (took 3.2 sec)
   ℹ backend                   → Resources ready
   Ingress: http://test.local.app.infra/hello-backend
✔ frontend                  → Deploying version v-7ad9eb5a97... → Done (took 3.2 sec)
   ℹ frontend                  → Resources ready
   Ingress: http://test.local.app.infra/hello-frontend
   Ingress: http://test.local.app.infra/call-backend

Done! ✔
```

Рисунок 4.6 – Результат виконання команди infra deploy

Після виконання команди "infra deploy" у проекті "demo-project-start". Команда відповідає за розгортання інфраструктури проекту з використанням Kubernetes.

Спочатку виводяться повідомлення про перевірку статусу провайдерів та розв'язання графу модулів. Далі проходять процеси побудови бекенду та фронтенду. Після цього розгортується бекенд, і після перевірки ресурсів створюється Ingress для доступу до нього за адресою http://test.local.app.infra/hello-backend. Також розгортується фронтенд і створюються два Ingress-об'єкти для доступу до нього за адресами http://test.local.app.infra/hello-frontend та http://test.local.app.infra/call-backend.

Curl - це команда для взаємодії з додатком через HTTP запити. Результати curl запитів (Рисунок 4.7) демонструють, що додатки успішно розгорнуті і працюють правильно. Кожен запит повертає відповідь, яку додатки повинні повертати - "Hello from backend!" для запиту до бекенду, {"message":"Backend says: 'Hello from backend!'"} для запиту до фронтенду з наступним викликом

```
➜  demo-project-start: curl 'http://test.local.app.infra/hello-backend'
Hello from backend!

➜  demo-project-start: curl 'http://test.local.app.infra/call-backend'
{"message":"Backend says: 'Hello from backend!'"}

➜  demo-project-start: curl 'http://test.local.app.infra/hello-frontend'
Hello from the frontend!
```

Рисунок 4.7 – Результати curl запитів

бекенду, і "Hello from the frontend!" для запиту до фронтенду без виклику бекенду.

Команда infra logs використовується для перегляду журналів (logs) запущених сервісів в середовищі (Рисунок 4.6). У згенерованому результаті можна побачити логи різних сервісів, які були запущені в середовищі - `backend`

та `frontend`. Кожен сервіс має свій окремий лог, в якому знаходяться повідомлення, виведені під час роботи програми.

```
➜  demo-project-start: infra logs

ℹ providers                    → Getting status...
✔ providers                    → Getting status... → Cached
ℹ providers                    → Run with --force-refresh to force a
ℹ graph                        → Resolving 2 modules...
✔ graph                        → Resolving 2 modules... → Done

Service logs:
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
backend → Server running...

frontend →
frontend → > frontend@1.0.0 start /app
frontend → > node main.js
frontend →
frontend → Frontend service started

backend  → Hello from backend!

backend  → Hello from backend!

frontend → Backend says: 'Hello from backend!'

frontend → Hello from the frontend!
```

Рисунок 4.8 – Результати виконання команди infra logs

У даному випадку можна побачити, що сервіс backend вивів повідомлення "Hello from backend!" двічі. Сервіс `frontend` вивів повідомлення про старт

("Frontend service started") та свої логи - "Backend says: 'Hello from backend!'" та "Hello from the frontend!".

Команда infra delete видаляє створене середовище та всі його ресурси з Kubernetes кластера, що були створені під час попереднього деплою. Перш за все, виконується видалення всіх ресурсів, які були створені під час деплою, наприклад, pods, services, deployments, config maps, secrets, ingress, namespace. Після цього видаляється неймспейс, в якому розміщувалися всі ці ресурси. Після успішного видалення виводиться підтвердження (Рисунок 4.9).

```
➔  demo-project-start: infra delete environment
Deleting environment

✔ providers                    → Getting status... → Cached
   ℹ Run with --force-refresh to force a refresh of provider statuses.
✔ graph                        → Resolving 2 modules... → Done
✔ backend                      → Deleting... → Done (took 1.5 sec)
✔ frontend                     → Deleting... → Done (took 1.5 sec)

✔ Cleaning up environments...
   ✔ kubernetes                   → Deleting namespace test-default (this
```

Рисунок 4.9 – Результати виконання команди infra delete

4.2.Аналіз та порівняння результатів

Під час тестування додатку Infra Runner було створено базовий проект, який складається з двох сервісів - backend та frontend. Були описані конфігураційні файли для кожного сервісу, а також описано загальні конфігураційні файл проекту. Після цього було налаштовано локальний кластер

Kubernetes та виконано команду infra deploy, яка згенерувала конфігураційні файли Kubernetes та розгорнула сервіси в локальному кластері. Було перевірено роботу сервісів за допомогою команди curl та продемонстровано спосіб моніторингу роботи сервісів в кластері за допомогою команди infra logs. На завершення була виконана команда infra delete environment для видалення всіх ресурсів в кластері. Було видно, що створення, моніторинг та видалення ресурсів в кластері автоматизовано та робиться за допомогою єдиного CLI. За допомогою уніфікованого підходу та опису ресурсів було автоматично згенеровані усі допоміжні конфігураційні файли для управління ресурсами в кластері.

Це приклад використання інфраструктури як коду (Infrastructure as Code - IaC) та Kubernetes для автоматизації розгортання та керування додатками. Інфраструктура як код дозволяє декларативно описувати інфраструктурні ресурси та їх залежності, тоді як Kubernetes забезпечує управління контейнерами та оркестрування за допомогою керування ресурсами, масштабування та відновлення після збоїв. Використання єдиного CLI для керування ресурсами в Kubernetes дозволяє легко і швидко створювати, моніторити та видаляти ресурси. Всі ці інструменти дозволяють ефективно керувати інфраструктурою та додатками, що зменшує ризики та спрощує процес використання Kubernetes.

ВИСНОВКИ ДО РОЗДІЛУ 4

Для тестування децентралізованого cli додатку для керування ресурсами в kubernetes кластері та розгортання середовища системи, було виконано наступні кроки: перевірка наявності необхідних пакетів та програмного забезпечення; створення базового проекту з двох сервісів (backend та frontend), опис конфігураційних файлів для кожного сервісу та загального конфігураційного файлу проекту; налаштування локального кластеру Kubernetes та виконання команди infra deploy для розгортання сервісів в локальному кластері; перевірка роботи сервісів за допомогою команди **curl**; демонстрація способу моніторингу роботи сервісів в кластері за допомогою команди **infra logs**; виконання команди **infra delete** для видалення всіх ресурсів в кластері.

Після проведення тестування децентралізованого cli додатку було успішно згенеровані конфігураційні файли Kubernetes та розгорнуті сервіси в локальному кластері. Після перевірки роботи сервісів ресурси кластера було успішно видалені.

ВИСНОВКИ

В ході роботи над дисертацією було проведено аналіз існуючих рішень щодо визначення хмарної інфраструктури та її опису. У першому розділі проаналізовано інструменти управління контейнерезованими програмами, такі як Kubernetes, AWS Fargate, Docker Swarm, Docker Compose та Nomad. В результаті цього дослідження зроблено висновок, що інфраструктура хмарного середовища важлива для сучасних інформаційних технологій, і для її ефективного управління та розгортання можна використовувати різноманітні інструменти, такі як Ansible, Terraform та CloudFormation, а також спеціалізовані інструменти управління контейнерами.

У другому розділі було проведене дослідження показало, що веб-розробники стикаються зі значною кількістю складних задач, пов'язаних з управлінням та розгортанням інфраструктури своїх додатків. Для розв'язання цих завдань використовуються різні інструменти та технології, а серед них популярним є Kubernetes, який забезпечує автоматизацію та масштабування контейнеризованих додатків. Були проаналізовані інструменті та підходи для розробки додатку такі, як CLI – стандартним та простий інтерфейс взаємодії, який використовується в більшості інструментів розробки. Використання CLI дозволяє розробникам взаємодіяти з системою з будь-якого терміналу та автоматизувати процеси, зробивши їх зручними для використання в CI/CD. Підхід на базі DAG відповідає вимогам сучасних розподілених систем та є більш гнучким та прозорим, порівняно з іншими підходами. DAG дозволяє визначити залежності між компонентами та ефективніше використовувати ресурси. Це важливо для складних проектів, де багато компонентів та залежностей можуть ускладнити управління та підтримку системи.

У третьому розділі досліджено основні можливості та характеристики Node.js. Завдяки використанню мови JavaScript, яка раніше використовувалася переважно на стороні клієнта, але зараз використовується для написання серверного коду, розробники можуть створювати швидкі та масштабовані веб-застосунки з підтримкою асинхронного програмування та неблокуючого введення/виведення. У другій частині розділу також описано архітектуру розробленого програмного забезпечення, яке складається з трьох компонентів: CLI – інтерфейсу командного рядка, Server – центрального компонента та Agent - агента, який виконує розгортання модулів на вузлах кластера Kubernetes. Разом ці компоненти дозволяють ефективно керувати процесом розгортання та керувати модулями проекту.

У четвертому розділі було успішно протестована робота cli-додатку, в ході якої були успішно згенеровані конфігураційні файли Kubernetes та розгорнуті сервіси в локальному кластері, перевірено роботу сервісів, моніторинг та видалення ресурсів кластера.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Хмарні технології. Переваги і недоліки. [Електронний ресурс] – Режим доступу:https://valtek.com.ua/ua/system-integration/it-infrastructure/clouds/cloud-technologies
2. Інфраструктура як код. [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%84%D1%80%D0%B0%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D1%8F%D0%BA_%D0%BA%D0%BE%D0%B4
3. Ansible. [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Ansible
4. Terraform. [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Terraform
5. Що таке Kubernetes? [Електронний ресурс] – Режим доступу: https://kubernetes.io/uk/docs/concepts/overview/what-is-kubernetes/
6. Вступний посібник з AWS Fargate Переваги і недоліки. [Електронний ресурс] – Режим доступу: https://techukraine.net/%D0%B2%D1%81%D1%82%D1%83%D0%BF%D0%BD%D0%B8%D0%B9-%D0%BF%D0%BE%D1%81%D1%96%D0%B1%D0%BD%D0%B8%D0%BA-%D0%B7-aws-fargate/
7. Що таке Docker і навіщо він? [Електронний ресурс] – Режим доступу: https://qagroup.com.ua/publications/shcho-take-docker-i-navishcho-vin/
8. YAML. [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/YAML

9. Що таке JSON. Усе про цей формат передачі даних в інтернеті. [Електронний ресурс] – Режим доступу: https://apix-drive.com/ua/blog/useful/scho-take-json
10. HCL Native Syntax Specification. [Електронний ресурс] – Режим доступу: https://github.com/hashicorp/hcl/blob/main/hclsyntax/spec.md
11. Вступ до CLI. [Електронний ресурс] – Режим доступу: https://uk.education-wiki.com/2019691-what-is-cli