Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних систем та мереж
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: *Kubernetes кластер в гібридній хмарі для виконання завдань Big Data*

Виконав(ла): студент(ка) *IV* курсу, групи *CI-41*
спеціальності *123 «Комп'ютерна інженерія»*
(шифр і назва спеціальності)

(підпис) *Андруньків С.Р.* (прізвище та ініціали)

Керівник (підпис) *Луцків А.М.* (прізвище та ініціали)

Нормоконтроль (підпис) *Луцик Н.С.* (прізвище та ініціали)

Завідувач кафедри (підпис) *Осухівська Г.М.* (прізвище та ініціали)

Рецензент (підпис) *Гладьо Ю.Б.* (прізвище та ініціали)

Тернопіль
2024

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет *комп'ютерно-інформаційних систем і програмної інженерії*
(повна назва факультету)

Кафедра *комп'ютерних систем та мереж*
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
Осухівська Г.М.
(підпис) (прізвище та ініціали)
« 25 » 04 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня *бакалавр*
(назва освітнього ступеня)

за спеціальністю *123 «Комп'ютерна інженерія»*
(шифр і назва спеціальності)

студенту *Андрунькові Сергію Романовичу*
(прізвище, ім'я, по батькові)

1. Тема роботи *Kubernetes кластер в гібридній хмарі для виконання завдань Big Data*

Керівник роботи *Луцків Андрій Мирославович, к.т.н., доц.*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « 24 » квітня 2024 року № № 4/7-408
2. Термін подання студентом завершеної роботи *24.06.2024 р.*
3. Вихідні дані до роботи *Технічне завдання*

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ
1. Аналіз технічного завдання
2. Проєктна частина
3. Практична частина
4. Безпека життєдіяльності, основи охорони праці
Висновки
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Діаграма взаємодії кластера Kubernetes у гібридній хмарі
2. Розгортання компонентів кластера Apache Spark на платформі Kubernetes
3. Розгортання мультихмарного середовища з використанням Google Anthos
4. Етапи розгортання Kubernetes кластера та запуску Spark додатків

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	*Пилипець М.І., д.т.н., проф. каф. МТ*		

7. Дата видачі завдання 25.04.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	*Ознайомлення із завданням кваліфікаційної роботи*	*01.02-09.02.2024*	*Виконано*
2	*Аналіз технічного завдання*	*05.02-11.02.2024*	*Виконано*
3	*Аналіз вимог до апаратного та програмного забезпечення*	*29.04-02.05.2024*	*Виконано*
4	*Розробка функціональної схеми роботи кластера*	*03.05-10.05.2024*	*Виконано*
5	*Розробка алгоритму розгортання кластера*	*11.05-15.05.2024*	*Виконано*
6	*Вибір та налаштування хмарних платформ*	*16.05-18.05.2024*	*Виконано*
7	*Підготовка terraform-скриптів для розгортання інфраструктури*	*19.05-24.05.2024*	*Виконано*
8	*Розгортання kubernetes-кластера*	*25.05-27.05.2024*	*Виконано*
9	*Написання та запуск BigData-додатків у гібридному кластері*	*28.05-02.06.2024*	*Виконано*
10	*Налаштування системи моніторингу та аналізу результатів*	*03.06-05.06.2024*	*Виконано*
11	*Виконання підрозділу «Безпека життєдіяльності, основи охорони праці»*	*06.06-09.06.2024*	*Виконано*
12	*Оформлення пояснювальної записки*	*16.06-20.06.2024*	*Виконано*
13	*Нормоконтроль*	*20.06.2024*	*Виконано*
14	*Попередній захист кваліфікаційної роботи*	*14.06.2024*	*Виконано*
15	*Захист кваліфікаційної роботи*	*24.06.2024*	*Виконано*

Студент _____ *Андруньків Сергій Романович*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Луцків Андрій Мирославович*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Kubernetes кластер в гібридній хмарі для виконання завдань Big Data // Кваліфікаційна робота на здобуття освітнього ступеня бакалавр // Андруньків Сергій Романович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних систем та мереж, група СІ-41 // Тернопіль, 2024 // с. – 77, рис. – 53, табл. – 1, додат. – 4, бібліогр. – 17.

Ключові слова: Кластер, Kubernetes, Ansible, Kubespray, Helm, YAML, Terraform, інфраструктура, POD, гібридна хмара, Anthos, GCP, GKE, AWS, EKS, контейнер, Docker, Big Data, Spark, Scala.

В ході виконання кваліфікаційної роботи бакалавра було створено гібридну хмару на інфраструктурах Google та Amazon з використанням Google Anthos, на якій розгорнуто та налаштовано Kubernetes кластер для запуску Spark-задач у Docker-контейнерах.

Пояснювальна записка кваліфікаційної роботи містить чотири розділи.

У першому розділі обгрунтовується актуальність теми, визначаються потреби до розгортання Kubernetes кластера у гібридній хмарі. Визначаються вимоги до його продуктивності, масштабування, вибір Big Data технологій, а також коротко описуються принципи роботи використовуваних технологій

Другий розділ присвячений розробці архітектури та функціонування Kubernetes кластера на гібридній хмарній інфраструктурі, враховуючи специфіку задач Big Data. Крім цього, тут розглядаються конкретні інструменти для створення хмари, розгортання кластера та написання додатків із врахуванням особливостей роботи з ними та їх обгрунтованому виборі.

У третьому розділі реалізовується розгортання Kubernetes кластера у гібридній хмарі провайдерів GCP та AWS і написання додатків для кластера.

Четвертий розділ присвячений аспектам безпеки життєдіяльності та основи охорони праці.

ABSTRACT

Kubernetes cluster in a hybrid cloud for Big Data tasks // Bachelor’s thesis // Andrunkiv Serhii Romanovych // Ivan Puluj Ternopil National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Computer Systems and Networks, group SI-41 // Ternopil, 2024 // s. - 77, fig. - 53, tab. - 1, supplement. - 4, bibliog.. - 17.

Keywords: Cluster, Kubernetes, Ansible, Kubespray, Helm, YAML, Terraform, infrastructure, POD, hybrid cloud, Anthos, GCP, GKE, AWS, EKS, container, Docker, Big Data, Spark, Scala.

In the course of the bachelor's thesis, a hybrid cloud was created on Google and Amazon infrastructures using Google Anthos, on which a Kubernetes cluster was deployed and configured to run Spark tasks in Docker containers.

The explanatory note of the qualification work contains four sections.

The first section substantiates the relevance of the topic, identifies the needs for deploying a Kubernetes cluster in a hybrid cloud. It defines the requirements for its performance, scaling, choice of Big Data technologies, and briefly describes the principles of operation of the technologies used.

The second section is devoted to the development of the architecture and operation of a Kubernetes cluster on a hybrid cloud infrastructure, taking into account the specifics of Big Data tasks. In addition, it discusses specific tools for creating a cloud, deploying a cluster, and writing applications, taking into account the peculiarities of working with them and their reasonable choice.

The third section describes how to deploy a Kubernetes cluster in a hybrid cloud of GCP and AWS providers and write applications for the cluster.

The fourth section is devoted to the aspects of life safety and the basics of labor protection.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

GCP – Google Cloud Platform

GKE – Google Kubernetes Engine

GCC – Google Cloud Console

AWS – Amazon Web Services

EKS – Elastic Kubernetes Service

VPC – Virtual Private Cloud

YAML – YAML Ain't Markup Language

HCL – HashiCorp Configuration Language

UI – User Interface

CLI – Command Line Interface

VM – Virtual Machine

IaC – Infrastructure as Code

ВСТУП

У сучасному світі обробка великих обсягів даних стала важливою складовою для багатьох сфер діяльності, від бізнесу до науки. Це призвело до зростання популярності рішень, спрямованих на оптимізацію та швидке вирішення завдань Big Data. Для цього найкраще підходять рішення, які використовують паралельні та розподілені методи обчислень, ізольовані та легко масштабовані середовища виконання коду, розміщені у публічних хмарах з контролем над інфраструктурою.

Розгортання кластерів у гібридних хмарах включає в себе поєднання декількох публічних хмарних послуг (таких як AWS, Azure, GCP) або поєднання приватних хмарних ресурсів (on-premise) та публічних хмарних послуг. Такі підходи дозволяють організаціям поєднати переваги приватних і публічних хмарних ресурсів, забезпечуючи гнучкість, масштабованість та безпеку.

В даній кваліфікаційній роботі реалізовується одне із таких рішень: Kubernetes кластер, розгорнутий в гібридній хмарі GCP+AWS з інтегрованим фреймворком Apache Spark, що є незамінною інфраструктурою для обробки та аналізу великих обсягів даних. Розгортання такого кластера дозволить використовувати переваги хмарного сервісу Google Cloud Platform разом із сервісом Amazon Web Services.

На основі технічного завдання здійснено попередній аналіз вимог до апаратного та програмного забезпечення, що дає змогу врахувати особливості розгортання майбутнього кластера у гібридній хмарі, а також здійснювати тестування на ньому Spark-додатків для оцінки допустимого навантаження.

Використання технології Kubernetes забезпечить автоматичну оркестрацію контейнерів, що в свою чергу призводить до ефективного використання обчислювальних ресурсів для виконання масштабованих додатків обробки даних. Платформа AWS та її інструменту EKS забезпечить гнучке масштабування та використання ресурсів Kubernetes кластера в залежності від потреб, тим самим економлячи багато коштів. Поєднання AWS-кластера із GCP (GKE) буде являти

собою гібридну обчислювальну інфраструктуру, яка буде повністю керуватися за допомогою Google Cloud Console або інших інструментів.

Розміщення кластера на платформах AWS та GCP забезпечить високу доступність навіть у випадку відмови деяких вузлів в певних регіональних зонах на стороні обох інфраструктур.

РОЗДІЛ 1 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ

1.1 Аналіз вимог до гібридної хмарної системи

Для початку проаналізуємо проектовану систему, що стоїть перед нами, визначимо її основні характеристики та мінімальні вимоги. В загальному, виконання поставленого завдання буде відбуватися у чотири головні етапи:

– налаштування проектів, ролей і правил для платформ GCP та AWS;

– створення гібридної хмарної платформи на AWS та GCP, розгортання на ній Kubernetes кластера за допомогою декларативного підходу;

– написання Spark-додатків, пакування їх у образи Docker та розміщення на DockerHub;

– декларативний опис розгортання і запуск Spark додатків (контейнерів) у гібридному кластері.

В ході виконання кваліфікаційної роботи повинні бути виконані такі завдання:

– спроектовані загальна структура та функціональна схема роботи кластера;

– визначені чіткі вимоги до апаратної та програмної частин (к-сть ЦП, обсяг ОЗП, тип і версія ОС, системні утиліти);

– підготовлена локальна програмна платформа для розгортання та адміністрування кластера (gcloud, kubectl, terraform);

– вибрані та обгрунтовані необхідні інструменти для автоматизації розгортання та Big Data обчислень;

– розгорнутий Kubernetes кластер у гібридній хмарі за допомогою Terraform та Google Anthos із врахуванням всіх вимог до апаратної та програмної частин;

– інтегровані BigData інструменти та фреймворки у кластер;

– реалізовані у вигляді коду на мові Scala програми для опрацювання великих даних;

– розроблені програми запаковані у контейнери та створений для них декларативний опис розгортання у гібридному кластері;

– запущені Big Data обчислення на Kubernetes кластері, проаналізовані результати та здійснено моніторинг використання ресурсів;

– отримані результати виконання обчислювальної задачі та журнали її виконання за допомогою засобів моніторингу.

Крім цього, важливо також передбачити потреби у масштабуванні системи з огляду на зростання обсягів даних та завдань в майбутньому. Kubernetes кластер повинен бути здатним ефективно масштабуватися для відповіді на зростаючі потреби обробки Big Data.

До інших вимог стосовно функціонування кластера також належать вимоги безпеки, які включають в себе контроль доступу, шифрування даних, моніторинг безпеки та заходи захисту від зловмисних атак.

Важливо також провести ретельний аналіз вартості розгортання та підтримки, який допомагає визначити оптимальний бюджет для розгортання та підтримки Kubernetes кластера. Враховуючи вартість обладнання, програмного забезпечення, підтримки та інших витрат, можна планувати ефективне використання ресурсів організації.

1.2 Загальні принципи розгортання Kubernetes кластера

Розглянемо детально загальні особливості розгортання кластера та кроки, які необхідно буде при цьому виконати.

Будь-який кластер є множиною взаємопов’язаних обчислюваних вузлів. Вузли можуть бути як фізичними (“bare-metal”), так і віртуальними, що є більш поширеним явищем на сьогоднішній день. Для ефективної спільної роботи над

однією обчислювальною задачею вузли кластера повинні мати свою внутрішню мережу із власним діапазоном адрес, а також використовувати деяку розподілену файлову систему, наприклад HDFS або об'єктне сховище даних (AWS S3, GCP Object storage). Таким чином, при подальшому розгортанні кластера потрібно врахувати етап створення і налаштування внутрішньої мережі та забезпечення розподіленої файлової системи або спільного сховища.

В нашому завданні кожен із вузлів буде брати участь у керуванні або виконанні завдань і відповідно ділитися на дві категорії: master-вузли та worker-вузли.

При розгортанні кластера на віртуальній інфраструктурі варто прорахувати фізичні ресурси машин та грамотно їх розподілити між віртуальними, аби запобігти ситуації їх нестачі або ж надлишку.

Для того, аби розгорнути кластер, потрібні наявні master- та worker-вузли, а також потрібно створити деякий manager-вузол, який не братиме безпосередньо участі в обчисленнях майбутнього кластера, а буде організовувати сам кластер на етапі його розгортання за допомогою певних інструментів автоматизації, таких як Ansible (див. рис. 1.1).

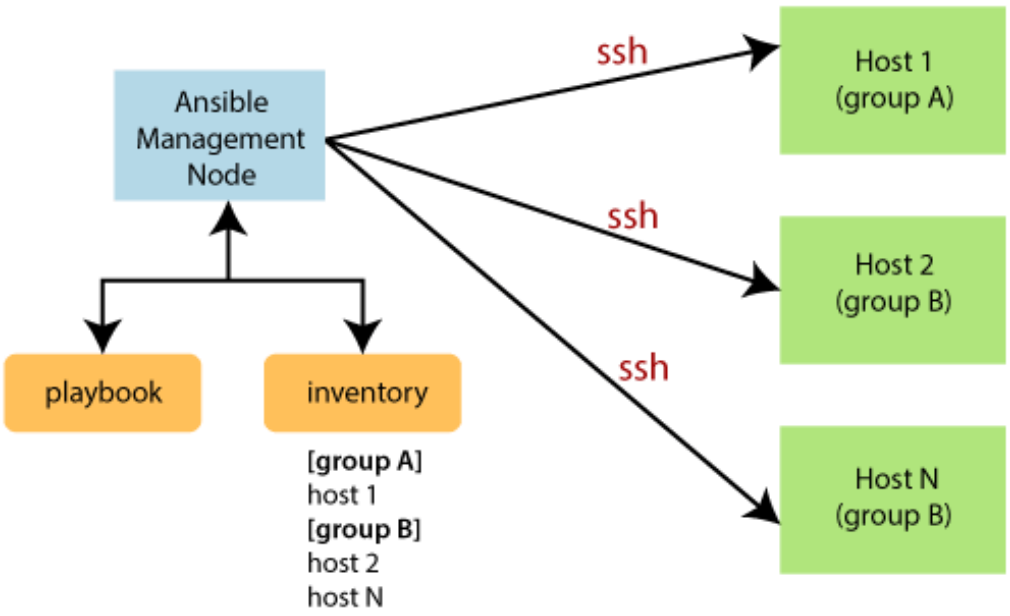

Рисунок 1.1 – Загальна схема розгортання кластера

При розгортанні кластера, практично вся робота виконується на manager-вузлі. На ньому відбувається створення мережі між майбутніми вузлами кластера, написання інструкцій в декларативному форматі на мові YAML для опису правил розгортання інфраструктури кластера. У цьому ж файлі (Ansible playbook) можна задати адреси кожному із вузлів та вказати, які з них будуть master-вузлами, а які worker-вузлами.

Також у даному конфігураційному файлі вмикається/вимикається велика кількість розширень для кластера, наприклад, інструменти та утиліти для моніторингу/керування кластером (Dashboard Web UI).

Manager-вузол використовує SSH-з'єднання з кожним із вузлів кластера, що дозволяє нам ефективно централізовано налаштовувати велику кількість вузлів з використанням декларативних інструкцій мови YAML.

1.3 Створення гібридної хмари та розгортання у ній кластера

Важливим етапом є також створення кластера у гібридній хмарі, що являє собою процес налаштування комбінованого середовища для обчислень, поєднуючи в собі різні приватні та (або) публічні ресурси (див. рис. 1.2). Даний етап вимагає уважного планування та виконання. Основна відмінність від звичайної хмари полягає в тому, що гібридна хмара дозволяє організаціям зберігати та обробляти деякі дані на власних серверах (приватна хмара), а інші дані - у публічних хмарних сервісах.

Загалом, у нас є два варіанти як ми можемо організувати гібридну хмару:

1) приватно-публічна гібридна хмара – частина обчислювальних ресурсів знаходиться на приватній інфраструктурі, контроль над якою повністю належить користувачеві, тоді як інша частина розташована на публічних хмарних платформах, таких як AWS, GCP або Azure;

2) мультихмара – використовуються ресурси з декількох публічних хмарних платформ, таких як AWS, GCP, Azure, IBM Cloud тощо.

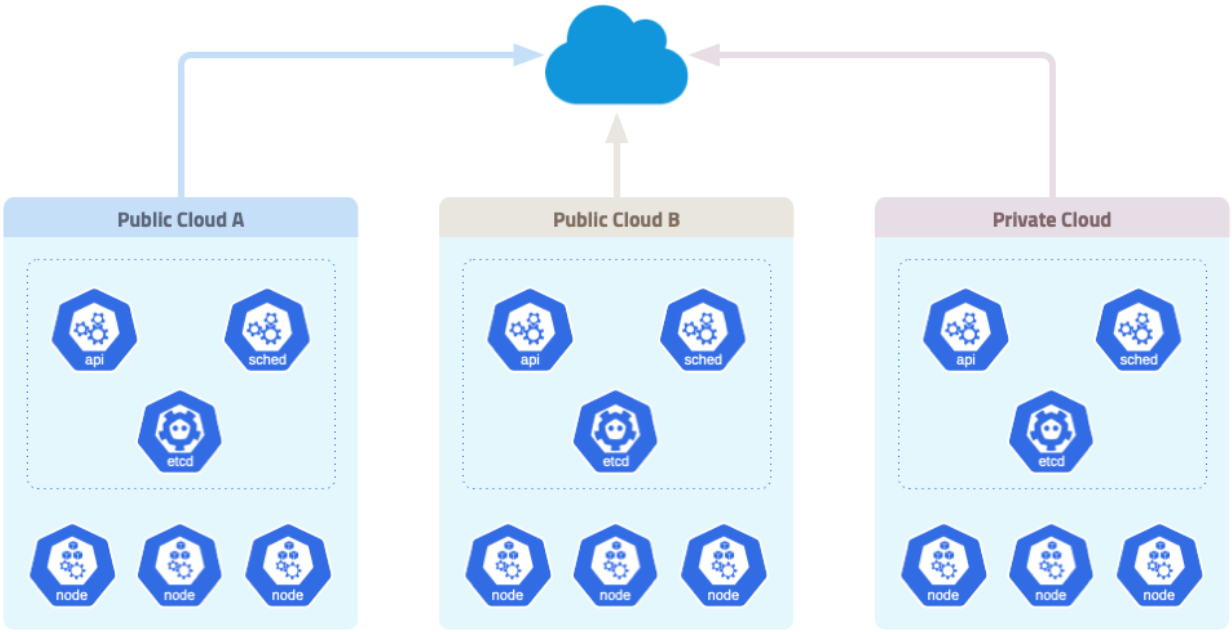

Рисунок 1.2 – Модель гібридної хмари

Багато клієнтів віддають перевагу використанню кількох хмарних провайдерів. Гібридний або мультихмарний підхід має додаткову складність, що пояснюється наявністю декількох інтерфейсів керування, складних наслідків для безпеки та дотримання різних політик використання.

На рисунку 1.3 ми бачимо загальну схему організації гібридної хмари, а також деякий шар «Hybrid cloud manager», який надає інтерфейс керування мультихмарою з дотриманням політик різних провайдерів, організацією безпеки та логування/моніторингу. Тому важливо використовувати інструмент, який надаватиме уніфіковану платформу для керування кластерами у різних хмарах (публічні, приватні та on-premise), дозволить керувати, розподіляти, замінювати та адмініструвати інформацію на різних хмарних сервісах. До цього ж, кластер повинен мати можливість працювати як на екземплярах віртуальних машин, так і на фізичних вузлах (bare-metal).

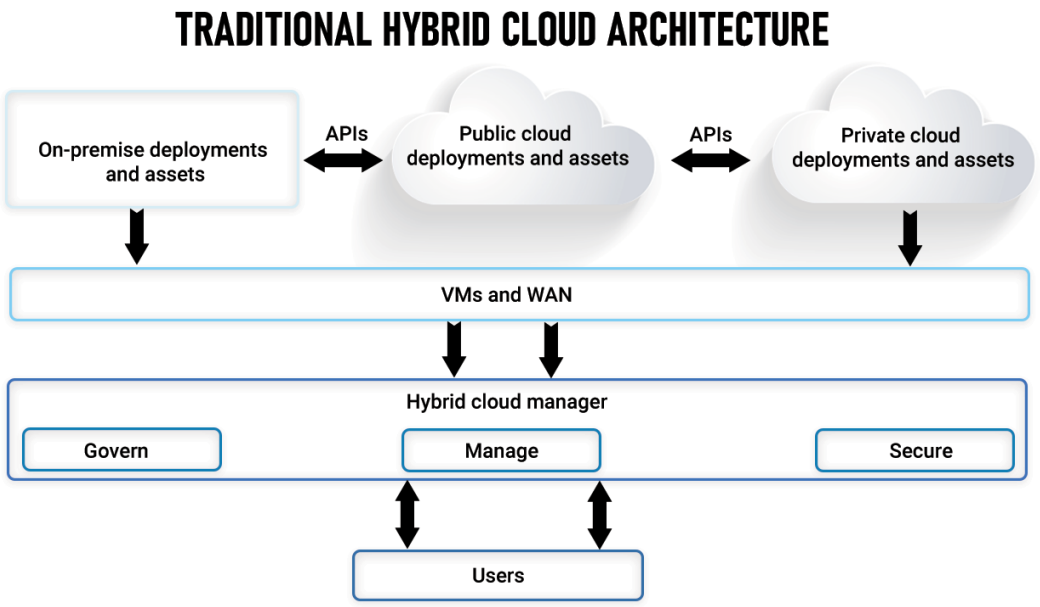

Рисунок 1.3 – Загальна схема керування гібридною хмарою

При розгортанні кластера ми будемо використовувати інструмент Google Anthos, який допоможе спростити керування хмарними службами Google. Аналогами даного інструменту є Microsoft Azure Arc, Amazon EKS Anywhere та VMware Tanzu. Проте, ці інструменти є менш універсальні, гірше документовані, мають менше розвинену екосистему та вищу складність у налаштуванні і управлінні. Тому, виходячи із наведених суджень, було обрано саме цей інструмент.

Найоптимальнішим способом розгортання кластера за допомогою описаного вище інструмента, є використання декларативного опису розгортання. Одним із таких інструментів є Terraform, що працює під капотом у Google Anthos та використовує принцип IaC (Infrastructure as Code).

Основна ідея при виконанні кваліфікаційної роботи полягає в тому, щоб розгорнути вузли кластера у хмарі одного або кількох провайдерів і, водночас, забезпечити інтерфейс керування, розгортання додатків та моніторингу, виключно, у одного з них. Таким чином сервіс керування різними хмарними сервісами різних провайдерів хмарних послуг належатиме до екосистеми, лише, одного хмарного сервісу одного провайдера.

1.4 Реалізація та розгортання додатків Big Data

Оскільки, головною метою і ціллю розгортання нашого кластера є задачі Big Data, варто спочатку розглянути основні поняття та принципи інструментів для обробки великих даних, зокрема ті, які використовуватимуться у даній роботі.

Загалом, Big Data – це концепція, що описує великі обсяги даних, які потребують спеціалізованих методів для збору, зберігання, обробки та аналізу. Зазвичай, ці дані характеризуються досить великим обсягом, різноманітністю (структуровані, напівструктуровані, неструктуровані) і швидкістю збирання та обробки. На сьогоднішній день існує досить багато таких інструментів, які можна інтегрувати у кластерні системи для ефективного їх використання.

Інтегруються ці інструменти багатьма способами. Водночас, вони виступають фреймворками для створення BigData-додатків. До них належать: Apache Spark, Flink, Hadoop MapReduce тощо. Одним із найпоширеніших способів інтегрування фреймворків та відповідних інструментів на кластерах є використання пакетних менеджерів. Пакетний менеджер дозволяє керувати бібліотеками програм, залежностями між ними, репозиторіями, завантажувати і встановлювати з них потрібні інструменти та утиліти для роботи з Big Data, і не тільки, на кластер. У сфері великих даних найпоширенішими мовами програмування є: Scala, Python та Java, відповідно пакетами програм можуть виступати jar-пакунки для Scala та Java з системами автоматичного збирання maven, gradle, sbt, або Python-бібліотеки за допомогою pip, conda, anaconda.

Для виконання задач на обчислювальному кластері застосовуються планувальники обчислювальних завдань, які розподіляють наявні обчислювальні ресурси між обчислювальними завданнями користувачів, керують чергами виконання завдань, а також балансуванням навантаження на кластер. У класичних Hadoop-екосистемах, центральною частиною яких є кластери, використовувався планувальник завдань Apache YARN, проте, з урахуванням більшої ефективності використання обчислювальних ресурсів та гнучкістю все більшої популярності

набуває використання Kubernetes. Kubernetes - це система оркестрації контейнерів з частинами великої обчислювальної задачі, якими виступають виконуючі модулі (Spark executor'и) та керуючі-розподіляючі модулі (Spark driver'и). Kubernetes має свій спеціалізований планувальник обчислювальних завдань для Apache Spark. Водночас, останні версії Apache Spark передбачають хорошу інтеграцію з Kubernetes. У даній роботі розглянуто розгортання платформи Kubernetes для виконання завдань Apache Spark.

Для розгортання сервісів та програм у платформі Kubernetes можна скористатись спеціалізованим інструментом - Helm. Даний інструмент містить так звані helm chart, які є докладним описом розгортання та запуску сервісів і програм на платформі Kubernetes. Він надає можливість гнучко керувати репозиторіями helm chart та практично миттєво розгортати різноманітні сервіси та фреймворки, які описують це.

Розглянемо систему для опрацювання великих даних Apache Spark, яка орієнтована як на потокове (streaming), так і на пакетне (batch) опрацювання даних. У нашому випадку використовуватимемо пакетне опрацювання даних, для чого скористаємось інструментом Apache Spark. Його можна швидко і просто розгорнути за допомогою згаданого вище інструменту helm. Отримавши потрібне середовище для виконання Spark задач, можемо переходити до реалізації конкретного завдання. Apache Spark підтримує кілька мов програмування: Scala, Java, Python і R. Найчастіше використовуються Scala і Python. Оскільки ми будемо писати задачі на мові Scala, то використовуватимемо інструмент sbt для компіляції Scala-коду та формування JAR-архіву. У випадку успішної компіляції наш додаток буде запаковано у JAR-архів.

Для того, щоб даний додаток міг виконуватися на Kubernetes кластері, він має бути поміщений у Docker-образі (див. рис. 1.4), який опублікований на певному публічному репозиторії та в подальшому розгорнутий на цьому кластері. З використанням декларативного підходу та мови YAML, Docker-образ буде

розгоратися у Kubernetes-кластері, який також має засоби для оркестрації виконання цього додатку.

Рисунок 1.4 – Spark-задачі у Docker-контейнері

Більшість Spark-додатків, що запущені в Kubernetes-кластері, можуть отримувати доступ до вхідних даних з різних джерел, які можуть розташовуватися в хмарних середовищах GCP та AWS. Для доступу до даних Spark додатки використовують API відповідних сервісів або протоколи, такі як Hadoop FileSystem API. Також В нашому випадку будемо використовувати сховище Google Cloud Object Storage для вхідних даних наших додатків.

РОЗДІЛ 2 ПРОЄКТНА ЧАСТИНА

2.1 Загальна архітектура роботи кластера

Перейдемо до проєктної частини і побудуємо загальну архітектуру Kubernetes-кластера, яка представлена на рисунку 2.1. Оскільки, ми розгортаємо кластер у гібридній хмарі, то будемо відповідно мати такі шари архітектури:

– шар локальної машини;
– проміжний шар керування кластером;
– обчислювальний шар з ресурсами.

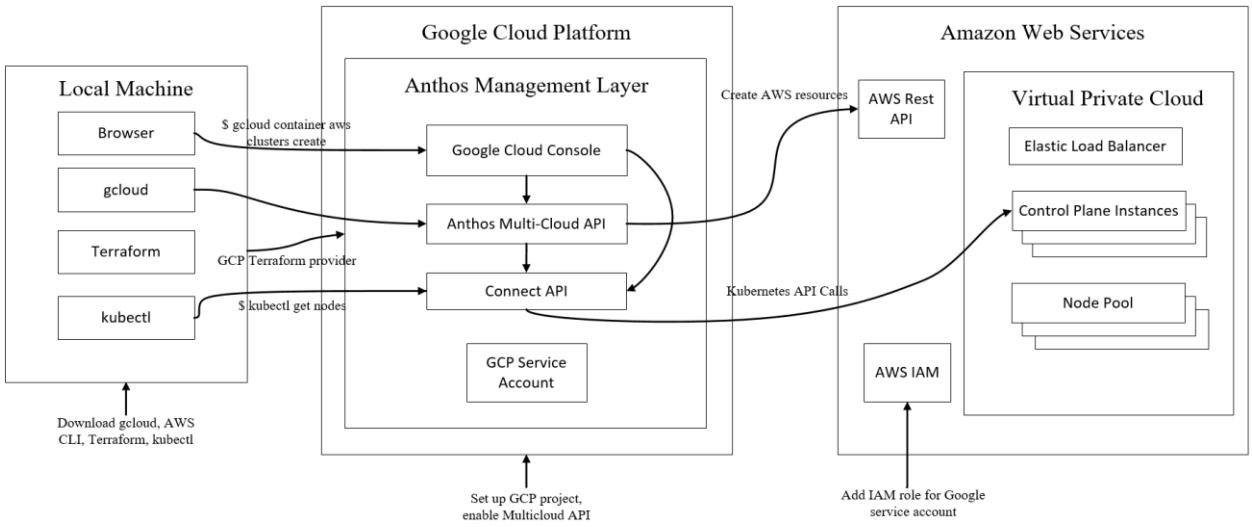

Рисунок 2.1 – Архітектура Kubernetes кластера

Перший шар відіграє роль інтерфейсу керування кластером та централізованого керування і моніторингу ресурсами. Саме тут користувач використовує різні інструменти для розгортання та адміністрування кластера, такі як браузер – для доступу до графічного інтерфейсу GCP та GCC, gcloud і Terraform – для взаємодії із API Anthos Multi-Cloud, kubectl – для взаємодії із

API та керування кластером.

Anthos Multi-Cloud є важливим інструментом у нашій роботі, який буде розглянуто детальніше далі. На його основі базується проміжний шар нашої архітектури. Google Anthos відіграє роль керування обчислювальними ресурсами та вузлами, що знаходяться у хмарі Amazon. На цьому шарі відбувається налаштування проєкт GCP та мультихмарного API. Взаємодіючи із Google Cloud Console через браузер або gcloud, ми використовуємо API Anthos Multi-Cloud, який взаємодіє із AWS Rest API на стороні Amazon, а також звичайний GCP API, що керує екземплярами та ресурсами у AWS.

Третій шар є обчислювальною хмарою у нашому кластері. Тут варто приділити увагу такому компоненту хмарного середовища як VPC. Virtual Private Cloud (VPC) – це приватна мережа, яка створюється у хмарному середовищі для ізоляції та керування ресурсами, такими як віртуальні машини (EC2), бази даних та додатки. VPC надає можливість контролювати свої мережеві середовища, забезпечуючи високий рівень безпеки та гнучкості.

Основні мережеві компоненти Amazon VPC:

VPC – поєднує в собі всі інші мережеві ресурси (згаданий вище);

Subnets – підмережі, що розташовуються в конкретних Availability Zones (можуть бути публічними (із доступом до інтернету) або приватними (без прямого доступу до інтернету));

Internet Gateway – компонент, що дозволяє VPC підключатися до інтернету;

NAT Gateway – надає екземплярам в приватних підмережах можливості ініціювати вихідні з’єднання до інтернету або інших AWS сервісів, не дозволяючи вхідні підключення з інтернету.

Крім мережевих складових, Amazon VPC містить безпосередньо обчислювальні ресурси у вигляді EC2 екземплярів (Control Plane Instances), груп обчислювальних вузлів (Node Pools) та балансувальника навантаження (Elastic Load Balancer).

Також на третьому шарі знаходиться блок AWS IAM, який керує правами доступу користувачів до різноманітних сервісів AWS, задає ролі, правила та політики використання.

2.2 Обґрунтування вибору інструментів для розгортання кластера

Технологій для розгортання кластера існує велика кількість, тому важливо підібрати ті, що найкраще відповідають конкретним потребам та характеристикам проєкту. При обґрунтуванні вибору інструментів слід враховувати функціональні можливості, спільноту та підтримку, інтеграцію з іншими системами, масштабованість та продуктивність. Вибір оптимальних технологій допоможе забезпечити ефективне розгортання та управління кластером, забезпечуючи успішну реалізацію проєкту.

В наступних підрозділах розглянемо конкретні інструменти для розгортання кластера та їх особливості використання в рамках даної кваліфікаційної роботи.

2.2.1 Платформа Kubernetes

Kubernetes є відкритою системою, розробленою компанією Google, для автоматизації розгортання, масштабування та керування контейнеризованими додатками. Вона надає рішення для оркестрації контейнерів, що дозволяє ефективно керувати додатками, які працюють у контейнерах.

Kubernetes став стандартом у сфері контейнеризації та розгортання мікросервісів, він дозволяє компаніям швидше та ефективніше впроваджувати свої додатки в хмарному середовищі, забезпечуючи високу доступність та масштабованість системи.

Дана система еластично масштабує ресурси, а також забезпечує ефективне керування ними (ЦП і пам’ять), що робить його ідеальним вибором для обробки Big Data завдань. Kubernetes забезпечує відмовостійкість та відновлення системи у

випадку відмови вузла. Це гарантує безперервну роботу обробки даних навіть у разі виникнення проблем з апаратним або програмним забезпеченням.

Розгортання у різних хмарних середовищах також є великою перевагою, оскільки дозволяє легко переміщувати та масштабувати додатки Big Data між різними хмарами або між хмарою та локальними середовищами (On-premise). Крім цього, є широка підтримка екосистеми інструментів, що дозволяє легко інтегрувати його з іншими технологіями, такими як Apache Spark, Apache Hadoop, а також з іншими інструментами Big Data.

2.2.2 Мова розмітки та опису даних YAML

YAML – це один із ключових форматів, що використовується для конфігурації та опису ресурсів у Kubernetes. Завдяки своїй простоті, зручності читання та можливості створення складних структур даних, YAML став стандартом для написання маніфестів (конфігураційних файлів) у Kubernetes.

У Kubernetes YAML-файли використовуються для опису різних ресурсів, таких як Pod, Deployment, Service, ConfigMap, Secret та інших. Ці файли містять специфікації того, як має виглядати певний ресурс, його конфігурацію та бажаний стан. Наприклад, для створення Pod у YAML файлі вказується його назва, образ контейнера, порти та інші параметри.

YAML-файли у Kubernetes мають чітку структуру з трьома основними секціями: apiVersion, kind та metadata. Секція apiVersion визначає версію API, яка використовується для створення ресурсу, kind визначає тип ресурсу (наприклад, Pod, Service), а metadata містить метадані ресурсу, такі як ім'я та мітки. Секція spec містить специфікацію ресурсу, яка може включати в себе контейнери, обсяги, політики рестарту тощо.

2.2.3 Утиліта Terraform

Terraform є інструментом для опису інфраструктури як коду (IaC), що дозволяє автоматизувати створення, зміну та видалення інфраструктурних ресурсів

у хмарних середовищах (див. рис. 2.2). Цей інструмент, розроблений компанією HashiCorp, підтримує різноманітні хмарні провайдери, включаючи AWS, GCP, Azure та багато інших.

У Terraform ресурси описуються у файлах з розширенням .tf за допомогою спеціальної мови конфігурації HCL. Ці файли є конфігураційними файлами, які описують бажаний стан інфраструктури. Вони включають в себе визначення ресурсів, змінних та вихідних даних (віртуальні машини, мережі, бази даних). Наприклад, для створення віртуальної машини у AWS у Terraform файлі вказується її тип, AMI, ключі доступу та інші параметри.

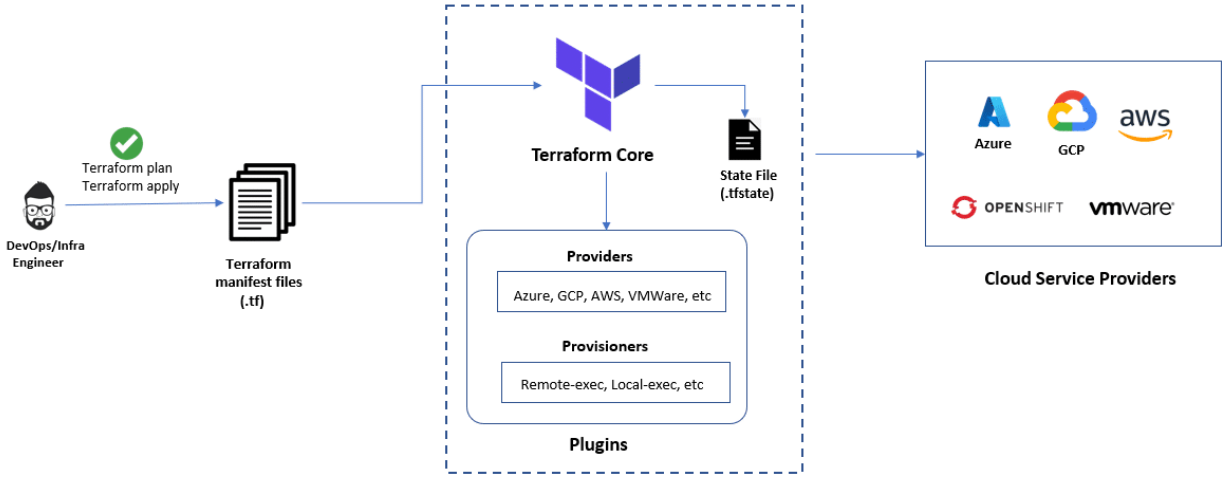

Рисунок 2.2 – Архітектура Terraform

Terraform Core – центральний компонент, який обробляє конфігураційні файли, зчитує маніфест файли, взаємодіє з провайдерами та керує станом інфраструктури. Провайдери – це модулі, які дозволяють Terraform взаємодіяти з різними хмарними провайдерами, такими як AWS, GCP, Azure та іншими. Provisioners використовуються для виконання локальних або віддалених команд після створення або модифікації ресурсів.

Файл стану (.tfstate) зберігає поточний стан інфраструктури. Завдяки ньому Terraform відстежує зміни та забезпечує ідемпотентність. Цей файл важливий для правильного управління інфраструктурою, тому що він допомагає уникнути

дублювання ресурсів та забезпечує коректне внесення змін. Останній компонент на схемі – це хмарні провайдери, з якими взаємодіє Terraform. Вони надають ресурси, які описані у маніфест файлах, та забезпечують інфраструктуру для додатків, що розгортаються.

2.2.4 Kubernetes Dashboard

Kubernetes Dashboard є потужним веб-інтерфейсом для Kubernetes, який дозволяє користувачам керувати та моніторити стан додатків, що працюють у кластері. Він значно спрощує взаємодію з кластером і забезпечує візуальне уявлення про ресурси та їх використання. Основна мета Kubernetes Dashboard – полегшити виконання щоденних завдань адміністрування та управління додатками, надаючи користувачам зручний та інтуїтивно зрозумілий інтерфейс.

Однією з ключових функцій Kubernetes Dashboard є підтримка ролей та облікових записів користувачів, що забезпечує гнучке управління доступом. Це дозволяє адміністраторам налаштовувати права доступу для різних користувачів або груп користувачів, забезпечуючи необхідний рівень безпеки та контролю.

В даній кваліфікаційній роботі ми розгорнемо та налаштуємо ролі для Kubernetes Dashboard у нашому кластері. Після цього він стане доступним через веб-інтерфейс, до якого можна підключитися за допомогою проксі-сервера з використанням токену користувача.

2.3 Огляд інструментів для розгортання кластера у гібридній хмарі

Як раніше згадувалося, гібридна хмара дозволяє об'єднати ресурси кількох хмарних провайдерів і локальних дата-центрів в єдину інфраструктуру. Це забезпечує гнучкість, масштабованість та високу доступність. Розглянемо детальніше основні інструменти для розгортання кластера у гібридній хмарі, які ми будемо використовувати.

2.3.1 Платформи GCP та GKE

В рамках даної роботи GCP та GKE відіграватимуть роль платформ централізованого доступу та керування кластером Kubernetes із AWS. Використання GKE дозволить легко створювати та керувати кластером Kubernetes безпосередньо з консолі GCP, централізовано управляючи ресурсами, розподіляючи навантаження і забезпечуючи високу доступність додатків. GCP надає інструменти для моніторингу, логування та безпеки, що інтегруються з іншими сервісами Google, такими як Anthos. Такий підхід забезпечить ефективне управління мультихмарними середовищами з єдиної точки доступу.

2.3.2 Сховище Google Cloud Storage

GCS є масштабованим та високоефективним сервісом для зберігання об'єктів від GCP. Цей сервіс надає можливість зберігання та доступу до будь-якого обсягу даних у будь-який час і з будь-якого місця в інтернеті. В даній роботі GCS буде використано для зберігання вхідних даних для Big Data додатків, що будуть запущені на стороні AWS.

Рекомендованим методом доступу до GCS є використання сервісних облікових записів Google Cloud. Вони дозволяють налаштовувати детальні правила доступу до ресурсів GCS, забезпечуючи високий рівень безпеки. Ключовий файл JSON, пов'язаний із сервісним обліковим записом, використовується для аутентифікації при запитах до GCS.

2.3.3 Платформи AWS та EKS

AWS разом із сервісом Elastic Kubernetes Service (EKS) є платформою, на якій буде розгортатися наш кластер Kubernetes, що керуватиметься із GCP. На рисунку 2.3 представлено схему роботи сервісу EKS на платформі AWS і які ресурси при цьому задіюються.

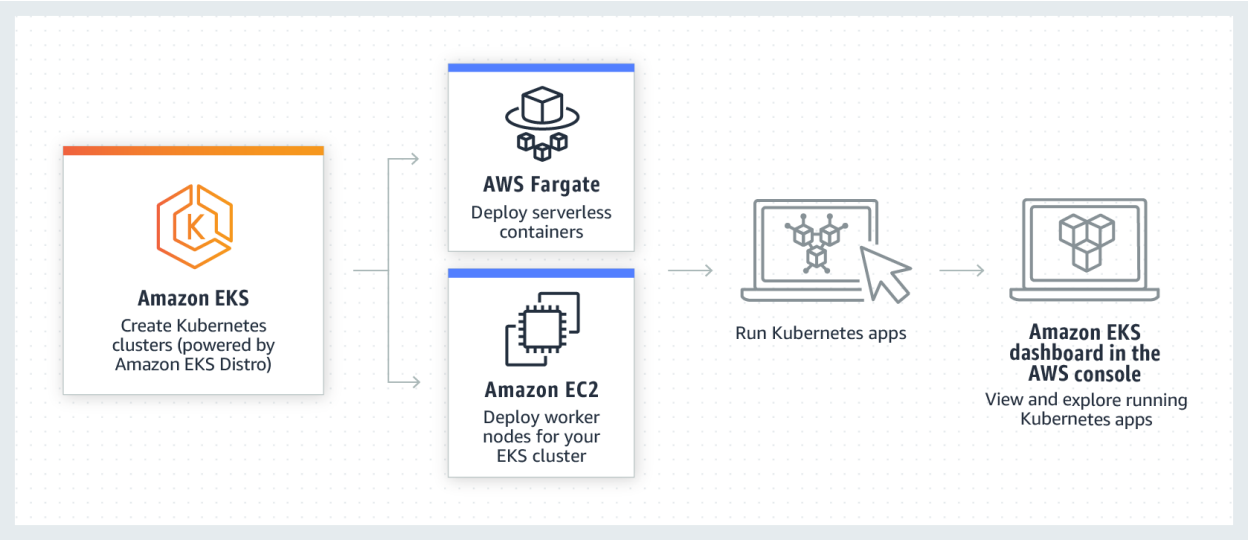

Рисунок 2.3 – Схема роботи AWS та EKS

EKS створює Kubernetes-кластери, використовуючи Amazon EKS Distro, який є дистрибутивом Kubernetes, оптимізованим для роботи в AWS. Fargate дозволяє розгортати контейнери без необхідності управління інфраструктурою, тобто запускати серверлес-додатки, які автоматично масштабуються відповідно до навантаження.

EC2 використовується для розгортання worker-вузлів для EKS-кластерів. EC2 надає велику кількість видів екземплярів під різні задачі та бюджет. Їх буде детальніше розглянуто у практичній частині.

Консоль AWS EKS Dashboard є також компонентом архітектури EKS, що надає зручний інтерфейс для перегляду та управління запущеними Kubernetes-додатками, проте в нашому випадку для цього будуть використовуватися привілегії GCP.

2.3.4 Платформа Google Anthos

Google Anthos – це платформа для управління мультихмарними та гібридними середовищами, яка дозволяє розгортати та управляти контейнеризованими додатками на будь-якій інфраструктурі, включаючи GCP,

AWS, Azure та локальні дата-центри. Anthos забезпечує єдину платформу для розгортання, моніторингу та управління Kubernetes-кластерами та додатками.

Розглянемо детальніше архітектуру Google Anthos, яку зображено на рисунку 2.4. У самому центрі Anthos знаходиться Kubernetes Control Plane, який керує всіма кластерами Kubernetes, незалежно від того, де вони розташовані: у GCP, на on-premise, чи у інших хмарних провайдерів (AWS, Azure). Це забезпечує єдину точку управління для всіх контейнеризованих додатків. Шар Multicluster Automation відповідає за автоматизацію управління кількома кластерами. Він дозволяє автоматично масштабувати ресурси, розгортати оновлення і забезпечувати балансування навантаження між різними кластерами. Завдяки цьому, адміністратори можуть ефективно керувати великими і складними середовищами без значних зусиль.

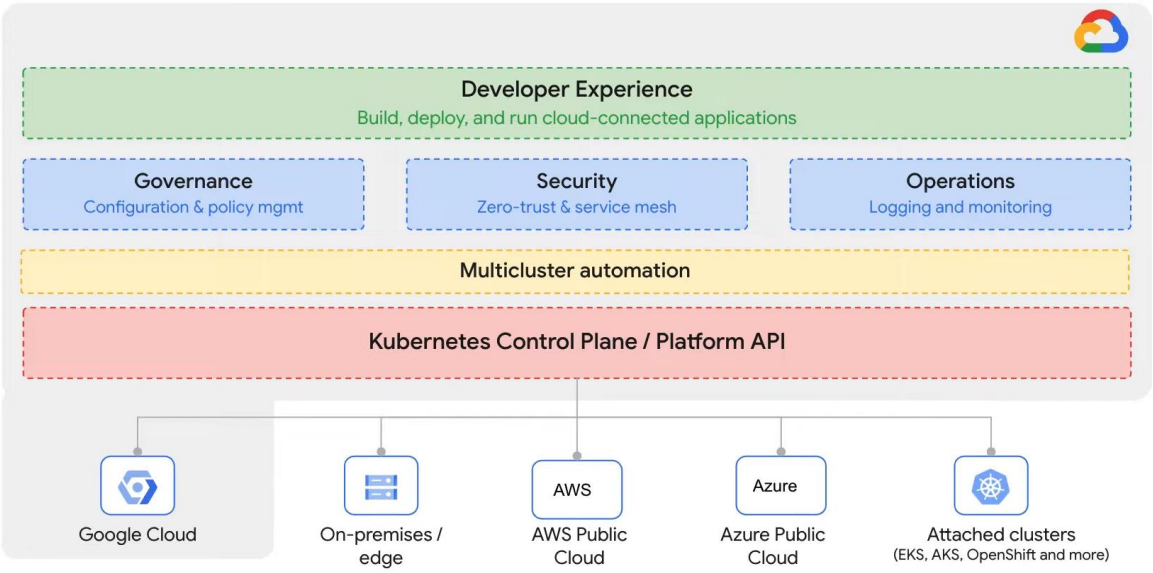

Рисунок 2.4 – Архітектура Google Anthos

На третьому рівні ієрархії знаходяться три компоненти:

– Governance – дозволяє використовувати єдиний підхід до налаштувань безпеки та керування доступом серед усіх хмарних провайдерів;

– Security – потрібен для впровадження аутентифікації, авторизації та шифрування трафіку між сервісами;

– Operations – забезпечує детальну видимість роботи додатків і інфраструктури.

Найвищий шар Developer Experience надає розробникам інструменти для швидкого створення, розгортання та управління додатками, підключеними до хмари.

2.4 Принципи роботи Apache Spark у Kubernetes

Розглянемо детальніше принцип роботи Spark-фреймворку на кластерних системах. На рисунку 2.5 представлено архітектуру Apache Spark, яка демонструє, як Spark обробляє дані з використанням проміжного менеджера кластера. До основних компонентів архітектури можна віднести Driver Program, Cluster Manager та Worker Nodes.

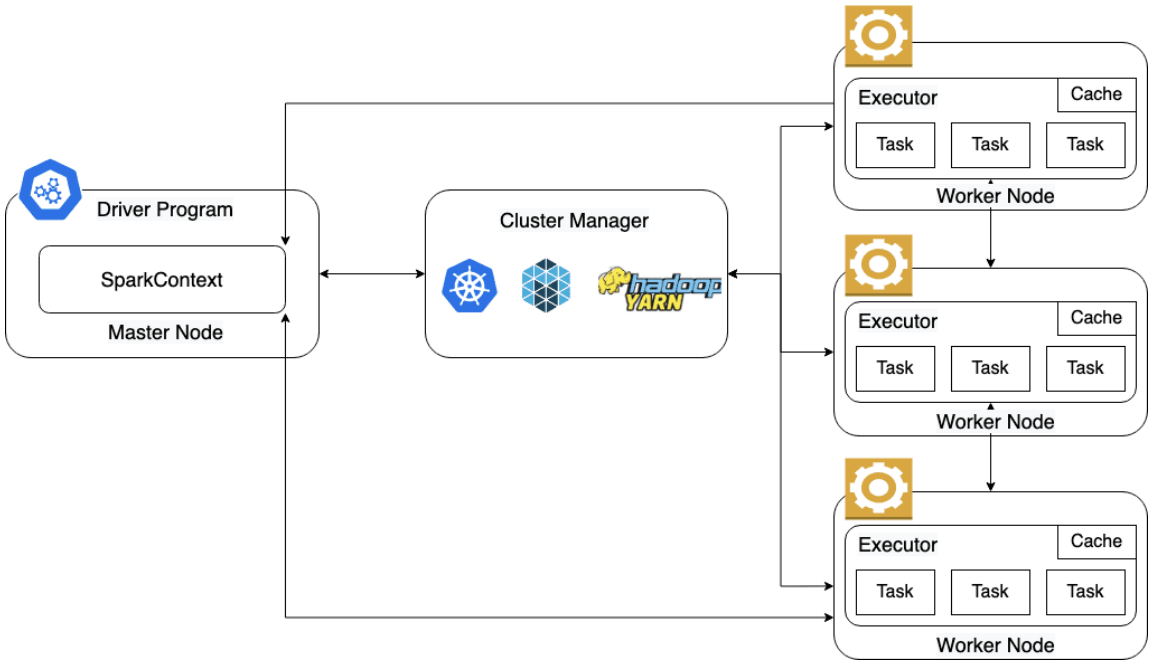

Рисунок 2.5 – Архітектура Apache Spark у поєднанні з Cluster Manager

Архітектура самого Apache Spark в кластерних середовищах передбачає виділення одного головного компонента, так званого Spark-драйвера, серед всіх

інших вузлів, що беруть на себе роль виконавців (“executor”). Компонент Driver Program на попередньо згаданому рисунку репрезентує один із вузлів Kubernetes кластера, якому відведено роль master-вузла в контексті Spark. Саме на цьому вузлі буде функціонувати Spark-драйвер і контролювати роботу всіх executor-вузлів, розподіляти навантаження між ними, забезпечуючи тим самим паралельну та розподілену пакетну обробку даних. Призначення одному із вузлів ролі Spark-драйвера, передбачає створення на ньому Spark-контексту – об'єкту, що ініціалізує Spark-додаток і встановлює зв'язок з кластером для керування ресурсами. Він створюється на стороні драйвера і контролює виконання програми.

Компонент Cluster Manager керує ресурсами кластеру. Spark підтримує різні типи менеджерів, такі як Kubernetes (кластери Kubernetes), YARN (Hadoop кластери) та Standalone Cluster Manager (власний менеджер від Spark).

Останній компонент Worker Nodes складається складається із вузлів кластера, які виконують завдання. Зазвичай, кожен із робочих вузлів містить по одному виконавцю («executor»). Spark-executors – це процеси на worker-вузлах, що виконують задачі («tasks») і зберігають дані в кеші («cache»). Задачі в контексті Spark є окремими частинами роботи, які виконуються на executors.

Нижче наведено загальний принцип роботи Spark та виконання задачі з використанням менеджера кластера, як проміжного компонента:

– SparkContext створюється у драйвері (Master Node);

– SparkContext підключається до Cluster Manager і запитує ресурси (CPU, RAM) для виконання завдань;

– Cluster Manager розподіляє ресурси і ініціює Executors на worker вузлах;

– Executors запускаються на worker вузлах і чекають на задачі від драйвера;

– Driver Program розбиває роботу на дрібні задачі (tasks) і розподіляє їх між Executors;

– Executors виконують задачі паралельно на різних worker вузлах;

– Executors виконують обчислення, обробляють дані і зберігають результати у кеш (якщо це передбачено);

– результати передаються назад драйверу.

В даній архітектурі ініціалізація та управління ресурсами здійснювалися через окремий Cluster Manager (наприклад, Kubernetes, YARN, Standalone). Крім вищенаведеної архітектури, можливий також підхід, що дозволяє повністю інтегрувати управління ресурсами в Kubernetes, де API Server виконує роль диспетчера ресурсів (див. рис. 2.6).

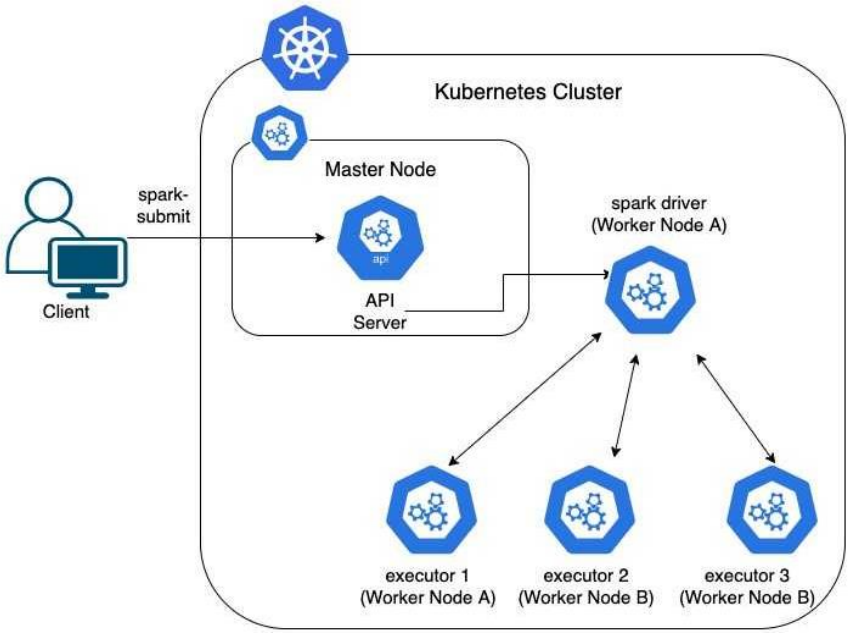

Рисунок 2.6 – Архітектура Apache Spark інтегрована у Kubernetes

У попередній архітектурі SparkContext підключався до Cluster Manager для запиту ресурсів і розподілу завдань. У новій архітектурі Spark Driver, розташований на одному з worker вузлів, взаємодіє безпосередньо з API Server Kubernetes для розподілу завдань серед executors. Executors, які раніше могли знаходитися на будь-яких доступних worker вузлах та були виділені Cluster Manager, тепер розподіляються та керуються API Server.

Архітектура у другому варіанті більше інтегрована з Kubernetes, що дозволяє використовувати всі переваги Kubernetes для управління контейнерами та

ресурсами кластеру. Це включає автоматичне масштабування, відновлення після збоїв та оркестрацію контейнерів.

2.5 Загальна послідовність розгортання кластера та інтеграції Spark-додатків

Розглянувши схеми функціонування Kubernetes кластера у гібридній хмарі та Spark-фреймворку у кластерних системах, перейдемо до складання покрокового плану для реалізації проєкту.

На рисунку 2.7 представлено загальний план виконання проєкту, що включає в себе підготовку інструментів, середовища та ролей для розгортання кластера, написання Terraform-скриптів, перевірку стану кластера, написання додатків на Scala з використанням фреймворку Apache Spark, створення контейнерів, написання деплойментів та інші етапи.

Перший крок налаштування середовища включає в себе встановлення та конфігурацію всіх потрібних інструментів. До основних належать утиліти командного рядка kubectl, gcloud та aws-cli, які дозволять отримувати доступ до розгорнутого кластера після проведення авторизації у відповідні облікові записи. Інструмент docker (docker-engine) знадобиться для пакування розроблених Spark-додатків та подальшого їх розгортання.

Далі ми створюємо проєкт у GCP, що дозволить нам централізувати ресурси та керувати ними з одного місця. У AWS створюються IAM ролі та політики, необхідні для інтеграції з GCP, а також налаштовуються права доступу для сервісних облікових записів.

На локальній машині вказуються авторизаційні дані облікового запису потрібного користувача AWS, якому були надані відповідні дозволи у попередньому кроці, для забезпечення доступу до ресурсів AWS. Це необхідно для подальшого використання Terraform для управління ресурсами в AWS. Крім цього, тут також включаються необхідні сервіси в GCP, такі як Google Anthos для управління мульти-хмарними середовищами.

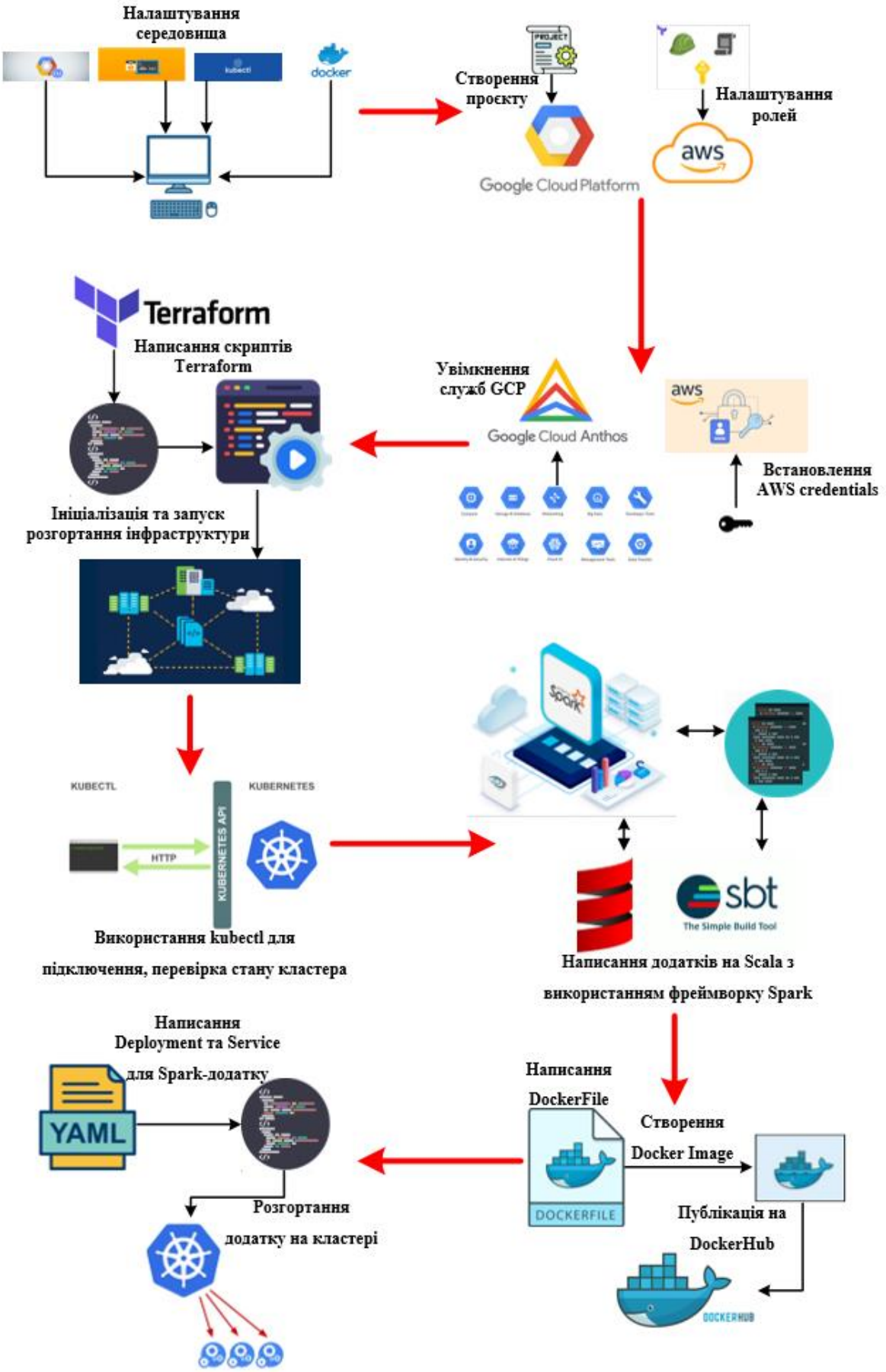

Рисунок 2.7 – Схема розгортання кластера та інтеграція Spark-додатків

Після підготовки середовища відбувається написання конфігураційних файлів Terraform для визначення інфраструктури, яку потрібно розгорнути в GCP та AWS. Сюди входить визначення кластеру Kubernetes, мереж, ролей та інших ресурсів. Далі виконується команда «terraform init» для ініціалізації робочого середовища Terraform, яка завантажує необхідні провайдери та модулі. Запускається команда «terraform apply», яка застосовує конфігураційні файли та розгортає визначені ресурси у GCP та AWS.

Після розгортання за допомогою kubectl налаштовується підключення до розгорнутого кластера Kubernetes. Виконується перевірка стану кластера та його ресурсів, щоб переконатися, що все працює коректно.

Розробляються додатки на мові програмування Scala з використанням фреймворку Apache Spark, який дозволяє обробляти великі обсяги даних. Використовується sbt (Simple Build Tool) для управління проєктом, збирання та тестування коду.

Після того, як програма готова, створюється Dockerfile для контейнеризації Spark-додатків. Dockerfile містить інструкції для створення Docker-образу, включаючи встановлення необхідних залежностей, копіювання коду та налаштування середовища виконання. За допомогою Docker створюється образ (Docker image) додатку на основі Dockerfile, який в подальшому публікується на DockerHub, що дозволяє зберігати його та використовувати для розгортання в будь-якому кластері.

Для ефективного розподілу задач на вузлах Kubernetes кластера пишуться YAML файли для Kubernetes Deployment та Service. Deployment визначає, як додаток має бути розгорнутий та масштабований у кластері, а Service визначає спосіб доступу до додатку ззовні. За допомогою команди «kubectl apply» YAML файли застосовуються до кластера, що запускає процес розгортання додатку, після чого Spark-додаток розгортається у кластері та починає обробку даних.

РОЗДІЛ 3 ПРАКТИЧНА ЧАСТИНА

3.1 Проєкти, ролі і правила для платформ GCP та AWS

Перед створенням гібридної хмари серед двох провайдерів хмарних послуг і розгортанням у ній кластера, налаштуємо самі хмарні платформи потрібним нам чином.

Для початку авторизуємося в Google Cloud та створимо новий проєкт, щоб відділити всю роботу, що стосується Big Data Kubernetes кластера від інших задач (див. рис. 3.1).

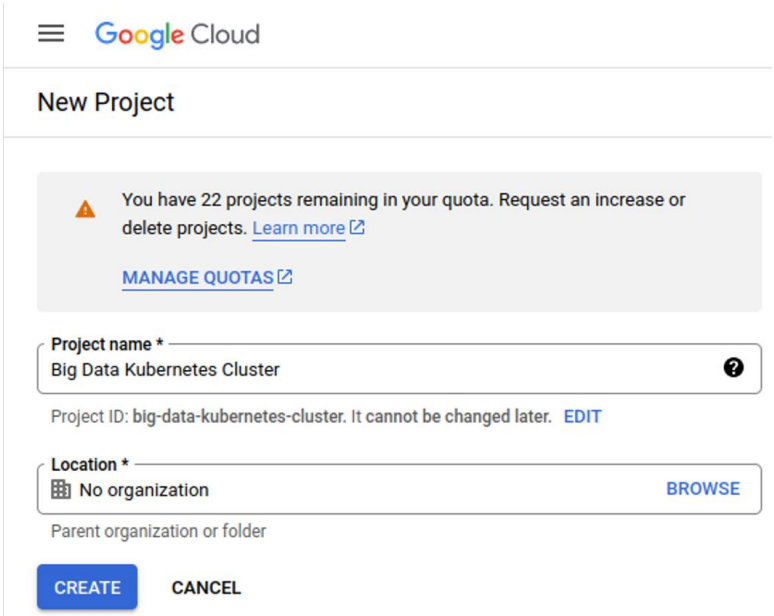

Рисунок 3.1 – Створення нового проєкту в GCP

Як вже згадувалося раніше, GCP та AWS є платформами, що надають велику кількість API сервісів, які можна вмикати або вимикати в залежності від потреб розробника або DevOps'а. Для нашого завдання необхідно увімкнути Kubernetes Engine API (або GKE API) (див. рис. 3.2).

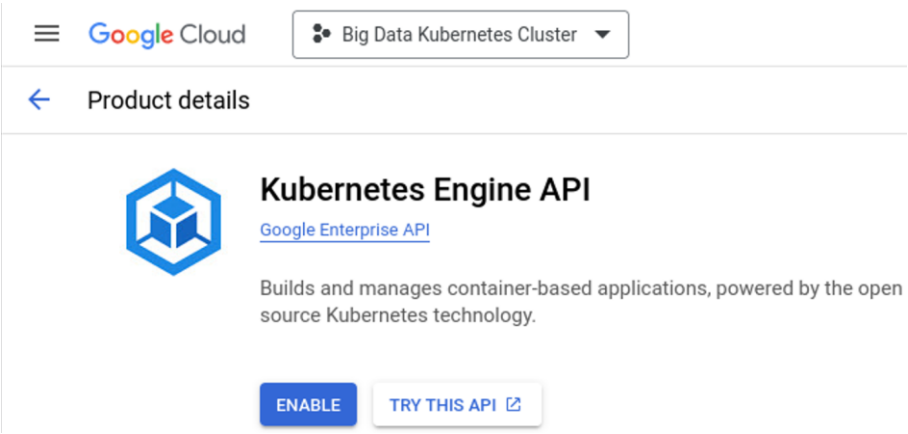

Рисунок 3.2 – Увімкнення GKE API

За замовчуванням під час реєстрації облікового запису на платформі AWS створюється профіль користувача root, який має всі права на виконання будь-яких дій. Проте, як і в багатьох UNIX-системах, у AWS пряме виконання багатьох задач root-користувачем не є рекомендованим. Тому нам необхідно створити звичайного користувача, наприклад, під своїм ім'ям і надати йому лише ті права, які є необхідними (див. рис. 3.3).

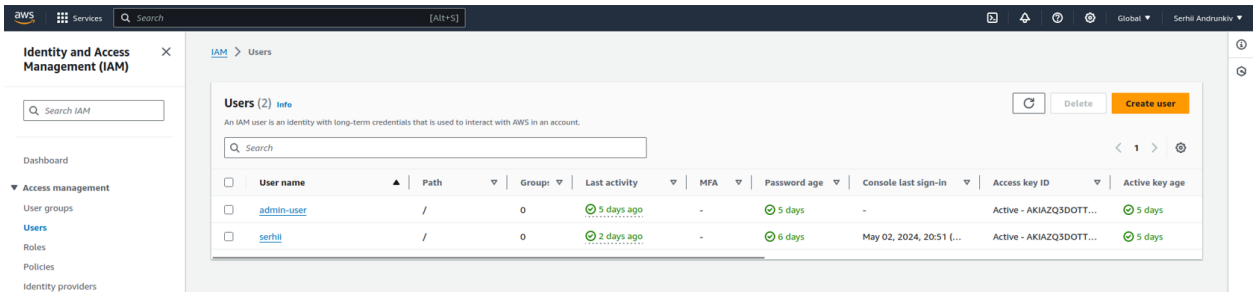

Рисунок 3.3 – Створення нового користувача у AWS

Після створення користувача варто також створити йому token для доступу до облікового запису із системи, на якій ми в подальшому проводитимемо розгортання гібридного кластера. Створені ідентифікатор доступу та секретний ключ доступу потрібно помістити у відповідний конфігураційний файл `~/.aws/credentials` у прихованій директорії (див. рис. 3.4).

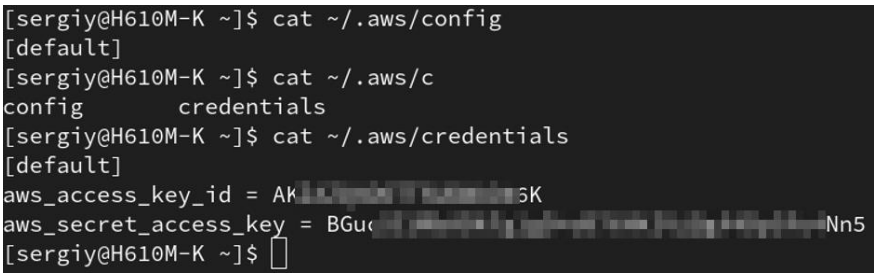

Рисунок 3.4 – Налаштування доступу до облікового запису AWS

Налаштуємо щойно створений обліковий запис в AWS. Перейдемо на вкладку Permissions і надамо всі необхідні права на використання сервісу EKS, а також деякі свої правила (див. рис. 3.5).

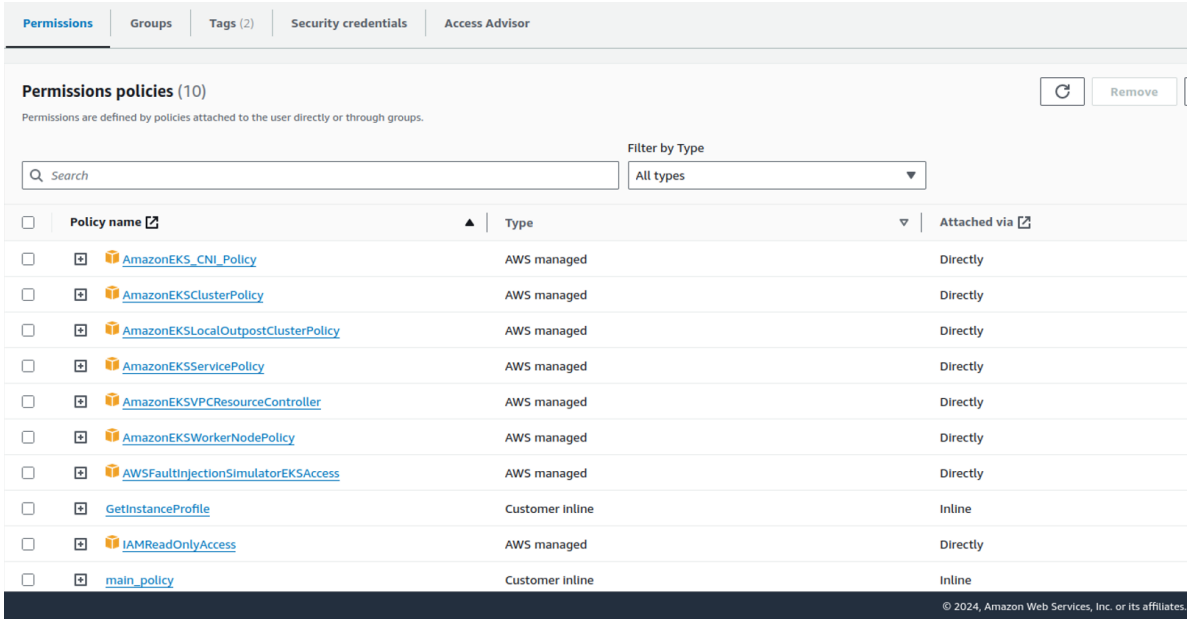

Рисунок 3.5 – Надання правил для використання EKS

Розглянемо детальніше одне з найважливіших правил, що напряму стосується використання сервісу EKS – AmazonEKSClusterPolicy (див. лістинг 3.1).

Лістинг 3.1 – Фрагмент JSON-коду правила AmazonEKSClusterPolicy

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

Продовження лістингу 3.1

```
            "Action": [
                "autoscaling:DescribeAutoScalingGroups",
                "autoscaling:UpdateAutoScalingGroup",
                "ec2:AttachVolume",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:CreateRoute",
                "ec2:CreateSecurityGroup",
                "ec2:CreateTags",
                "ec2:CreateVolume",
                "ec2:DeleteRoute",
                "ec2:DeleteSecurityGroup",
                "ec2:DeleteVolume",
                "ec2:DescribeInstances",
                ……
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName":
"elasticloadbalancing.amazonaws.com"
                }
            }
        }
    ]
}
```

Наведені вище дозволи включають ряд дій, які можуть виконуватися для керування різними аспектами інфраструктури, такими як автомасштабування, мережеві налаштування та керування ресурсами еластичного балансування навантаження.

Головні аспекти даного правила:

– `"Effect": "Allow"` означає, що всі вказані дії в даному Statement є дозволеними для виконання;

– поле `"Action"` містить перераховані дії включають в себе керування екземплярами EC2, об'єктами EBS, мережевими ресурсами, а також дії з

налаштуваннями та керуванням ресурсами Elastic Load Balancer та іншими діями, пов'язаними з кластерами EKS;

– `"Resource": "*"` вказує, що дозволені всі ресурси, що відповідають вказаним діям.

У цьому правилі також вказано додаткову умову, яка дозволяє виконання дії `"iam:CreateServiceLinkedRole"` тільки тоді, коли `"iam:AWSServiceName"` рівне `"elasticloadbalancing.amazonaws.com"`. Це дозволяє створювати ролі служб, необхідні для інтеграції Amazon EKS із сервісом Elastic Load Balancer.

3.2 Розгортання кластера у гібридній хмарі

Всі необхідні вимоги на хмарних платформах налаштовані, тому можемо переходити до розгортання Kubernetes-кластера у гібридній хмарі згідно тої моделі, яка розглядалася у проєктній частині.

Перейдемо в термінал на головній системі та переконаємося, що встановлена остання версія GCP SDK (див. рис. 3.6).

```
[sergiy@H610M-K ~]$ gcloud version
Google Cloud SDK 473.0.0
alpha 2024.04.19
beta 2024.04.19
bq 2.1.4
bundled-python3-unix 3.11.8
core 2024.04.19
gcloud-crc32c 1.0.0
gke-gcloud-auth-plugin 0.5.8
gsutil 5.27
[sergiy@H610M-K ~]$
```

Рисунок 3.6 – Перевірка версії GCP

Авторизуємося через термінал у GCP обліковий запис через, щоб мати змогу керувати хмарною платформою без веб-інтерфейсу (див. рис. 3.7).

Рисунок 3.7 – Авторизація у GCP через термінал

Далі налаштуємо змінну з ідентифікатором проєкту, який був створений у попередньому підрозділі та виконаємо команду `gcloud config set project "${PROJECT_ID}"`, щоб перейти в його контекст (див. рис. 3.8).

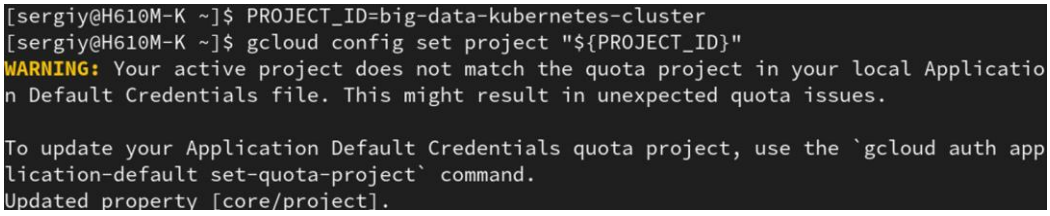

Рисунок 3.8 – Зміна контексту проєкту в GCP SDK

У попередньому підрозділі ми увімкнули сервіс GKE API у GCP, щоб мати змогу створювати Kubernetes-кластери. Для того, аби мати змогу розгорнути кластер та керувати ним через інтерфейс GCP, потрібно увімкнути ще декілька сервісів, які безпосередньо стосуються мультрихмарності, інструменту Google Anthos та моніторингу (див. рис. 3.9).

Розглянемо детальніше, для чого вони потрібні:

– gkemulticloud.googleapis.com – ключовий сервіс, що дозволяє створювати, керувати та розгортати кластери Kubernetes у сторонніх гібридних хмарах (в нашому випадку AWS);

– gkeconnect.googleapis.com – з’єднує кластери Kubernetes на різних платформах і керує ними з Anthos у GCP;

– connectgateway.googleapis.com – створює безпечні та конфіденційні тунелі між кластерами, що працюють на різних платформах;

– cloudresourcemanager.googleapis.com – керує ресурсами в GCP, такими як проекти, облікові записи користувачів та інші ресурси, які можуть бути використані для організації і керування Anthos та іншими сервісами GCP;

– anthos.googleapis.com – представляє власне сам Anthos, який є платформою для управління розподіленими середовищами контейнерів;

– logging.googleapis.com та monitoring.googleapis.com – збирають системні журнали та здійснюють моніторинг ресурсів у GCP.

```
[sergiy@H610M-K ~]$ gcloud --project="${PROJECT_ID}" services enable \
gkemulticloud.googleapis.com \
gkeconnect.googleapis.com \
connectgateway.googleapis.com \
cloudresourcemanager.googleapis.com \
anthos.googleapis.com \
logging.googleapis.com \
monitoring.googleapis.com
Operation "operations/acat.p2-649968390089-e31e143b-fbc4-47d3-a98d-1b542097bb40" finis
hed successfully.
```

Рисунок 3.9 – Увімкнення додаткових сервісів GCP

Створимо новий каталог у домашній директорії та склонуємо репозиторій з GitHub із шаблонами для Google Anthos. В цій директорії будуть зберігатися наші конфігураційні файли та скрипти Terraform для розгортання кластера у гібридній хмарі GCP+AWS (див. рис. 3.10).

```
[sergiy@H610M-K ~]$ mkdir k8s_cluster_in_multi_cloud && cd k8s_cluster_in_multi_cloud
[sergiy@H610M-K k8s_cluster_in_multi_cloud]$ git clone https://github.com/GoogleCloudP
latform/anthos-samples.git
Cloning into 'anthos-samples'...
remote: Enumerating objects: 4584, done.
remote: Counting objects: 100% (369/369), done.
remote: Compressing objects: 100% (246/246), done.
remote: Total 4584 (delta 206), reused 248 (delta 121), pack-reused 4215
Receiving objects: 100% (4584/4584), 5.85 MiB | 3.79 MiB/s, done.
Resolving deltas: 100% (2775/2775), done.
[sergiy@H610M-K k8s_cluster_in_multi_cloud]$ cd anthos-samples/anthos-multi-cloud/AWS
```

Рисунок 3.10 – Клонування репозиторію із шаблонами Anthos

Відкриємо конфігураційний файл `terraform.tfvars`, в якому відредагуємо ідентифікатор проєкту в якому буде розгортатися кластер, логін облікового запису та унікальну назву для нашого кластера (див. рис. 3.11).

Змінним `node_pool_instance_type` і `control_plane_instance_type` присвоїмо значення t3.medium, які в свою чергу репрезентують екземпляри віртуальних машин (EC2 Instances) із кількістю віртуальних CPU рівною 2 та об'ємом RAM рівним 4 ГБ. Дивлячись та таблицю 3.1, можна побачити, що t3.medium є вигідним варіантом, оскільки ціна за годину для нас є досить помірною, що робить його привабливим варіантом для більшості застосувань.

Рисунок 3.11 – Редагування terraform.tfvars

Таблиця 3.1 – Порівняння Amazon EC2 екземплярів

Назва	ЦП	Пам'ять (ГБ)	Базовий рівень продуктивності / Віртуальні CPU	Отримання кредитів CPU/година	Підвищена пропускна здатність EBS (Мбіт/с)	Ціна за годину по запиту*
t3.nano	2	0,5	5%	6	До 2085	0,0052 USD
t3.micro	2	1,0	10%	12	До 2085	0,0104 USD
t3.small	2	2,0	20%	24	До 2085	0,0209 USD
t3.medium	2	4,0	20%	24	До 2085	0,0418 USD
t3.large	2	8,0	30%	36	До 2780	0,0835 USD
t3.xlarge	4	16,0	40%	96	До 2780	0,1670 USD

Версію Kubernetes-кластера вказуємо не вище 1.27.10-gke.500, оскільки в іншому випадку роль агента служби Multi-Cloud API буде вимагати дозволу iam:GetInstanceProfile у проєкті AWS.

Останніми налаштовуються регіональні зони. Вибір us-east4 для GCP і us-east-1 для AWS буде доречним, оскільки ці зони розташовані в одному географічному регіоні, але представлені різними хмарними платформами. Щодо підмереж, вибір зон us-east-1a, us-east-1b та us-east-1c у регіоні AWS us-east-1 також буде непоганим, оскільки це дозволить розподілити ресурси кластера для забезпечення високої доступності та стійкості.

Базове налаштування гібридного кластера завершено. Тепер варто приділити увагу декларативним скриптам Terraform для розгортання кластера. У директорії `~/k8s_cluster_in_multi_cloud\anthos-samples\anthos-multi-cloud\AWS` розміщено головний конфігураційний файл main.tf, в якому можна прописати загальні налаштування кластера. Наприклад, його перші рядки містять код генерації випадкового суфікса для імені кластера, щоб уникнути банальної ситуації однакових імен (див. лістинг 3.2).

Лістинг 3.2 – Фрагмент Terraform-коду генерації суфіксу імені

```
locals {
  name_prefix = "${var.name_prefix}-${random_string.suffix.result}"
}
resource "random_string" "suffix" {
  length    = 2
  special   = false
  lower     = true
  min_lower = 2
}
```

В самій же директорії `~/k8s_cluster_in_multi_cloud\anthos-samples\anthos-multi-cloud\AWS` знаходиться каталог modules, вміст якого у раніше згаданому головному файлі репрезентується як логічні модулі для організації коду та виконання різних завдань. Наприклад, фрагмент, що представлено у лістингу 3.3 являє собою підключення модулів налаштування VPC та розгортання кластера Anthos на GCP та AWS. В кожній такій сутності є ключ source із значенням директорії до Terraform-файлу відповідного модуля.

Лістинг 3.3 – Фрагмент Terraform-коду з підключеними модулями

```
module "vpc" {
  source                          = "./modules/vpc"
  aws_region                      = var.aws_region
  vpc_cidr_block                  = var.vpc_cidr_block
  anthos_prefix                   = local.name_prefix
  subnet_availability_zones       = var.subnet_availability_zones
  public_subnet_cidr_block        = var.public_subnet_cidr_block
  cp_private_subnet_cidr_blocks = var.cp_private_subnet_cidr_blocks
  np_private_subnet_cidr_blocks = var.np_private_subnet_cidr_blocks
}

module "anthos_cluster" {
  source                                          =
"./modules/anthos_cluster"
  anthos_prefix                                   =
local.name_prefix
  location                                        =
var.gcp_location
  aws_region                                      = var.aws_region
  cluster_version                                 =
coalesce(var.cluster_version, module.gcp_data.latest_version)
  database_encryption_kms_key_arn                 =
module.kms.database_encryption_kms_key_arn
  control_plane_config_encryption_kms_key_arn     =
module.kms.control_plane_config_encryption_kms_key_arn
……
```

Одним із найважливіших таких модулів є `./modules/anthos_cluster/main.tf`, в якому ми редагуємо детальні налаштування гібридного кластера. У лістингу 3.4 можна побачити фрагмент ресурсу `google_container_aws_cluster`, в якому знаходиться вже знайомий вміст – параметри для встановлення регіону, зон, імені кластера, його опису та динамічний блок визначення прав доступу до кластера. Більшість із значень цих параметрів ми встановили у конфігураційному файлі `terraform.tfvars`.

Лістинг 3.4 – Фрагмент Terraform-коду налаштування кластера

```
resource "google_container_aws_cluster" "this" {
  aws_region  = var.aws_region
  description = "Hybrid GCP+AWS Big Data Cluster"
  location    = var.location
  name        = var.anthos_prefix
  authorization {
```

Продовження лістингу 3.4

```
    dynamic "admin_users" {
      for_each = var.admin_users

      content {
        username = admin_users.value
      }
    }
  }
    ……
```

Переважна більшість Kubernetes кластерів мають дві основні групи компонентів – Control Plane і Node Pool. Control plane відповідає за керування кластером Kubernetes. Він містить різні компоненти, такі як API сервер, контролери, планувальник і диспетчери. Ці компоненти забезпечують функціональність Kubernetes, таку як розподіл роботи між вузлами, розгортання контейнерів, моніторинг та балансування навантаження. Зазвичай він має свої власні вимоги до ресурсів і конфігурується окремо від інших частин кластера.

У лістинг 3.5 представлено блок налаштування Control Plane для нашого кластера, де важливо виділити вже знайомі параметри `iam_instance_profile`, `instance_type` та `version` і тег імені, який формується із додаванням відповідного суфіксу. Блоки `main_volume` і `root_volume` визначають обсяги, тип і шифрування для головного і кореневого дисків відповідно.

Лістинг 3.5 – Блок налаштування Control Plane

```
control_plane {
    iam_instance_profile = var.control_plane_iam_instance_profile
    instance_type        = var.control_plane_instance_type
    subnet_ids           = var.subnet_ids
    tags = {
      "Name" : "${var.anthos_prefix}-control-plain"
    }
    version = var.cluster_version
    aws_services_authentication {
      role_arn = var.role_arn
    }
    config_encryption {
      kms_key_arn = var.control_plane_config_encryption_kms_key_arn
    }
```

Продовження лістингу 3.5

```
    database_encryption {
      kms_key_arn = var.database_encryption_kms_key_arn
    }
    main_volume {
      size_gib    = 50
      volume_type = "GP3"
      iops        = 3000
      kms_key_arn                                                    =
var.control_plane_main_volume_encryption_kms_key_arn
    }
    root_volume {
      size_gib    = 50
      volume_type = "GP3"
      iops        = 3000
      kms_key_arn                                                    =
var.control_plane_root_volume_encryption_kms_key_arn
    }
  }
```

І насамкінець, сформуємо блок ресурсу для групи робочих вузлів кластера – Node Pool. У лістингу 3.6 представлено даний ресурс, з якого варто виділити такі блоки, як autoscaling, в якому ми задаємо достатню для нас кількість мінімальної на максимальної кількості вузлів; config, де прописано налаштування шифрування, типи екземплярів вузлів та параметри головного і кореневого дисків; timeouts, який відповідає за максимальний допустимий час для створення, модифікації та видалення групи вузлів.

Лістинг 3.6 – Terraform-код ресурсу групи вузлів гібридного кластера

```
resource "google_container_aws_node_pool" "this" {
  name      = "${var.anthos_prefix}-node-pool"
  cluster   = google_container_aws_cluster.this.id
  subnet_id = var.node_pool_subnet_id
  version   = var.cluster_version
  location  = google_container_aws_cluster.this.location
  max_pods_constraint {
    max_pods_per_node = 110
  }
  autoscaling {
    min_node_count = 5
    max_node_count = 10
  }
```

Продовження лістингу 3.6

```
config {
  config_encryption {
    kms_key_arn = var.node_pool_config_encryption_kms_key_arn
  }
  instance_type        = var.node_pool_instance_type
  iam_instance_profile = var.node_pool_iam_instance_profile
  root_volume {
    size_gib    = 30
    volume_type = "GP3"
    iops        = 3000
    kms_key_arn = var.node_pool_root_volume_encryption_kms_key_arn
  }
  tags = {
    "Name" : "${var.anthos_prefix}-node-pool"
  }
}
timeouts {
  create = "45m"
  update = "45m"
  delete = "45m"
}}
```

Після завершення формування та редагування Terraform-скриптів, можемо переходити до запуску розгортання самого кластера ввівши команду `terraform init` (див. рис. 3.12). В результаті її виконання у локальний кеш буде збережено дані провайдерів використовуваних хмарних послуг, перевірено та завантажено модулі для їх для використання у нашому проєкті.

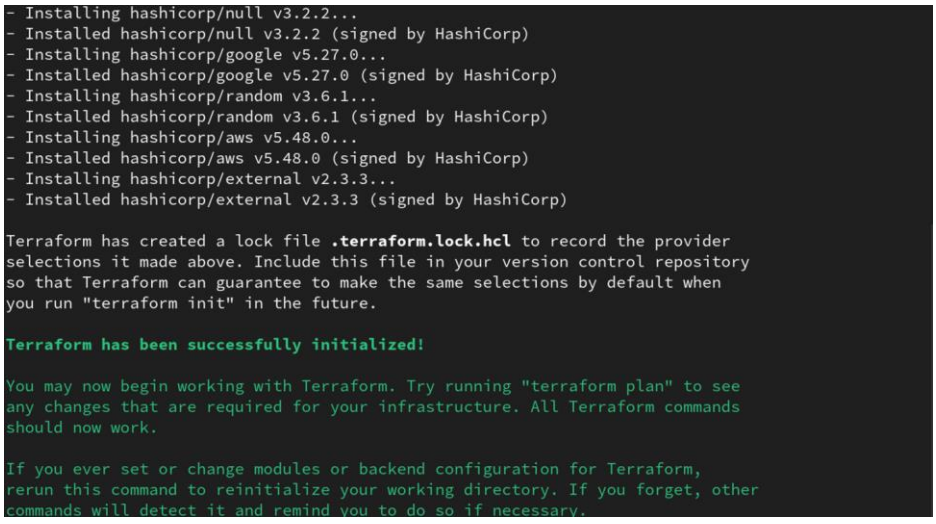

Рисунок 3.12 – Ініціалізація робочого середовища Terraform

Далі запустимо команду `terraform apply`, щоб розпочати розгортання Kubernetes-кластера (див. рис. 3.13). Дана процедура запустить аналіз конфігураційного файлу, визначить, які зміни потрібно внести до інфраструктури та надішле запити до провайдера AWS.

```
[sergiy@H610M-K AWS]$ terraform apply
module.create_vars.data.external.env_override[0]: Reading...
module.create_vars.data.external.env_override[0]: Read complete after 0s [id=-]
module.gcp_data.data.google_container_aws_versions.this: Reading...
module.gcp_data.data.google_project.project: Reading...
module.iam.data.aws_iam_policy_document.cp_assume_role_policy_document: Reading...
module.iam.data.aws_iam_policy_document.np_assume_role_policy_document: Reading...
module.kms.data.aws_caller_identity.current: Reading...
module.iam.data.aws_iam_policy_document.cp_assume_role_policy_document: Read complete after 0s [i
d=2851119427]
module.iam.data.aws_iam_policy_document.np_assume_role_policy_document: Read complete after 0s [i
d=2851119427]
module.kms.data.aws_caller_identity.current: Read complete after 0s [id=654654217445]
module.kms.data.aws_iam_policy_document.root_volume_encryption_policy_document: Reading...
module.kms.data.aws_iam_policy_document.root_volume_encryption_policy_document: Read complete aft
er 1s [id=18347800]
module.gcp_data.data.google_container_aws_versions.this: Read complete after 1s [id=2024-05-04 20
:22:12.218339927 +0000 UTC]
module.gcp_data.data.google_project.project: Read complete after 2s [id=projects/big-data-kuberne
tes-cluster]
module.iam.data.aws_iam_policy_document.api_assume_role_policy_document: Reading...
```

Рисунок 3.13 – Запуск розгортання кластера у гібридній хмарі

Через деякий час (приблизно 12-15 хвилин) гібридний Kubernetes кластер буде успішно створено. Налаштуємо контекст за допомогою команди на рисунку 3.14. Це дозволить нам використовувати шлюз GKE Connect як кінцеву точку і в будь-який час отримувати доступ до кластера з основної системи.

```
[sergiy@H610M-K AWS]$ gcloud container hub memberships get-credentials big-data-k8s-cluster-kq
Starting to build Gateway kubeconfig...
Current project_id: big-data-kubernetes-cluster
A new kubeconfig entry "connectgateway_big-data-kubernetes-cluster_global_big-data-k8s-cluster-kq
" has been generated and set as the current context.
```

Рисунок 3.14 – Налаштування контексту Kubernetes-кластера

Щоб переконатися в успішності роботи кластера, запустимо команду `kubectl get nodes` та побачимо 5 робочих вузлів у стані “Ready” (див. рис. 3.15).

```
NAME              STATUS   ROLES    AGE   VERSION
ip-10-0-1-161     Ready    <none>   47h   v1.27.10-gke.500
ip-10-0-1-172     Ready    <none>   46h   v1.27.10-gke.500
ip-10-0-1-18      Ready    <none>   46h   v1.27.10-gke.500
ip-10-0-1-26      Ready    <none>   47h   v1.27.10-gke.500
ip-10-0-1-80      Ready    <none>   46h   v1.27.10-gke.500
```

Рисунок 3.15 – Список вузлів гібридного кластера

У веб-інтерфейсі Google Cloud Console можна спостерігати щойно розгорнутий кластер (див. рис. 3.16). Тут явно можна побачити, що кластер має тип «AWS», що репрезентує його як гібридний кластер.

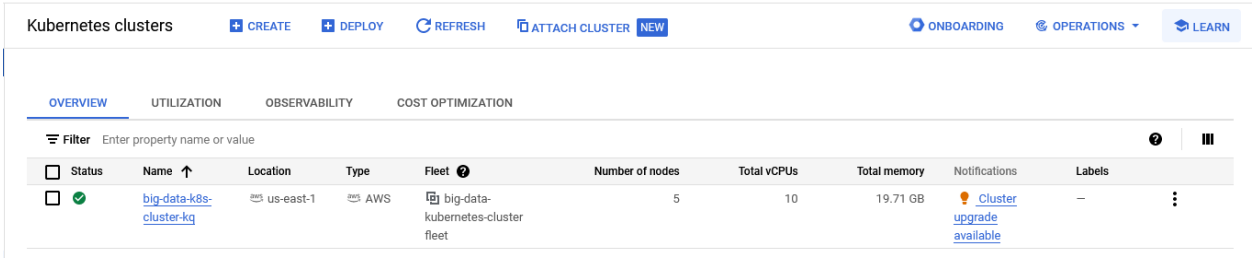

Рисунок 3.16 – Розгорнутий кластер в Google Cloud Console

3.3 Створення та налаштування сховища даних GCS

Google Cloud Storage є масштабованим і надійним об’єктним сховищем, яке використовується для зберігання даних різного типу та обсягу. Це сховище підтримує широкий спектр сценаріїв використання, включаючи зберігання резервних копій, архівів, розподілених обчислень та аналітики. У цьому розділі описано процес створення та налаштування сховища даних GCS для потреб Kubernetes-кластера.

У веб-інтерфейсі GCP перейдемо до розділу «Cloud Storage» та створимо новий Bucket (див. рис. 3.17).

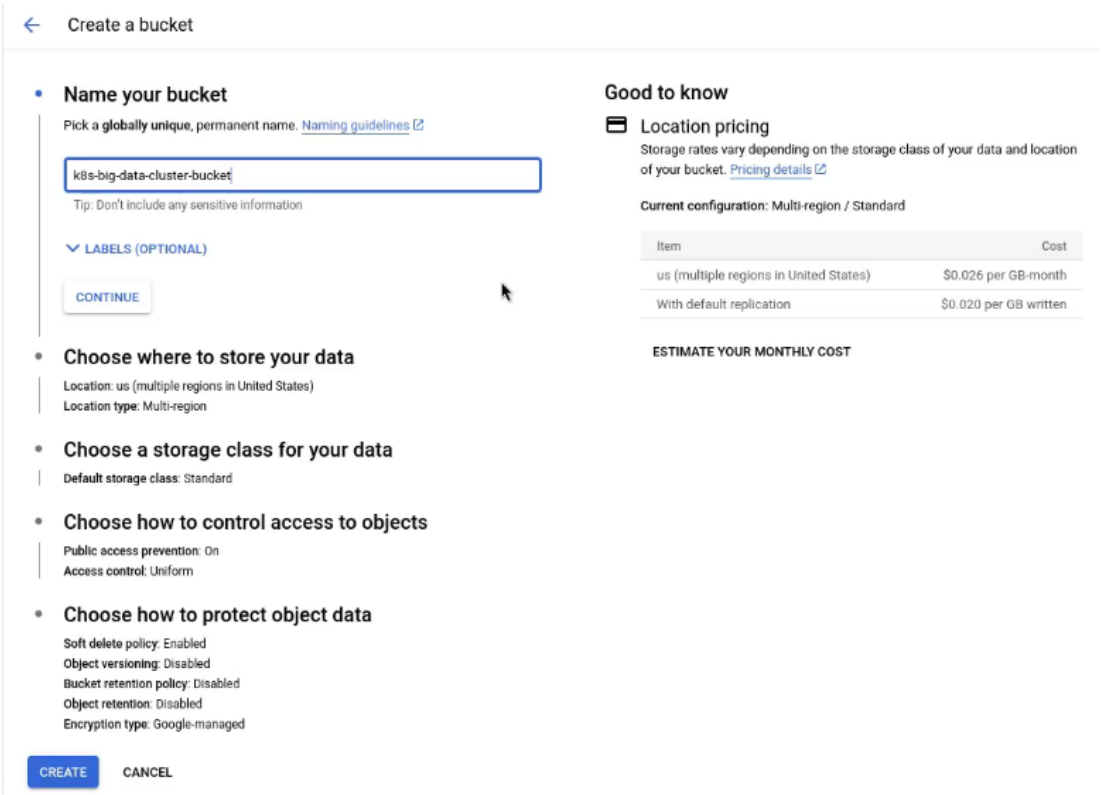

Рисунок 3.17 – Створення сховища даних GCS

В самому сховищі на вкладці Objects можемо керувати файловою системою та структурою каталогів. Завантажимо туди файл із вхідними даними для нашого додатку, який будемо розробляти в наступному розділі (див. рис. 3.18).

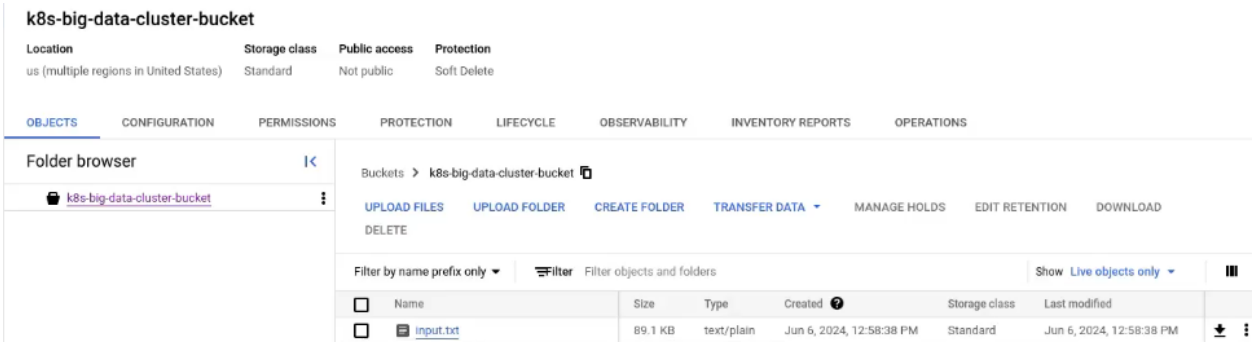

Рисунок 3.18 – Завантаження файлу із вхідними даними у сховище

Для доступу до GCS з Kubernetes, додаток Spark у контейнері повинен мати відповідні права доступу. Для цього використовується сервісний обліковий запис GCP з необхідними правами доступу до сховища GCS.

На рисунку 3.19 представлено створення сервісного облікового запису у GCP і надання йому ролі «Storage Object Viewer».

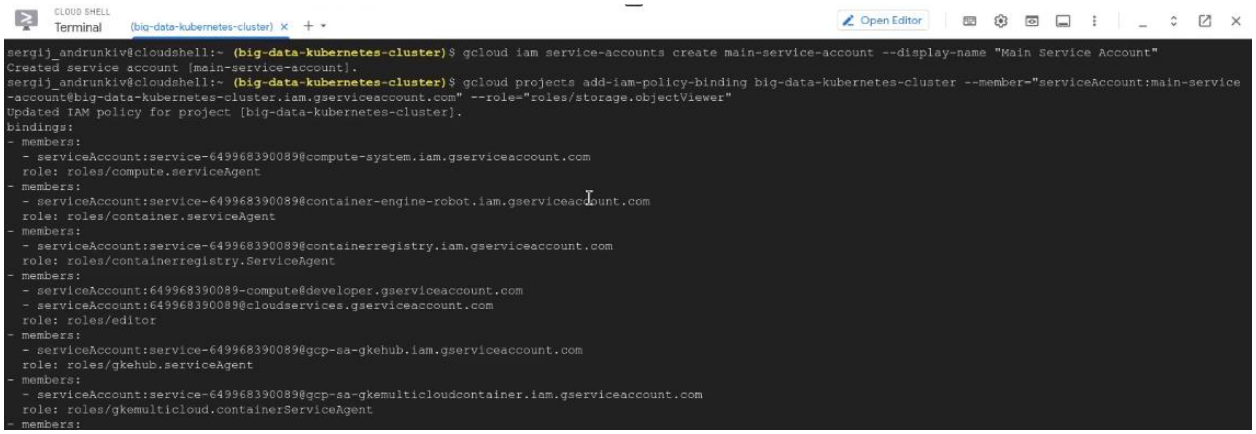

Рисунок 3.19 – Створення сервісного облікового запису GCP

Для захищеного доступу та ідентифікації об’єктного сховища майбутнім додатком на кластері, потрібно згенерувати JSON-ключ для сервісного облікового запису та зберегти його (див. рис. 3.20).

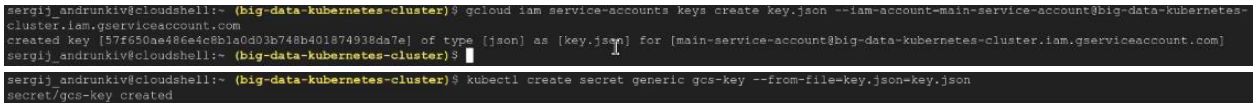

Рисунок 3.20 – Генерування JSON-ключа

3.4 Створення Spark-додатків та формування образів Docker

Як згадувалося раніше, Apache Spark – це потужний фреймворк для обробки та аналізу великих обсягів даних у розподіленому середовищі. Він надає широкі можливості для обробки даних в реальному часі, машинного навчання, аналізу графів та інших завдань, що стосуються Big Data.

Пакування Spark-додатків у Docker-контейнери дозволить стандартизувати середовище виконання та забезпечить їх переносимість між різними

обчислювальними середовищами. Контейнери Docker містять усі необхідні залежності та конфігурації, що робить їх легкими для розгортання та управління.

Для того, щоб перевірити кластер на працездатність та визначитися із мінімальними вимогами ресурсів необхідних для фреймворку Spark, напишемо, зберемо та запакуємо в образ тестову програму на мові Scala з використанням Spark, яка буде підраховувати із доволі великого вхідного файлу кількості повторень кожного слова. Створимо в підкаталозі `src/main/scala` файл Spark-програми (див. лістинг 3.7).

Лістинг 3.7 – Програма на мові Scala для підрахунку всіх слів

```
import org.apache.spark.sql.SparkSession
import org.apache.hadoop.fs.Path

object WordCount {
  def main(args: Array[String]): Unit = {
    val spark = SparkSession.builder()
      .appName("Word Count")
      .getOrCreate()

    spark.conf.set("fs.gs.impl",
"com.google.cloud.hadoop.fs.gcs.GoogleHadoopFileSystem")
    spark.conf.set("fs.AbstractFileSystem.gs.impl",
"com.google.cloud.hadoop.fs.gcs.GoogleHadoopFS")
    spark.conf.set("fs.gs.project.id", "k8s-big-data-cluster")
    spark.conf.set("google.cloud.auth.service.account.enable",
"true")
    spark.conf.set("google.cloud.auth.service.account.json.keyfile",
"key.json")
    val gcsFilePath = "gs://k8s-big-data-cluster-bucket/input.txt"
    // завантаження вхідного файлу з GCS
    val lines = spark.sparkContext.textFile(gcsFilePath)
    // розділ кожного рядку на слова та їх підрахунок
    val wordCounts = lines.flatMap(line => line.split(" "))
                          .map(word => (word, 1))
                          .reduceByKey(_ + _)
    // вивід результатів
    wordCounts.collect().foreach(println)
    // Очікуємо завершення SparkSession
    spark.streams.awaitAnyTermination()
    // завершення SparkSession
    spark.stop()
  }
}
```

Створимо файл build.sbt в корені проєкту, в якому опишемо, згідно вимог: правила компіляції та збірки програми, вказавши назву проєкту, версію програми та scala, а також зовнішні бібліотеки (див. рис. 3.21).

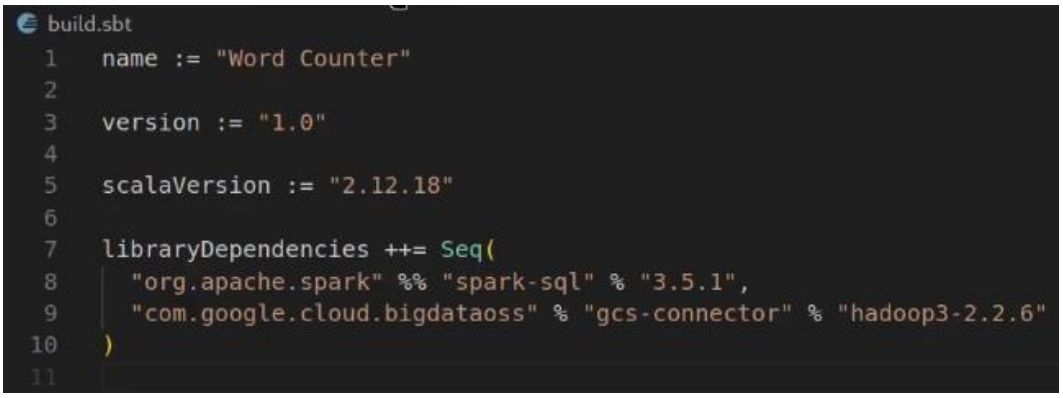

Рисунок 3.21 – Файл з правилами для збирання програми

Фрагмент вхідного файлу із текстом, який містить близько декількох десятків тисяч слів з повтореннями представлено на рисунку 3.22.

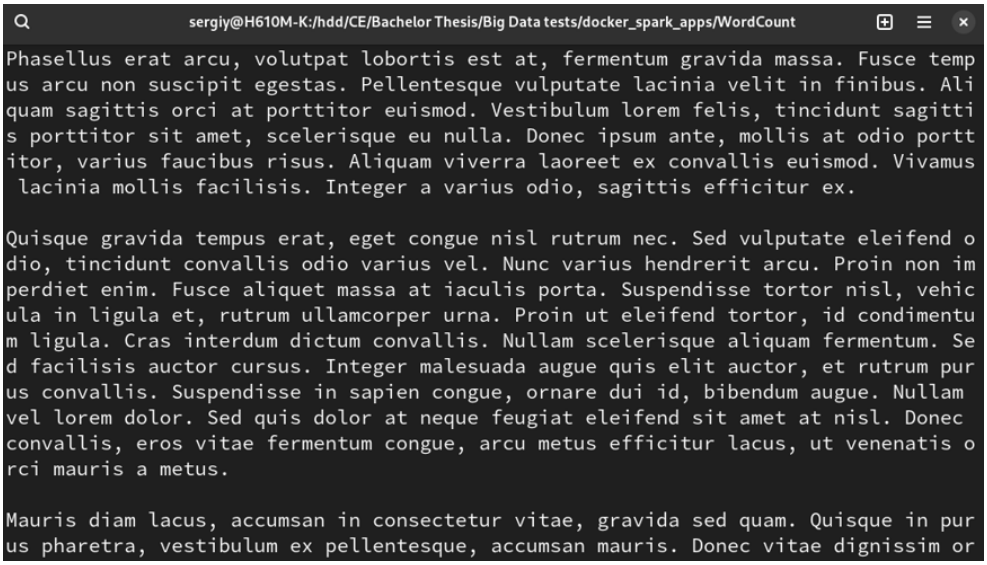

Рисунок 3.22 – Вхідний файл для Spark-програми

За допомогою утиліти sbt package зберемо нашу програму. При її запуску відбувається автоматичний пошук файлу build.sbt в поточному каталозі, який допомагає в компіляції програми (див. рис. 3.23).

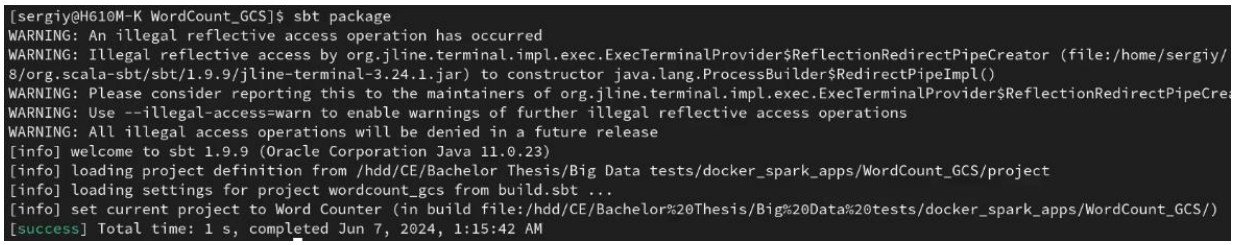

Рисунок 3.23 – Збирання Spark-програми

Перейдемо до пакування програми у Docker-образ. Створимо файл Dockerfile, в якому опишемо правила створення самого контейнера з образу, а також подальші дії в ньому, такі як запуск потрібних команд, прокидання портів тощо (див. рис. 3.24). В нашому випадку, першою командою є вибір основного шару (базового образу) Spark для нашого образу, оскільки програма написана з використанням фреймворку Spark. Після цього копіюємо всі потрібні для виконання програми файли (включно із вхідним) у каталог `/opt/bitnami/spark` в Docker-образі. Забезпечимо прокидання порту 4040 ззовні контейнера, щоб отримати доступ до веб-інтерфейсу Spark на цільовій машині та запустимо JAR-файл програми з використанням spark-submit, передавши як аргумент вхідний файл з даними.

```
2  FROM bitnami/spark:latest
3  # Копіюємо JAR-файл програми у контейнер
4  COPY target/scala-2.12/word-counter_2.12-1.0.jar /opt/bitnami/spark/wordcount.jar
5  # Копіюємо файл автентифікації у контейнер
6  COPY key.json /opt/bitnami/spark/key.json
7  # Копіюємо GCS Connector JAR
8  COPY gcs-connector-hadoop3-latest.jar /opt/bitnami/spark/jars/
   gcs-connector-hadoop3-latest.jar
9  # Встановлюємо необхідні змінні середовища
10 ENV GOOGLE_APPLICATION_CREDENTIALS=/opt/bitnami/spark/key.json
11 EXPOSE 4040
12 # Команда для запуску програми при старті контейнера
13 CMD ["spark-submit", "--class", "WordCount", "--jars", "/opt/bitnami/spark/jars/
   gcs-connector-hadoop3-latest.jar", "/opt/bitnami/spark/wordcount.jar"]
```

Рисунок 3.24 – Вміст Dockerfile

Сформуємо Docker-образ з використанням команди `docker build` та конфігураційного файлу Dockerfile. Успішне послідовне виконання всіх команд із Dockerfile і, як результат, створення Docker-образу можна побачити на рисунку 3.25.

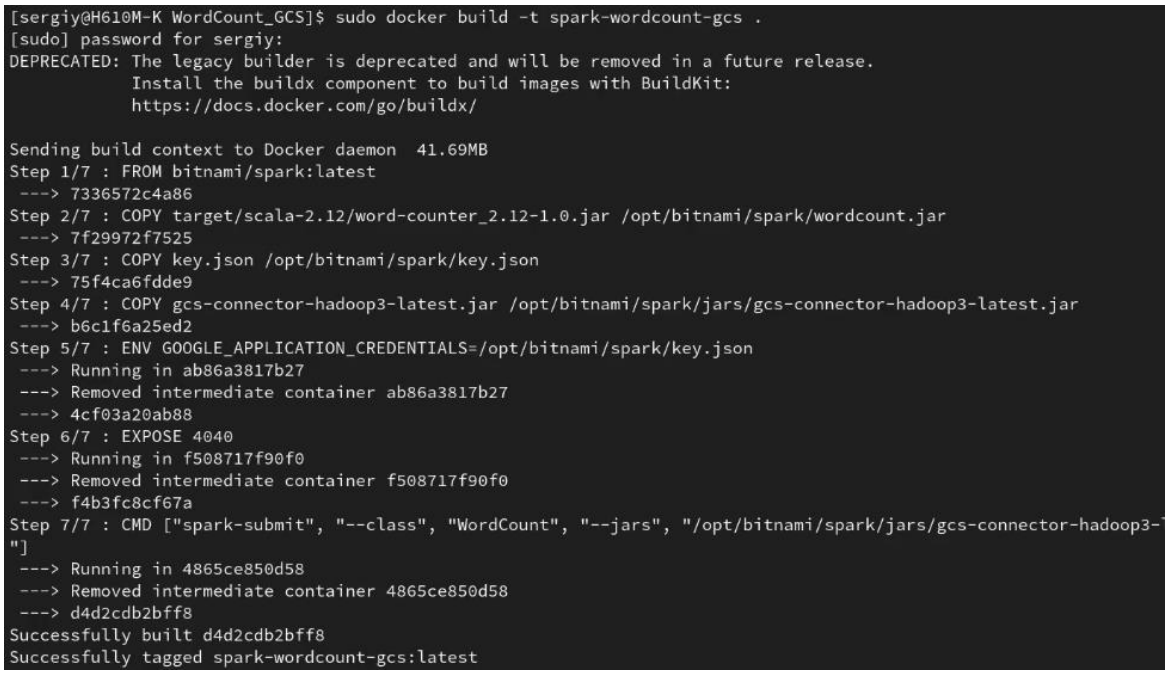

Рисунок 3.25 – Створення Docker-образу

Переглянемо поточні Docker-образи і переконаємося, що він є у списку. Після цього додамо до нього тег та опублікуємо на DockerHub за допомогою каманди `docker push`, що в майбутньому його можна було звідти дістати і розгорнути у гібридному кластері (див. рис. 3.26).

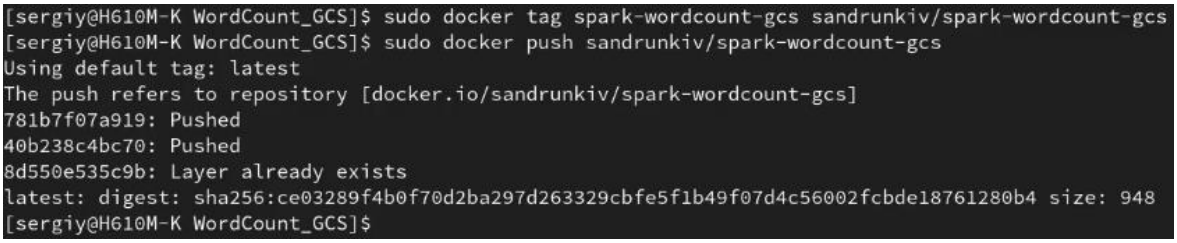

Рисунок 3.26 – Публікація образу на DockerHub

Відкриємо браузер і авторизуємося на сайті DockerHub в той акаунт, який є підключений до локальної утиліти docker (див. рис. 3.27). В розділі “Repositories” видно щойно створений Docker-образ з детальною інформацією про нього, під яку систему він зібраний, як його розгорнути і тд.

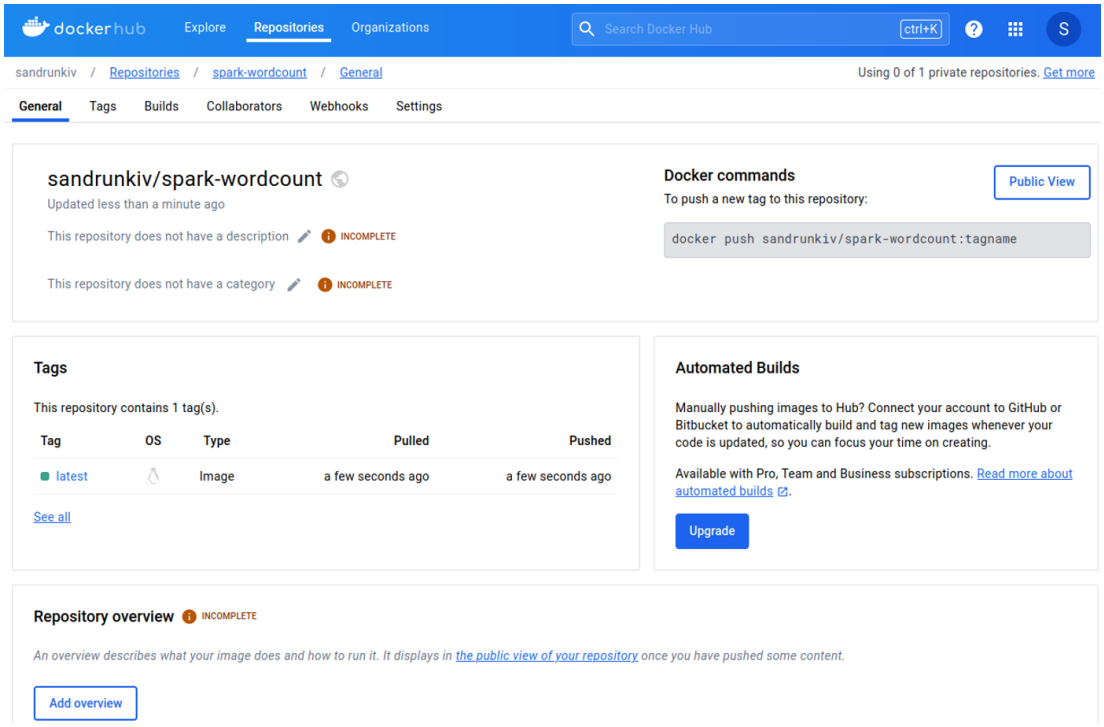

Рисунок 3.27 – Інформація про Docker-образ на DockerHub

3.5 Розгортання і запуск Spark додатків у гібридному кластері

Гібридний Kubernetes кластер готовий, тепер спробуємо розгорнути на ньому фреймворк Spark за допомогою менеджера Helm, а також створити розгортання попередньо створеного нами Docker-образу.

З використанням утиліти curl завантажимо скрипт встановлення менеджера Helm, надамо відповідні права на виконання за допомогою chmod, встановимо його і додамо репозиторій bitnami (див. рис. 3.28).

```
root@node1:~# curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
root@node1:~# chmod 700 get_helm.sh
root@node1:~# ./get_helm.sh
Downloading https://get.helm.sh/helm-v3.14.4-linux-amd64.tar.gz
Verifying checksum... Done.
Preparing to install helm into /usr/local/bin
helm installed into /usr/local/bin/helm
root@node1:~# helm repo add bitnami-repo https://charts.bitnami.com/bitnami
channel 3: open failed: connect failed: Connection refused
"bitnami-repo" has been added to your repositories
root@node1:~#
```

Рисунок 3.28 – Встановлення менеджера Helm на кластер

Встановимо Spark із репозиторію Bitnami, після чого збільшимо кількість екземплярів даного сервісу до чотирьох (див. рис. 3.29).

```
root@node1:~# helm install spark-release bitnami-repo/spark
NAME: spark-release
LAST DEPLOYED: Sat Apr 13 12:11:16 2024
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
CHART NAME: spark
CHART VERSION: 9.0.1
APP VERSION: 3.5.1
```

```
root@node1:~# helm upgrade spark-release bitnami-repo/spark --set worker.replicaCount=4
Release "spark-release" has been upgraded. Happy Helming!
```

Рисунок 3.29 – Встановлення сервісу Spark на кластер

Виконаємо команду `kubectl get pods` і переконаємося, що всі сервіси мають статус “Running” (див. рис. 3.30).

```
NAME                    READY   STATUS    RESTARTS   AGE
spark-release-master-0  1/1     Running   0          19m
spark-release-worker-0  1/1     Running   0          19m
spark-release-worker-1  1/1     Running   0          9m30s
spark-release-worker-2  1/1     Running   0          7m17s
spark-release-worker-3  1/1     Running   0          3m56s
```

Рисунок 3.30 – Перевірка всіх екземплярів сервісу Spark

Тепер перейдемо до розгортання Docker-контейнера із Spark програмою в середовищі кластера. Для цього нам потрібно створити декларативний файл розгортання на мові YAML (див. лістинг 3.8).

Розглянемо детальніше код розгортання. На найвищому рівні ієрархії він складається з двох основних частин, які характеризує поле “kind”: Deployment і Service. Обидві частини відповідають за різні аспекти розгортання та доступу до додатку в Kubernetes-кластері.

Частина Deployment є об'єктом Kubernetes, який відповідає за керування розгортанням подів з контейнерами, забезпечуючи автоматичне оновлення, масштабування та відновлення додатків у разі збою (див. лістинг 3.8).

Лістинг 3.8 – Частина Deployment YAML-маніфесту

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: spark-wordcount-gcs-deployment
  labels:
    app: spark-wordcount-gcs
spec:
  replicas: 2
  selector:
    matchLabels:
      app: spark-wordcount-gcs
  template:
    metadata:
      labels:
        app: spark-wordcount-gcs
    spec:
      containers:
      - name: spark-wordcount-gcs
        image: sandrunkiv/spark-wordcount-gcs:latest
        ports:
        - containerPort: 4040
        env:
        - name: GOOGLE_APPLICATION_CREDENTIALS
          value: /opt/bitnami/spark/key.json
        volumeMounts:
        - name: gcs-key
          mountPath: /opt/bitnami/spark/key.json
          subPath: key.json
      volumes:
      - name: gcs-key
        secret:
          secretName: gcs-key
---
apiVersion: v1
kind: Secret
metadata:
  name: gcs-key
type: Opaque
data:
  key.json: ewogICJ0eXBlIjogInNlcnZpY2VfYWNjb3……
```

Частина Service забезпечує мережевий доступ до набору подів (див. лістинг 3.9). Вона абстрагує доступ до подів, дозволяючи користувачам і іншим сервісам взаємодіяти з додатком через стабільну IP-адресу або DNS-ім'я, незалежно від змін у розташуванні подів.

Лістинг 3.9 – Частина Service YAML-маніфесту

```
apiVersion: v1
kind: Service
metadata:
  name: spark-wordcount-gcs-service
  labels:
    app: spark-wordcount-gcs
spec:
  type: LoadBalancer
  ports:
  - port: 4040
    targetPort: 4040
  selector:
    app: spark-wordcount-gcs
```

Обидві частини мають поля apiVersion та kind, які визначають API-версії для створення ресурсів та тип самого ресурсу відповідно. Поля name і labels дозволяють надати імена та мітки ресурсам (зазвичай відповідають імені додатку, що розгортається).

Поле spec дещо відрізняється в обох ресурсах. В ресурсі Deployment дане поле описує кількість створюваних реплік (подів) даного додатку. Тут також знаходиться поле template, яке описує шаблон створення подів. Воно містить поля для метаданих подів та специфікації контейнерів. В полі специфікації вказано масив контейнерів, ім’я контейнера, публічний образ для розгортання (DockerHub) та порти, які потрібно відкрити в контейнері.

В ресурсі Service поле spec містить інформацію про тип сервісу, в даному випадку LoadBalancer, що дозволить автоматично створити зовнішній балансувальник навантаження для доступу до сервісу. Крім цього, тут вказується селектор, що вказує, які поди будуть обслуговуватися цим сервісом на основі

міток, а також внутрішній і зовнішній порт для доступу до сервісу всередині кластера та ззовні відповідно.

Таким чином ресурс Deployment створює два поди з контейнером, що використовує образ sandrunkiv/spark-wordcount:latest та забезпечує автоматичне управління цими подами, включаючи оновлення, масштабування та відновлення у разі збою. Ресурс Service створює сервіс типу LoadBalancer, який забезпечує зовнішній доступ до подів через порт 4040 та балансує навантаження між репліками, розподіляючи трафік між подами з міткою app: spark-wordcount.

Завершивши формування декларативного файлу розгортання сервісу, запустимо команду `kubectl apply -f spark_wordcount_config.yaml`, яка створить deployment та відповідний йому сервіс (див. рис. 3.31).

```
user@node1:~$ nano spark_wordcount_config.yaml
user@node1:~$ kubectl apply -f spark_wordcount_config.yaml
deployment.apps/spark-wordcount-deployment created
service/spark-wordcount-service unchanged
```

Рисунок 3.31 – Створення deployment і сервісу Spark-додатку

Запустивши команду kubectl get all, можемо побачити поди нашого deployment, а також створений сервіс, його тип, IP-адресу всередині Kubernetes-кластера та відкриті внутрішні/зовнішні порти (див. рис. 3.32).

```
[sergiy@H610M-K ~]$ kubectl get all
NAME                                            READY   STATUS    RESTARTS          AGE
pod/spark-master-0                              1/1     Running   0                 47h
pod/spark-wordcount-deployment-995c898f6-2cl9x  1/1     Running   406 (4m24s ago)   47h
pod/spark-wordcount-deployment-995c898f6-zr99p  1/1     Running   408 (3m19s ago)   47h
pod/spark-worker-0                              1/1     Running   0                 47h
pod/spark-worker-1                              1/1     Running   0                 47h

NAME                              TYPE           CLUSTER-IP    EXTERNAL-IP                                                              PORT(S)           AGE
service/kubernetes                ClusterIP      10.1.0.1      <none>                                                                   443/TCP           47h
service/spark-headless            ClusterIP      None          <none>                                                                   <none>            47h
service/spark-master-svc          ClusterIP      10.1.179.80   <none>                                                                   7077/TCP,80/TCP   47h
service/spark-wordcount-service   LoadBalancer   10.1.84.76    a07dac6e09aed4480973ab63ceb39967-1658022929.us-east-1.elb.amazonaws.com   4040:31016/TCP    47h

NAME                                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/spark-wordcount-deployment   2/2     2            2           47h

NAME                                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/spark-wordcount-deployment-995c898f6   2         2         2       47h

NAME                             READY   AGE
statefulset.apps/spark-master    1/1     47h
statefulset.apps/spark-worker    2/2     47h
[sergiy@H610M-K ~]$
```

Рисунок 3.32 – Запущений deployment та сервіс Spark-додатку

Відкриємо браузер та впишемо зовнішню публічну IP-адресу та порт 4040, що представлені на рисунку 3.32. З'явиться веб-інтерфейс Spark, в якому можна побачити результат успішного виконання нашого додатку, кількість стадій, тривалість та багато іншого (див. рис. 3.33).

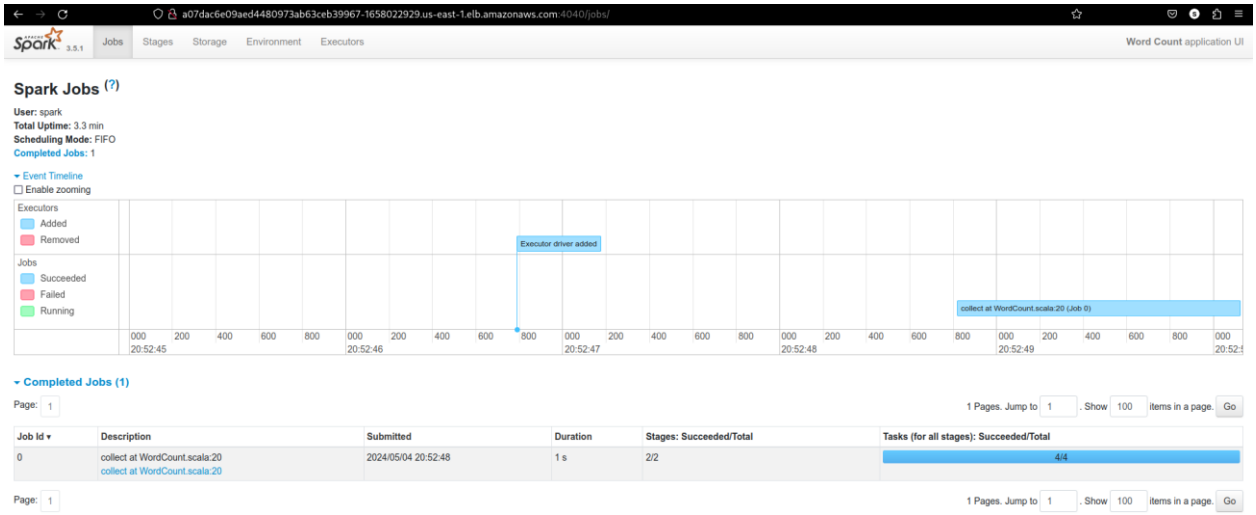

Рисунок 3.33 – Веб-інтерфейс Spark

Для прикладу розглянемо детальніше візуалізацію спрямованого ациклічного графа (див. рис. 3.34). Можна побачити, що наш додаток має дві стадії, які представлені горизонтально. В першій стадії відбувається операція читання вхідного файлу із великим набором слів, після чого вхідні рядки розбиваються на слова за пробілами і в подальшому всі списки слів об'єднуються в один великий список слів. Остання функція у першій стадії перетворює кожне слово в пару (ключ, значення), де ключем є саме слово, а значенням – число 1. В другій стадії є одна функція, яка бере всі пари (слово, 1) і підсумовує значення для кожного унікального слова.

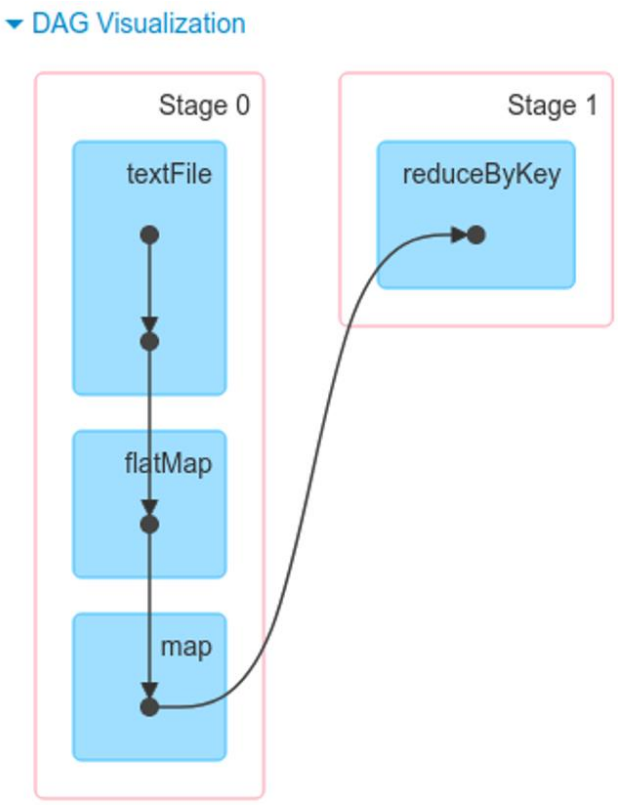

Рисунок 3.34 – Спрямований ациклічний граф

Фрагменти виведення, які є результатом виконання програми наведено на рисунках 3.35 та 3.36.

```
sergij_andrunkiv@cloudshell:~ (big-data-kubernetes-cluster)$ kubectl logs spark-wordcount-gcs-deployment-7dc5f75747-8r786
spark 22:37:10.58 INFO  ==>
spark 22:37:10.58 INFO  ==> Welcome to the Bitnami spark container
spark 22:37:10.59 INFO  ==> Subscribe to project updates by watching https://github.com/bitnami/containers
spark 22:37:10.59 INFO  ==> Submit issues and feature requests at https://github.com/bitnami/containers/issues
spark 22:37:10.59 INFO  ==> Upgrade to Tanzu Application Catalog for production environments to access custom-configured and pre-packaged software components. Gain e
nhanced features, including Software Bill of Materials (SBOM), CVE scan result reports, and VEX documents. To learn more, visit https://bitnami.com/enterprise
spark 22:37:10.59 INFO  ==>

24/06/06 22:37:17 INFO SparkContext: Running Spark version 3.5.1
24/06/06 22:37:17 INFO SparkContext: OS info Linux, 6.1.75+, amd64
24/06/06 22:37:17 INFO SparkContext: Java version 17.0.11
24/06/06 22:37:17 INFO ResourceUtils: ==============================================================
24/06/06 22:37:17 INFO ResourceUtils: No custom resources configured for spark.driver.
24/06/06 22:37:17 INFO ResourceUtils: ==============================================================
24/06/06 22:37:17 INFO SparkContext: Submitted application: Word Count
24/06/06 22:37:18 INFO ResourceProfile: Default ResourceProfile created, executor resources: Map(cores -> name: cores, amount: 1, script: , vendor: , memory -> name:
 memory, amount: 1024, script: , vendor: , offHeap -> name: offHeap, amount: 0, script: , vendor: ), task resources: Map(cpus -> name: cpus, amount: 1.0)
24/06/06 22:37:18 INFO ResourceProfile: Limiting resource is cpu
24/06/06 22:37:18 INFO ResourceProfileManager: Added ResourceProfile id: 0
24/06/06 22:37:18 INFO SecurityManager: Changing view acls to: spark
24/06/06 22:37:18 INFO SecurityManager: Changing modify acls to: spark
24/06/06 22:37:18 INFO SecurityManager: Changing view acls groups to:
24/06/06 22:37:18 INFO SecurityManager: Changing modify acls groups to:
24/06/06 22:37:18 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: spark; groups with view permissions:
 EMPTY; users with modify permissions: spark; groups with modify permissions: EMPTY
24/06/06 22:37:18 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
24/06/06 22:37:19 INFO Utils: Successfully started service 'sparkDriver' on port 37919.
24/06/06 22:37:20 INFO SparkEnv: Registering MapOutputTracker
24/06/06 22:37:20 INFO SparkEnv: Registering BlockManagerMaster
24/06/06 22:37:20 INFO BlockManagerMasterEndpoint: Using org.apache.spark.storage.DefaultTopologyMapper for getting topology information
24/06/06 22:37:20 INFO BlockManagerMasterEndpoint: BlockManagerMasterEndpoint up
24/06/06 22:37:20 INFO SparkEnv: Registering BlockManagerMasterHeartbeat
24/06/06 22:37:20 INFO DiskBlockManager: Created local directory at /tmp/blockmgr-ff17bd45-2c33-4fa1-a0f1-5de7d3779733
24/06/06 22:37:20 INFO MemoryStore: MemoryStore started with capacity 434.4 MiB
```

Рисунок 3.35 – Фрагмент логу розгортання додатку

```
24/06/06 22:37:30 INFO TaskSchedulerImpl: Removed TaskSet 1.0, whose tasks hav
24/06/06 22:37:30 INFO DAGScheduler: ResultStage 1 (collect at WordCount.scala
24/06/06 22:37:30 INFO DAGScheduler: Job 0 is finished. Cancelling potential s
24/06/06 22:37:30 INFO TaskSchedulerImpl: Killing all running tasks in stage 1
24/06/06 22:37:30 INFO DAGScheduler: Job 0 finished: collect at WordCount.scal
(ac,,11)
(interdum,46)
(sociosqu,4)
(mi.,16)
(ac,158)
(erat,66)
(tempus,,3)
(laoreet,43)
(fames,20)
(urna,42)
(auctor,,11)
(nunc,,11)
(lectus.,21)
(laoreet,,8)
(sapien.,15)
(metus,,12)
(quam,,9)
(nulla,54)
(lobortis,,6)
(rhoncus.,11)
(Fusce,50)
(sed.,5)
(luctus,,5)
(vulputate,,11)
(tincidunt,,7)
(nulla.,15)
(mollis.,11)
(ultricies,50)
(elementum,53)
(elit,,8)
(ridiculus,16)
```

Рисунок 3.36 – Фрагмент підрахованої кількості слів

3.6 Розгортання Kubernetes Dashboard всередині кластера

Для початку розгорнемо сам сервіс Kubernetes Dashboard з використанням шаблону YAML-маніфесту, що розміщений на GitHub (див. рис. 3.37).

```
sergij_andrunkiv@cloudshell:~ (big-data-kubernetes-cluster)$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.0-rc5/aio/deploy/recommende
d.yaml
namespace/kubernetes-dashboard created
serviceaccount/kubernetes-dashboard created
service/kubernetes-dashboard created
secret/kubernetes-dashboard-certs created
secret/kubernetes-dashboard-csrf created
secret/kubernetes-dashboard-key-holder created
configmap/kubernetes-dashboard-settings created
role.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrole.rbac.authorization.k8s.io/kubernetes-dashboard created
rolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
Warning: spec.template.spec.nodeSelector[beta.kubernetes.io/os]: deprecated since v1.14; use "kubernetes.io/os" instead
deployment.apps/kubernetes-dashboard created
service/dashboard-metrics-scraper created
Warning: spec.template.metadata.annotations[seccomp.security.alpha.kubernetes.io/pod]: non-functional in v1.27+; use the "seccompProfile" field instead
deployment.apps/dashboard-metrics-scraper created
sergij_andrunkiv@cloudshell:~ (big-data-kubernetes-cluster)$
```

Рисунок 3.37 – Розгортання сервісу Kubernetes Dashboard

Для того, або мати можливість авторизуватися через веб-інтерфейс, потрібно створити сервісний обліковий запис для Kubernetes Dashboard. У лістингах 3.10, 3.11 представлено YAML-маніфести для створення сервісного облікового запису разом із простором імен та ролі для цього облікового запису в контексті Kubernetes Dashboard.

Лістинг 3.10 – YAML-маніфест створення сервісного облікового запису

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kubernetes-dashboard
```

Лістинг 3.11 – YAML-маніфест присвоєння ролі облікового запису

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: admin-user
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: admin-user
  namespace: kubernetes-dashboard
```

Розгорнемо обидва маніфести та створимо токен для щойно створеного користувача з відповідними правами (див. рис. 3.38).

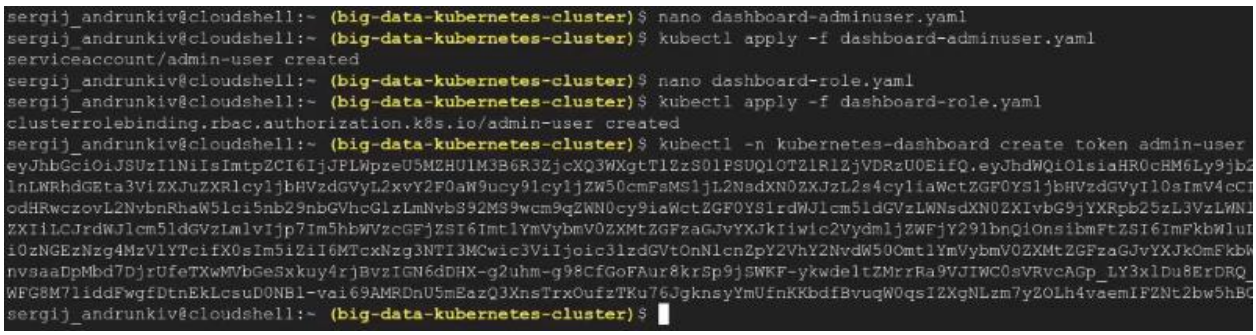

Рисунок 3.38 – Генерація токену для користувача Dashboard

Запустимо `kubectl proxy` та відкриємо веб-інтерфейс Kubernetes Dashboard на 8001 порту. Авторизуємось за допомогою згенерованого токену і можемо побачити головну сторінку сервісу (див. рис. 3.39).

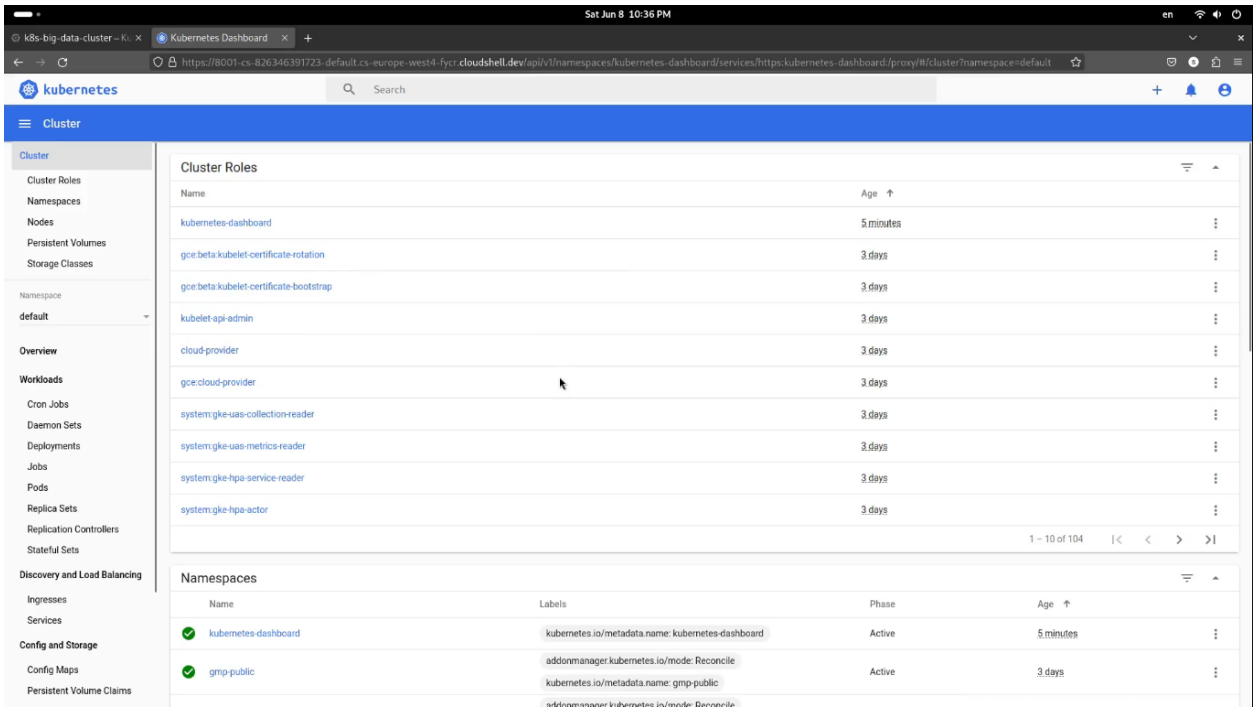

Рисунок 3.39 – Веб-інтерфейс Kubernetes Dashboard

На вкладці Nodes знаходиться інформація про використання ресурсів кластера у реальному часі. Можна побачити загальне використання ЦП та оперативної пам’яті для всього кластера, а також окремий стан кожного вузла (див. рис. 3.40).

На вкладці Deployments знаходиться список усіх deployment на кластері. На рисунку 3.41 можна побачити щойно розгорнутий Spark додаток для підрахунку набору слів з інформацією про простір імен, дату розгортання, час безперервної роботи, UID, використання ресурсів, статуси подів та інше.

На вкладці Services усі розгортання репрезентуються як сервіси із власним тегом, зовнішньою та внутрішньою IP-адресами (див. рис. 3.42).

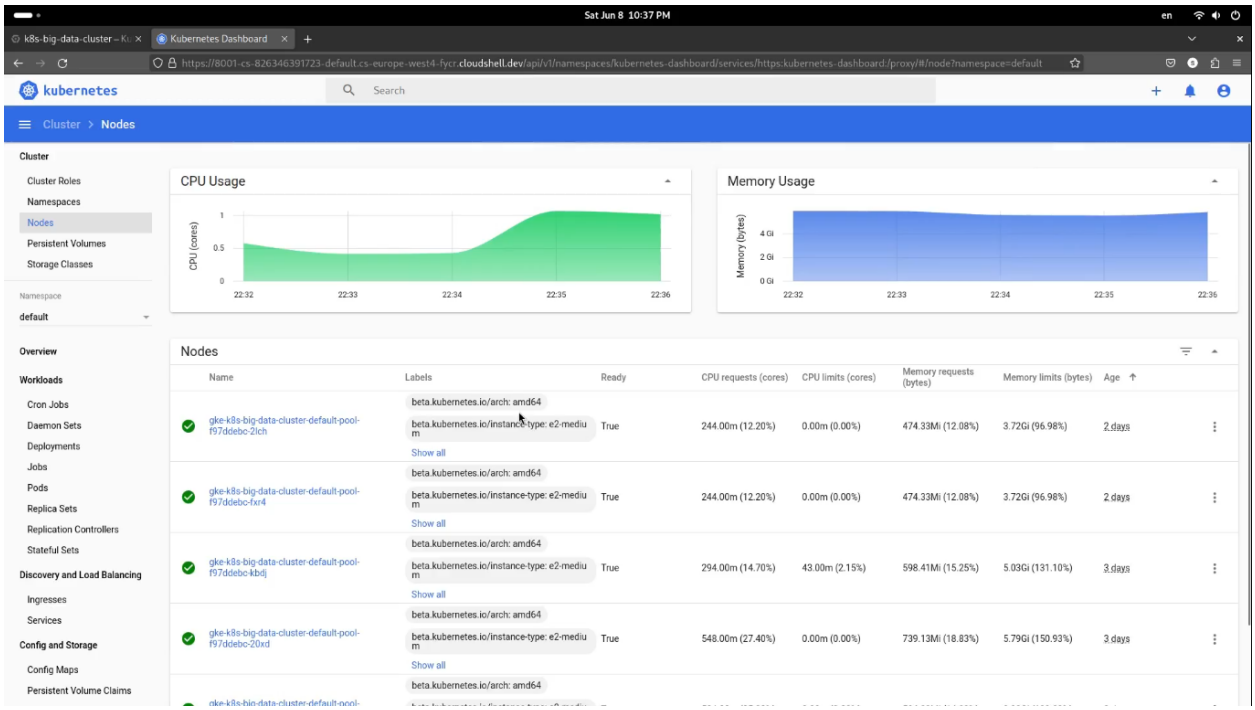

Рисунок 3.40 – Вкладка з моніторингом ресурсів кластера

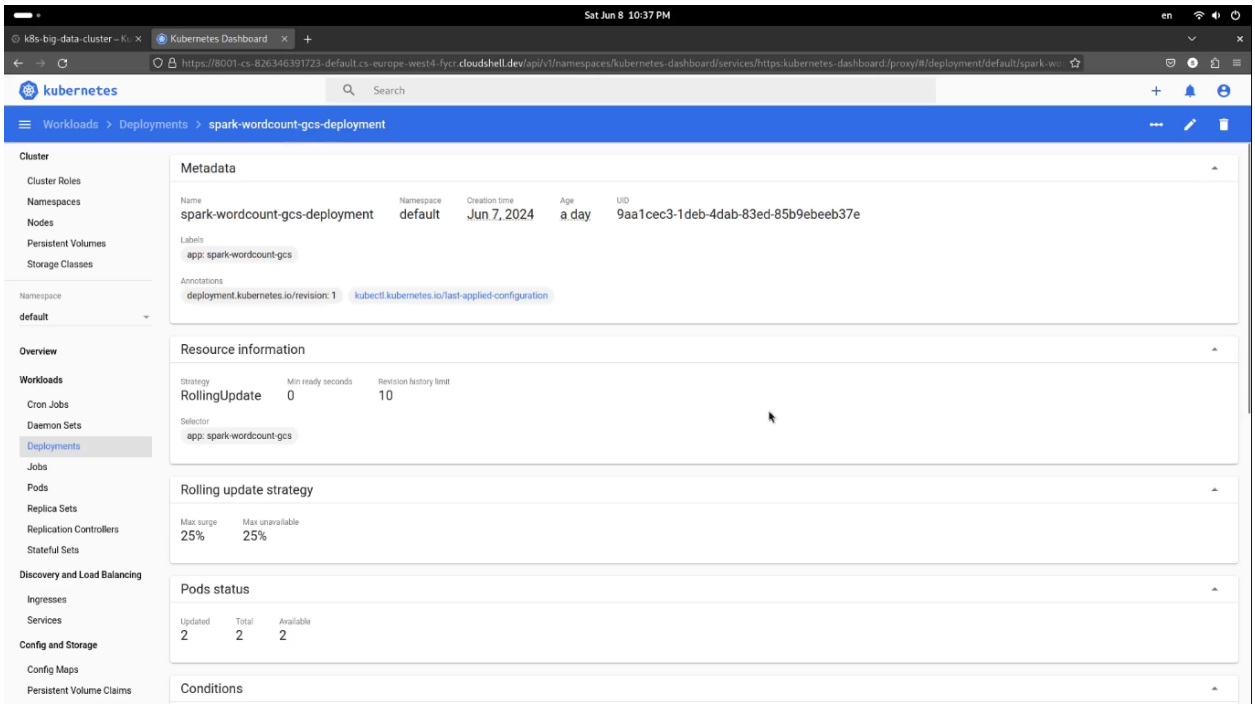

Рисунок 3.41 – Вкладка з розгорнутими додатками на кластері

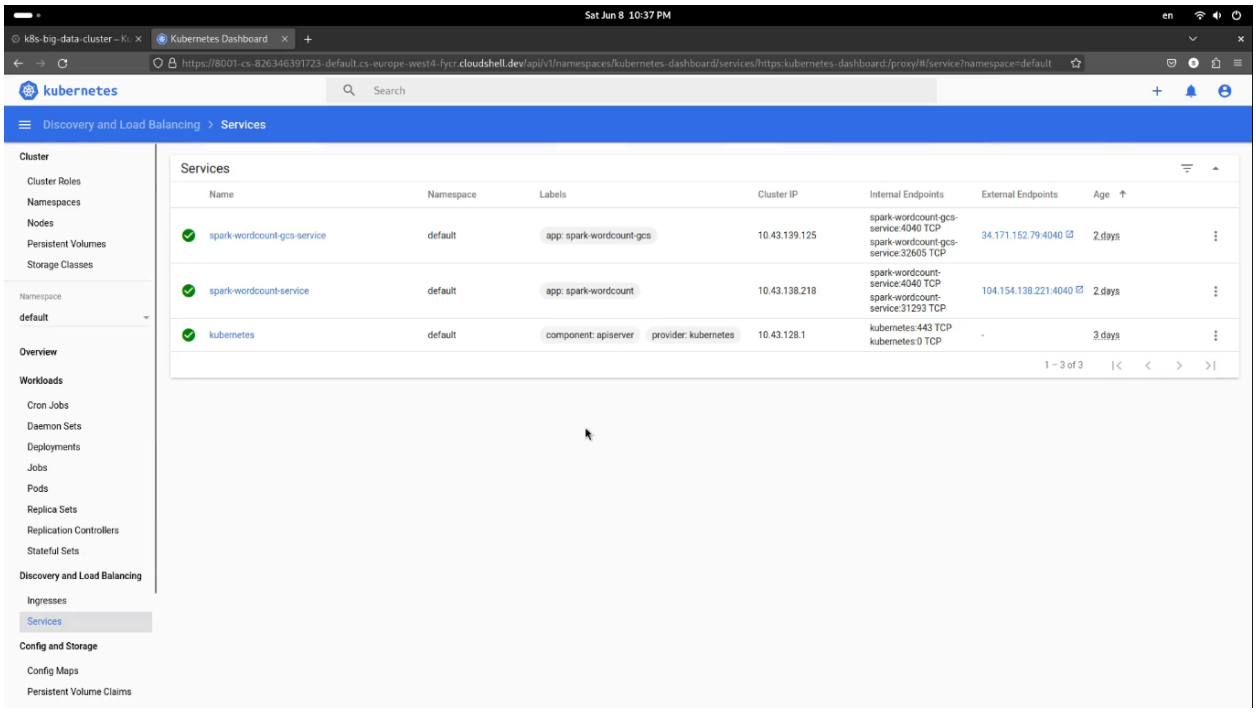

Рисунок 3.42 – Вкладка з сервісами, які є розгорнутими додатками

В результаті ми отримали повністю функціональний Kubernetes-кластер у хмарах провайдерів Google та Amazon, який забезпечує велику відмовостійкість, надійність, доступність та розширюваність за оптимальну плату та є повністю придатним для розгортання у ньому різних паралельних та розподілених додатків, що потребують значних ресурсів для виконання різних задач та їх моніторингу.

РОЗДІЛ 4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Заходи безпеки у дата-центрах

Дата-центри є критично важливими компонентами сучасної інформаційної інфраструктури, які забезпечують зберігання, обробку та передачу великих обсягів даних. Вони підтримують роботу підприємств, урядових установ, наукових досліджень та безлічі інших галузей, де надійний і безперебійний доступ до інформації має вирішальне значення. Однак, функціонування дата-центрів пов'язане з різноманітними ризиками, які можуть поставити під загрозу як саму інфраструктуру, так і дані, що зберігаються в ній. Для забезпечення безпеки та безперебійної роботи необхідно впроваджувати комплексні заходи безпеки, які охоплюють фізичну, електричну, кліматичну безпеку, а також захист від несанкціонованого доступу та управління відходами.

Однією із головних заходів безпеки у дата-центрах є протипожежна безпека, оскільки пожежа може завдати значних збитків, включаючи втрату обладнання та даних, а також серйозні порушення у наданні послуг. Для забезпечення протипожежної безпеки в дата-центрах необхідно впровадити комплексний підхід, що включає наступні заходи:

– встановлення систем раннього виявлення диму та автоматичних систем пожежогасіння, таких як системи на основі інертних газів (Inergen або FM-200), що не пошкоджують електронне обладнання;

– розробка та розміщення чітких планів евакуації, що вказують маршрути виходу та розташування засобів пожежогасіння, проведення регулярних навчань для персоналу щодо дій у разі пожежі;

– встановлення протипожежних перегородок і дверей для обмеження поширення вогню;

– використання негорючих матеріалів у конструкціях будівлі.

Електробезпека в дата-центрах має першорядне значення, оскільки неправильно спроектована або погано обслуговувана електрична система може стати причиною пожеж, уражень струмом та інших аварійних ситуацій.

Для забезпечення електробезпеки впроваджуються наступні заходи:

– забезпечення належного заземлення всього електричного обладнання для запобігання ураженню струмом та електричним розрядам, які можуть пошкодити чутливе обладнання;

– використання джерел безперебійного живлення для захисту від перепадів напруги та забезпечення безперервної роботи обладнання у випадку короткочасних відключень електроенергії;

– регулярне обслуговування та перевірка електричних щитів і кабелів на предмет пошкоджень або зносу;

– забезпечення належного маркування та організації кабелів для зниження ризику виникнення аварійних ситуацій.

Підтримання оптимального клімату в дата-центрі є ключовим для забезпечення стабільної та надійної роботи обладнання. Висока температура та вологість можуть призвести до перегріву та виходу з ладу техніки. Основні заходи з клімат-контролю включають:

– використання потужних систем кондиціонування повітря для підтримки оптимальної температури в приміщеннях дата-центру;

– впровадження методів розподілу холодного та гарячого повітря для підвищення ефективності охолодження;

– забезпечення належної вентиляції для підтримання циркуляції повітря і видалення надмірного тепла та вологості;

– постійний моніторинг температури та вологості за допомогою датчиків і систем контролю, налаштування автоматичного сповіщення про відхилення від встановлених параметрів.

Захист від несанкціонованого фізичного доступу до обладнання дата-центру є критичним для запобігання крадіжкам, вандалізму та інших загроз. Для цього встановлюються системи контролю доступу, що використовують картки доступу, біометричні сканери або коди для обмеження доступу до чутливих зон. Ведення журналів доступу для відстеження відвідувачів також сприяє підвищенню безпеки. Розгортання систем відеоспостереження з високою роздільною здатністю для моніторингу всіх зон дата-центру, а також забезпечення збереження записів для подальшого аналізу у разі інцидентів, є важливими складовими захисту. Залучення охоронних патрулів для регулярного обходу території дата-центру та оперативного реагування на підозрілі дії або порушення забезпечує додатковий рівень захисту.

Шум та вібрація в дата-центрах можуть негативно впливати на роботу обладнання та здоров'я персоналу. Основні заходи щодо зменшення шуму та вібрації включають використання спеціальних шумопоглинаючих матеріалів у конструкціях стін та підлоги для зниження рівня шуму, а також встановлення серверів та іншого обладнання на антивібраційні платформи або підставки для мінімізації впливу вібрації. Регулярний моніторинг рівня шуму та вібрації, аналіз їхнього впливу та впровадження коригуючих заходів при перевищенні допустимих норм сприяють підтриманню належних умов для роботи обладнання та персоналу.

Дата-центри генерують значну кількість електронних та інших відходів. Управління цими відходами є важливим аспектом екологічної безпеки та відповідності нормативним вимогам. Основні заходи включають організацію процесів рециклінгу електронних відходів, таких як старі сервери, кабелі та інше обладнання, а також утилізацію небезпечних відходів відповідно до екологічних стандартів. Впровадження політик зменшення кількості відходів також є важливим елементом управління відходами. Проведення навчань для персоналу щодо

належного поводження з відходами та важливості дотримання екологічних стандартів сприяє підвищенню обізнаності та відповідальності.

Забезпечення безпеки в дата-центрах вимагає комплексного підходу, який охоплює протипожежну безпеку, електробезпеку, клімат-контроль, захист від фізичного доступу, зменшення шуму та вібрації, а також управління відходами. Впровадження цих заходів не лише забезпечує безперебійну та надійну роботу обладнання, але й захищає здоров'я та безпеку персоналу, а також знижує негативний вплив на навколишнє середовище.

4.2 Управління охороною праці

Управління охороною праці в комп'ютерній сфері є критично важливим для забезпечення безпеки та здоров'я працівників. Зі збільшенням ролі інформаційних технологій у всіх сферах життя, питання охорони праці набувають дедалі більшої актуальності. Комплексний підхід до управління охороною праці включає аналіз потенційних ризиків, розробку та впровадження заходів для їх мінімізації, постійний моніторинг та вдосконалення існуючих процедур. У розділі розглянуто ключові аспекти управління охороною праці в контексті комп'ютерної сфери, зокрема ідентифікацію та оцінку ризиків, розробку та впровадження заходів з охорони праці, моніторинг та оцінку ефективності, а також постійне вдосконалення.

Для ефективного управління охороною праці потрібно ідентифікувати та оцінити ризики, пов'язані з роботою в комп'ютерній сфері. Це включає аналіз умов праці, обладнання, програмного забезпечення та робочих процесів. Фізичні ризики включають ергономічні фактори, електробезпеку, освітлення, рівень шуму та мікроклімат. Психосоціальні ризики пов'язані зі стресом, робочим навантаженням, соціальною ізоляцією та іншими факторами, що можуть негативно впливати на психічне здоров'я працівників. Використання методик оцінки ризиків, таких як

аналіз небезпек та критичних контрольних точок, допомагає виявити найбільш небезпечні аспекти роботи.

Важливо звертати увагу на специфічні ризики, що пов'язані з використанням комп'ютерних систем. Це можуть бути ризики електромагнітного випромінювання, ризики, пов’язані з неправильним користуванням комп'ютерними програмами, або збої у системах, що можуть спричинити стресову ситуацію. Важливим є також врахування можливих ризиків кібербезпеки, які можуть вплинути на психологічний стан працівників.

На основі результатів ідентифікації та оцінки ризиків розробляються конкретні заходи для мінімізації або усунення виявлених загроз. Це включає:

– використання регульованих стільців та столів, належне розташування моніторів, клавіатур та мишей, забезпечення оптимального освітлення;

– регулярна перевірка та обслуговування електрообладнання, використання захисних пристроїв, таких як джерела безперебійного живлення (UPS), встановлення систем заземлення;

– організація регулярних перерв, забезпечення можливості відпочинку, підтримка відкритого спілкування між працівниками та керівництвом, створення сприятливого мікроклімату в колективі;

– проведення регулярних навчань з охорони праці, інформування про нові ризики та способи їх запобігання, навчання методам самодопомоги та управління стресом.

Після впровадження заходів з охорони праці необхідно проводити постійний моніторинг їх ефективності. Це включає регулярний аналіз умов праці, збір та аналіз даних про інциденти та нещасні випадки, опитування працівників щодо їх задоволеності умовами праці. Систематичний збір даних про кількість інцидентів та їх причини дозволяє виявити слабкі місця в системі охорони праці та вжити заходів для їх усунення. На основі отриманих даних вносяться корективи до існуючих процедур та заходів з охорони праці. Важливим аспектом є залучення

працівників до процесу оцінки та покращення умов праці, що сприяє підвищенню їхньої мотивації та відповідальності за дотримання правил безпеки.

Управління охороною праці є динамічним процесом, що потребує комплексного підходу, що включає ідентифікацію та оцінку ризиків, розробку та впровадження заходів для їх мінімізації, постійний моніторинг та оцінку ефективності, а також постійне вдосконалення існуючих процедур.. Зміни в технологіях, робочих процесах, законодавчих вимогах та інших аспектах можуть вимагати перегляду існуючих заходів з охорони праці та розробки нових. Регулярні аудити, внутрішні та зовнішні перевірки, а також участь у програмах обміну досвідом з іншими компаніями допомагають виявляти нові можливості для покращення умов праці. Забезпечення безпеки та здоров'я працівників є ключовим елементом успішного функціонування будь-якої організації, що використовує комп'ютерні технології. Впровадження системи управління охороною праці сприяє підвищенню продуктивності, зниженню рівня травматизму та захворювань, а також підвищенню загального рівня задоволеності працівників.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи було досліджено процес створення Kubernetes кластера в гібридній хмарі для виконання завдань Big Data. Основна мета роботи полягала у розробці ефективної та масштабованої інфраструктури, яка б дозволила використовувати ресурси декількох хмарних платформ для обробки великих даних.

Основні результати та досягнення в ході виконання кваліфікаційної роботи:

– проведено аналіз інструментів для створення та управління Kubernetes-кластерами, GCP вибрано як централізований інструмент для управління кластером, а AWS – як платформу для розгортання кластерів через її надійність, масштабованість та глобальну доступність;

– з використанням Terraform було розроблено скрипти для автоматизованого розгортання інфраструктури, що включає налаштування як GCP, так і AWS ресурсів;

– використання Google Anthos дозволило об'єднати різні хмарні середовища в єдиний керований кластер, забезпечуючи централізоване управління та контроль за безпекою;

– з використанням сервісних облікових записів Google Cloud було створено сховище даних GCS для зберігання в ньому наборів великих даних для Spark додатків;

– застосування мови Scala разом із фреймворком Spark дало можливість ефективно реалізовувати алгоритми обробки даних, використовуючи потужний синтаксис та багатий набір бібліотек;

– інструмент контейнеризації Docker забезпечив ізоляцію Spark-додатків та його залежностей, що сприяло портативності та простоті розгортання додатків;

– використання DockerHub спростило керування образами контейнерів, дозволяючи автоматично оновлювати та розгортати масштабовані додатки у Kubernetes кластері з використанням YAML;

– розгортання і налаштування на кластері Kubernetes Dashboard дозволило отримувати про кластер та задачі більш детальну інформацію у більш зручному форматі у вигляді веб-інтерфейсу;

– розглянуто питання безпеки заходи безпеки у дата-центрах у яких розташовуються хмарні сервіси.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Lutskiv A. Adaptable Text Corpus Development for Specific Linguistic Research / Andriy Lutskiv, Nataliya Popovych // International Scientific-Practical Conference «Problems of Infocommunications. Science and Technology» (October 8-11, 2019), 2019. - C.217-223.

2. Lutskiv A. Big Data Approach to Developing Adaptable Corpus Tools /Andriy Lutskiv, Nataliya Popovych// Computational Linguistics and Intelligent Systems. Proc. 4thInt. Conf. COLINS 2020. Volume I:Workshop. Lviv, Ukraine, April23-24, 2020, CEUR-WS.org, online. pp.374-395. URL: http://ceur-ws.org/Vol-2604/.

3. Lutskiv A. Big data-based approach to automated linguistic analysis effectiveness.//A. Lutskiv, N. Popovych/ IEEE Third International Conference on Data Stream Mining & Processing August 21-25, 2020, Lviv, Ukraine pp.438–443. DSMP 2020.

4. Lutskiv A. Corpus-Based Translation Automation in Adaptable Corpus Translation Module /Andriy Lutskiv, Roman Lutsyshyn// Computational Linguistics and Intelligent Systems. Proc. 5th Int. Conf. COLINS 2021. Volume I: Workshop. Lviv, Ukraine, April 22-23, 2021, CEUR-WS.org, online. pp.374-395. URL: http://ceur-ws.org/Vol-2604/.

5. Yatsyshyn V. A Risks management method based on the quality requirements communication method in agile approaches / Vasyl Yatsyshyn, Oleh Pastukh, Andriy Lutskiv, Viktor Tsymbalistyy, Nataliia Martsenko //Information Technologies: Theoretical and Applied Problems 2022 (ITTAP 2022), Ternopil, Ukraine, November 22-24, 2022. pp.1-10.

6. Осухівська Г. М., Тиш Є. В., Луцик Н. С., Паламар А. М. Методичні вказівки до виконання кваліфікаційних робіт здобувачів першого (бакалаврського) рівня вищої освіти спеціальності 123 «Комп'ютерна інженерія» усіх форм навчання. Тернопіль, ТНТУ. 2022. 28 с.

7. Kubernetes Infrastructure - Infrastructure Components. URL: https://docs.openshift.com/dedicated/3/architecture/infrastructure_components/kubernetes_infrastructure.html (дата звернення: 25.04.2024).

8. Deploy to Kubernetes | Docker Docs. URL: https://docs.docker.com/guides/deployment-orchestration/kube-deploy (дата звернення: 25.04.2024).

9. Google Anthos: The First True Multi Cloud Platform? URL: https://bluexp.netapp.com/blog/gcp-cvo-blg-google-anthos-the-first-true-multi-cloud-platform (дата звернення: 26.04.2024).

10. What is Big Data and Why is it Important? URL: https://www.techtarget.com/searchdatamanagement/definition/big-data (дата звернення: 30.04.2024).

11. What Is Big Data? Definition, How It Works, and Uses. URL: https://www.investopedia.com/terms/b/big-data.asp (дата звернення: 30.04.2024).

12. Apache Spark - Wikiwand. URL: https://wikiwand.com/uk/Apache_Spark (дата звернення: 02.05.2024).

13. Configuration - Spark 3.5.1 Documentation. URL: https://spark.apache.org/docs/latest/configuration.html (дата звернення: 02.05.2024).

14. Terraform Registry. URL: https://registry.terraform.io (дата звернення: 04.05.2024).

15. Build infrastructure | Terraform | HashiCorp Developer. URL: https://developer.hashicorp.com/terraform/tutorials/gcp-get-started/google-cloud-platform-build (дата звернення: 04.05.2024).

16. Леськів Г.З., Верескля М.Р. Безпека життєдіяльності та охорона праці: навчальний посібник / Г.З. Верескля, М.Р. Верескля. - Львів: ЛДУВС, 2018. - 262 с.

17. Грибан В. Г., Фоменко А. Є., Казначеєв Д. Г. Г 82 Безпека життєдіяльності та охорона праці : підруч. / В. Г. Грибан, А. Є. Фоменко, Д. Г. Казначеєв. Дніпро: Дніпроп. держ. ун-т внутр. справ, 2022. 388 с.

ДОДАТОК А.

Технічне завдання

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії

Кафедра комп'ютерних систем та мереж

"Затверджую"

Завідувач кафедри КС

_____ Осухівська Г.М.

"___" _____ 2024 р.

KUBERNETES КЛАСТЕР В ГІБРИДНІЙ ХМАРІ ДЛЯ ВИКОНАННЯ ЗАВДАНЬ BIG DATA

ТЕХНІЧНЕ ЗАВДАННЯ

на 9 листках

Вид робіт: Кваліфікаційна робота

На здобуття освітнього ступеня бакалавр

Спеціальність 123 «Комп'ютерна інженерія»

"УЗГОДЖЕНО"

Керівник кваліфікаційної роботи

_____ к.т.н., доц. Луцків А.М.

"___"_____ 2024 р.

"ВИКОНАВЕЦЬ"

Студент групи СІ-41

_____ Андруньків С.Р.

"___"_____ 2024 р.

Тернопіль 2024

1. Загальні відомості

1.1 Повна назва та її умовне позначення

Повна назва теми кваліфікаційної роботи: «Kubernetes кластер в гібридній хмарі для виконання завдань Big Data».

Умовне позначення кваліфікаційної роботи: КС КРБ 123.106.00.00

1.2 Виконавець

Студент групи СІ-41, факультету комп'ютерно-інформаційних систем і програмної інженерії, кафедри комп'ютерних систем та мереж, Тернопільського національного технічного університету імені Івана Пулюя, Андруньків Сергій Романович.

1.3 Підстава для виконання роботи

Підставою для виконання кваліфікаційної роботи є наказ по університету (№ 4/7-408 від 24.04.2024 р.)

1.4 Планові терміни початку та завершення роботи

Плановий термін початку виконання кваліфікаційної роботи - 24.04.2024 р.

Плановий термін завершення виконання кваліфікаційної роботи – 26.06.2024 р.

1.5 Порядок оформлення та пред'явлення результатів роботи

Порядок оформлення пояснювальної записки та графічного матеріалу здійснюється у відповідності до чинних норм та правил ISO, ЕСКД, ЕСПД та ДСТУ.

Пред’явлення проміжних результатів роботи з виконання кваліфікаційної роботи здійснюється у відповідності до графіку, затвердженого керівником роботи.

Попередній захист кваліфікаційної роботи відбувається при готовності роботи на 90% , наявності пояснювальної записки та графічного матеріалу.

Пред’явлення результатів кваліфікаційної роботи відбувається шляхом захисту на відповідному засіданні ЕК, ілюстрацією основних досягнень за допомогою графічного матеріалу.

2. Призначення і цілі створення системи

2.1 Призначення системи

Kubernetes кластер в гібридній хмарі для виконання завдань Big Data призначений для забезпечення ефективного, гнучкого та масштабованого середовища, яке поєднує в собі декілька хмарних інфраструктур для обробки великих обсягів даних. Основне призначення системи полягає в оптимізації використання обчислювальних ресурсів, забезпеченні високої доступності та надійності сервісів, а також в автоматизації процесів розгортання, управління та масштабування додатків для Big Data.

2.2 Мета створення системи

Мета створення системи «Kubernetes кластер в гібридній хмарі для виконання завдань Big Data» полягає в розробці та впровадженні ефективної та економічно вигідної інфраструктури для обробки великих обсягів даних.

Основні завдання, які потрібно виконати для досягнення поставленої мети:

– спроектувати загальну структуру та функціональну схему роботи кластера;

– визначити вимоги до апаратної та програмної частин (к-сть ЦП, обсяг ОЗП, тип і версія ОС, системні утиліти);

– підготувати локальну програмну платформу для розгортання та адміністрування кластера (gcloud, kubectl, terraform);

– визначитися із вибором хмарних провайдерів, що надають платформи для розгортання в них кластерів;

– вибрати та обгрунтувати необхідні інструменти для автоматизації розгортання та Big Data обчислень;

– підготувати Terraform-скрипти для розгортання інфраструктури на платформах вибраних хмарних провайдерів;

– розробити додаток для опрацювання великих даних та запакувати його у контейнер;

– підготувати YAML-маніфест для створення деплойменту на розгорнутому кластері;

– проаналізувати результати та провести моніторинг використання ресурсів.

2.3 Характеристика об’єкту

2.3.1 Основні задачі та функції об’єкту

Об’єктом системи є гібридний Kubernetes кластер, що включає в себе компоненти декількох хмарних інфраструктур, призначених для виконання завдань Big Data. Система призначена для обробки великих обсягів даних, виконання аналітичних задач та забезпечення високої продуктивності додатків.

До основних задач та функцій об’єкту відносяться:

– масштабування обчислювальних ресурсів, що включає в себе автоматичне додавання та видалення обчислювальних вузлів в залежності від навантаження, а також підтримка горизонтального масштабування;

– забезпечення рівномірного розподілу обчислювальних задач між вузлами кластера з використанням балансувальників навантаження для оптимізації обробки даних;

– автоматичне розгортання контейнерів за допомогою інструментів оркестрації Kubernetes;

– використання хмарної платформи AWS для забезпечення апаратних обчислювальних ресурсів кластера та платформи GCP для централізованого керування кластером, моніторингу та запуску завдань опрацювання великих даних;

– використання хмарного сховища даних Google Cloud Storage для зберігання великих обсягів даних;

– дотримання політик безпеки для захисту даних та ресурсів;

– управління ролями та доступом користувачів до різних компонентів системи;

– впровадження систем моніторингу для контролю стану кластера та додатків.

3. Вимоги до системи

3.1 Вимоги до системи в цілому

Система «Kubernetes кластер в гібридній хмарі для виконання завдань Big Data», як модульний проект повинен забезпечувати швидке, продуктивне, оптимізоване, доступне та відмовостійке середовище для запуску Spark додатків та/або інших завдань опрацювання великих даних з використанням горизонтального масштабування, хмарних платформ двох «компаній-гігантів» Amazon та Google, інструменту мультихмарного рішення Google Anthos та сховища даних GCS.

3.1.1 Вимоги до структури та функціонування системи

Структура Kubernetes кластера в гібридній хмарі для виконання завдань Big Data складається з таких апаратних та програмних компонентів:

– локальна платформа керування кластером із попередньо налаштованими інструментами (gcloud, aws-cli, terraform, kubectl і тд);

– приватна хмара AWS (обчислювальна частина Kubernetes кластера);

– екземпляри віртуальних машин (EC2 Instances) на платформі AWS;

– віртуальна приватна хмара (Amazon VPC) – логічно ізольована частина хмари веб-служб, яка надає доступ до Amazon Elastic Compute Cloud через віртуальну приватну мережу на основі IPsec;

– load balancer AWS ELB для розподілу навантаження між вузлами кластера;

– docker runtime – середовище для виконання docker-контейнерів на вузлах;

– spark-pods – екземпляри частин Spark-задач у docker-контейнерах.

– приватна хмара GCP (керована частина Kubernetes кластера);

– сховище для великих даних Google Cloud Storage.

3.1.2 Вимоги до способів та засобів зв’язку між компонентами системи

У Kubernetes кластері в гібридній хмарі з’єднання між компонентами здійснюється за допомогою кількох рівнів мережевих технологій та протоколів. Для забезпечення зв'язку між подами використовуються віртуальні мережеві інтерфейси – Veth pairs. Для з'єднання між подами, вузлами та зовнішніми клієнтами використовуються сервіси ClusterIP, NodePort та LoadBalancer.

AWS Direct Connect та Google Cloud Interconnect використовуються для з’єднання між собою хмарних компонентів різних провайдерів. Вони забезпечують приватні, високошвидкісні та надійні з'єднання між різними датацентрами, минаючи Інтернет.

Google Cloud Storage Connector дозволяє Spark додаткам читати та записувати дані безпосередньо в GCS. Він інтегрується з Hadoop FileSystem API.

3.1.3 Вимоги по діагностуванню системи

Діагностування даної системи варто проводити згідно встановленого розкладу або у випадку виникнення збоїв у роботі Kubernetes кластера.

Головні інструменти, що використовуються для проведення діагностики, тестування та моніторингу стану Kubernetes кластера:

– kubectl – потрібен для отримання інформації про стан ресурсів, виконання діагностичних команд та налагодження проблем;

– Kubernetes Dashboard – система для моніторингу через веб-інтерфейс для візуалізації стану кластера, подів, сервісів та інших ресурсів.

3.1.4 Перспективи розвитку, модернізація системи

Перспективи розвитку та модернізації Kubernetes кластера включають в себе впровадження механізмів автоматичного горизонтального та вертикального масштабування для обчислювальних вузлів та подів на основі навантаження та використання ресурсів.

Сюди також входить інтеграція нових фреймворків та бібліотек для обробки великих даних, впровадження нових інструментів для моніторингу з метою забезпечення більш глибокого та детального аналізу метрик та логів, впровадження нових можливостей для інтеграції з іншими хмарними провайдерами та/або локальною інфраструктурою.

3.1.5 Вимоги до надійності системи

Надійність до системи «Kubernetes кластер в гібридній хмарі для виконання завдань Big Data» впроваджується шляхом забезпечення безперервної та стабільної роботи. Розміщення кластера у платних інфраструктурах хмарних провайдерів забезпечує його надійне функціонування, безперервну доступність шляхом використання механізмів відмовостійкості та резервування критичних компонентів, впровадження механізмів захисту даних та мережевої безпеки для запобігання несанкціонованому доступу.

3.1.6 Вимоги до функцій та задач, які виконує система

Головною вимогою до функції Kubernetes кластера в гібридній хмарі є забезпечення ефективного відмовостійкого розподіленого середовища з можливістю розгортання в ньому високонагружених додатків опрацювання великих наборів даних.

Основні функції та задачі, які має виконувати система:

– оркестрація контейнерів на вузлах кластера;

– здатність до масштабованості при зміні навантаженості;

– здатність ефективно розподіляти частини задач серед вузлів кластера;

– здатність самовідновлювати екземпляри вузлів та/або контейнерів (подів), які вийшли з ладу/були аварійно завершені;

– можливість переглядати детальну інформацію про кластер через графічний інтерфейс;

– можливість розширювати та керувати сховищами даних S3 та GCS;

3.1.7 Вимоги до апаратного забезпечення

Вимоги до робочих вузлів AWS EC2:

– Кількість ядер процесора – 2;

– Оперативна пам’ять – 4 ГБ;

– Обсяг SSD – >= 30 ГБ;

– Швидкість мережевого обладнання – 5 Гбіт/с.

Вимоги до локальної платформи:

– Кількість ядер процесора – 2-4;

– Оперативна пам’ять – 4-8 ГБ;

– Обсяг SSD/HDD – 50 ГБ;

– Швидкість мережевого обладнання – 150 Мбіт/с.

3.1.8 Вимоги до програмного забезпечення

Програмне забезпечення, що використовується для розгортання, налаштування, керування та моніторингу Kubernetes кластера в гібридній хмарі для виконання завдань Big Data:

– Amazon Linux – спеціально оптимізована операційна система Linux для роботи на екземплярах робочих станцій AWS;

– Debian/RHEL/Arch based linux – операційна система на базі одного із трьох найпоширеніших дистрибутивів Linux для локальної платформи керування кластером;

– kubelet – агенти вузлів Kubernetes кластера;

– kubectl – утиліта керування та моніторингу кластера;

– gcloud – утиліта для доступу до хмарної платформи GCP;

– aws-cli – утиліта для доступу до хмарної платформи AWS;

– terraform – інструмент командного рядка для розгортання кластерної інфраструктури;

– docker – інструмент створення та запуску Docker-контейнерів;

– docker engine – середовище для виконання Docker-контейнерів.

4. Вимоги до документації

Документація повинна відповідати вимогам ЄСКД та ДСТУ

Комплект документації повинен складатись з:

– пояснювальної записки;

– графічного матеріалу:

1 Діаграма взаємодії кластера Kubernetes у гібридній хмарі.

2 Розгортання компонентів кластера Apache Spark на платформі Kubernetes.

3 Розгортання мультихмарного середовища з використанням Google Anthos.

4 Етапи розгортання Kubernetes кластера та запуску Spark додатків.

*Примітка: У комплект документації можуть вноситися міни та доповнення в процесі розробки.

5. Стадії та етапи проектування

Таблиця 1 – Стадії та етапи виконання кваліфікаційної роботи бакалавра

№ етапу	Назва етапу виконання кваліфікаційної роботи	Термін виконання
1	Ознайомлення із завданням кваліфікаційної роботи	01.02-09.02.2024
2	Аналіз технічного завдання	05.02-11.02.2024
3	Аналіз вимог до апаратного та програмного забезпечення	29.04-02.05.2024

Продовження таблиці 1

4	Розробка функціональної схеми роботи кластера	03.05-10.05.2024
5	Розробка алгоритму розгортання кластера	11.05-15.05.2024
6	Вибір та налаштування хмарних платформ	16.05-18.05.2024
7	Підготовка terraform-скриптів для розгортання інфраструктури	19.05-24.05.2024
8	Розгортання kubernetes-кластера	25.05-27.05.2024
9	Написання та запуск BigData-додатків у гібридному кластері	28.05-02.06.2024
10	Налаштування системи моніторингу та аналізу результатів	03.06-05.06.2024
11	Виконання підрозділу «Безпека життєдіяльності, основи охорони праці»	06.06-09.06.2024
12	Оформлення пояснювальної записки	16.06-20.06.2024
13	Нормоконтроль	20.06.2024
14	Попередній захист кваліфікаційної роботи	14.06.2024
15	Захист кваліфікаційної роботи	24.06.2024

6. Додаткові умови виконання кваліфікаційної роботи

Під час виконання кваліфікаційної роботи у дане технічне завдання можуть вноситися зміни та доповнення.