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

магістр
(назва освітнього ступеня)

на тему: Технології розгортання інфраструктур опрацювання великих даних у хмарних сервісах

Виконав(ла): студент(ка) VI курсу, групи СІм-61
спеціальності 123 “Комп’ютерна інженерія”
(шифр і назва спеціальності)

_____ Голубовський М. П.
(підпис) (прізвище та ініціали)

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

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

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

Рецензент _____ Михалик Д. М.
(підпис) (прізвище та ініціали)

Тернопіль
2020

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)
« 01 » 10 2020 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

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

за спеціальністю *123 Комп’ютерна інженерія*
(шифр і назва спеціальності)

студенту *Голубовському Михайлу Петровичу*
(прізвище, ім’я, по батькові)

1. Тема роботи *Технології розгортання інфраструктур опрацювання великих даних у хмарних сервісах*

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

Затверджені наказом ректора від «*28*» *вересня* 2020 року № *4/7-687*
2. Термін подання студентом завершеної роботи *15 грудня 2020*
3. Вихідні дані до роботи *документація постачальників хмарних послуг GCP, AWS, OpenStack*

4. Зміст роботи (перелік питань, які потрібно розробити)
1. ВСТУП
2. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ
3. ВИБІР СЕРЕДОВИЩА ДЛЯ РОЗГОРТАННЯ ІНФРАСТРУКТУРИ ОПРАЦЮВАННЯ ВЕЛИКИХ ДАНИХ
4. РОЗРОБКА КОДУ ОПИСУ ІНФРАСТРУКТУРИ ОПРАЦЮВАННЯ ВЕЛИКИХ ДАНИХ
5. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ
5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень, слайдів)
1 - Тема кваліфікаційної роботи магістра
2 - Задачі
3 - Порівняння інструментів для реалізації підходу IaC
4 - Порівняння середовищ для розгортання інфраструктур опрацювання великих даних
5 - Граф залежності ресурсів розробленої інфраструктури
6 - Результат тестування роботоздатності системи
7 - Рекомендації щодо розгортання інфраструктур порацювання великих даних у хмарних сервісах
8 - Висновки

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	*Осухівська Г.М., доц.*		
Безпека в надзвичайних	*Стадник І. Я., проф.*		
ситуаціях			

7. Дата видачі завдання *30 вересня 2020 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	*Аналіз останніх досліджень у галузі розгортання*	*30.09.2020 -*	*виконано*
	інфраструктур опрацювання великих даних	*15.10.2020*	
2	*Аналіз можливих способів побудови інфраструктур у*	*16.10.2020 -*	*виконано*
	публічних та приватних хмарах	*30.10.2020*	
3	*Аналіз проблем, які виникають при розгортанні*	*02.11.2020 -*	*виконано*
	інфраструктур опрацювання великих даних	*13.11.2020*	
4	*Вибір оптимальних методів для розгортання*	*16.11.2020 -*	*виконано*
	інфраструктур у хмарних сервісах	*18.11.2020*	
5	*Апробування запропонованих методів шляхом*	*19.11.2020 -*	*виконано*
	розгортання інфраструктури з їх використанням	*24.11.2020*	
6	*Виконання розділу “Охорона праці та безпека в*	*25.11.2020 -*	*виконано*
	надзвичайних ситуаціях”	*04.12.2020*	
7	*Оформлення пояснювальної записки та графічного*	*07.12.2020 -*	*виконано*
	матеріалу	*11.12.2020*	
8	*Попередній захист кваліфікаційної роботи магістра*	*15.12.2020*	*виконано*
9	*Захист кваліфікаційної роботи магістра*	*22.12.2020*	

Студент _____ *Голубовський М. П.*
(підпис) (прізвище та ініціали)

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

АНОТАЦІЯ

Технології розгортання інфраструктур опрацювання великих даних у хмарних сервісах // Кваліфікаційна робота магістра // Голубовський Михайло Петрович // ТНТУ, Комп'ютерна інженерія, група Сім-61 // Тернопіль, 2020 // с. – 64, рис. – 19, табл. – 13, бібліогр. – 16.

Ключові слова: великі дані, хмарні обчислення, інфраструктура як код.

Кваліфікаційну роботу магістра присвячено дослідженню технологій та методів для розгортання інфраструктур опрацювання великих даних у хмарних сервісах. Розглянуто основні моделі хмарних інфраструктур опрацювання великих даних. Адаптовано практики з розробки програмного забезпечення для побудови хмарних інфрастуктур. Розроблено алгоритм вибору середовища для розгортання інфраструктур опрацювання великих даних у хмарних сервісах з урахуванням типових вимог замовника. Описано розгортання інфраструктури опрацювань великих даних у хмарному сервісі GCP з використання підходу «інфраструктура як код» засобами інструменту Terraform.

ANNOTATION

Technologies of infrastructure scanning of big data processing in cloud services // Master thesis // Mykhailo Holubovskyi // TNTU, Computer Engineering, group SIm-61 // Ternopil, 2020, p. – 64, fig. – 19, tab. – 13, bibliography – 16.

Key words: big data, cloud computing, infrastructure as code.

In the master thesis are considered the basic models of cloud infrastructures of big data processing. Software development practices adapted for building cloud infrastructures. Was developed an algorithm for selecting the environment for the deployment of big data processing infrastructures in cloud services, taking into account the typical requirements of the customer. was described the deployment of the big data processing infrastructure in the GCP cloud service using the "infrastructure as code" approach and the Terraform tool.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API	Application Programming Interface
AWS	Amazon Web Services
GCP	Google Cloud Platform
PaaS	Platform as a service
SaaS	Software as a service
IaC	Infrastructure as code
CI/CD	Continuous integration and continuous delivery
HCL	HashiCorp Configuration Language
VM	Virtual machine
MAI	Метод аналізу ієрархій
НО	Нормована оцінка
ІУ	Індекс узгодженості
ВУ	Відношення узгодженості
УІУ	Усереднений індекс узгодженості

ВСТУП

Актуальність теми. Актуальність теми пояснюється тим, що у спеціалістів, які працюють у сфері великих даних є потреба проводити обчислення, проте налаштування, підтримка обчислювальних кластерів для їх цілей є часозатратною та дорогою послугою. Традиційні підходи до розгортання обчислювальних інфраструктур мають ряд недоліків, зокрема, час розгортання, вартість інфраструктури, неможливість проведення тестування перед внесенням змін і повторного використання напрацювань, непостійність при ручному конфігуруванні ресурсів. Дослідженнями у сфері інфраструктур для опрацювання великих даних займалися багато науковців, серед яких: Yuri Demchenko, Cees de Laat, Canh Ngo, Peter Membrey, Daniil Gordijenko та багато інших. Проте, у їх працях недостатньо уваги приділено алгоритму вибору моделі інфраструктури опрацювання великих даних у хмарних сервісах з урахуванням типових вимог замовника та адаптації практик з розробки програмного забезпечення для побудови таких інфраструктур.

Мета і завдання дослідження.

Метою роботи є дослідження технологій та методів для розгортання інфраструктур опрацювання великих даних у хмарних сервісах.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

– проаналізувати моделі, методи та засоби розгортання інфраструктур опрацювання великих даних у приватних та публічних хмарах;

– проаналізувати проблеми, які виникають при розгортання інфраструктур опрацювання великих даних у хмарних сервісах;

– обґрунтувати вибір оптимальних методів для розгортання інфраструктур опрацювання великих даних у хмарних сервісах;

– апробувати запропоновані методи шляхом розгортання інфраструктури з їх використанням;

– сформувати рекомендації та висновки щодо використання запропонованих методів.

Об'єкт дослідження: процес розгортання інфраструктури для опрацювання великих даних.

Предмет дослідження: технології розгортання інфраструктур опрацювання великих даних у хмарних сервісах.

Методи дослідження. Для проведення дослідження використано загальнонаукові методи пізнання: спостереження, експерименту, описування та порівняння, а також метод експертних оцінок, зокрема метод аналізу ієрархій Т. Сааті.

Наукова новизна одержаних результатів

Наукова новизна одержаних результатів полягає в наступному:

– розроблено алгоритм вибору інфраструктури опрацювання великих даних у хмарних сервісах з урахуванням типових вимог замовника;

– дістало подальший розвиток застосування практик з розробки програмного забезпечення до побудови хмарних інфраструктур;

– сформульовано рекомендації щодо розгортання інфраструктур опрацювання великих даних у хмарних сервісах.

Практичне значення одержаних результатів

Отримані результати дозволяють обрати модель розгортання інфраструктур опрацювання великих даних у хмарних сервісах; обрати оптимальні методи для технології для побудови та розгортання такої інфраструктури.

Публікації. Результати дослідження апробовано на IX Міжнародній науково-технічній конференції молодих учених та студентів «Актуальні задачі сучасних технологій», VIII науково-технічній конференції «Інформаційні моделі, системи та технології».

Структура роботи. Робота складається з пояснювальної записки та графічної частини. Пояснювальна записка складається із вступу, 4 розділів, висновків, списку використаних джерел та додатків. Обсяг роботи: пояснювальна записка – 60 аркушів формату А4, графічна частина – 8 аркушів формату А1.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ РОЗГОРТАННЯ ІНФРАСТРУКТУР ОПРАЦЮВАННЯ ВЕЛИКИХ ДАНИХ У ХМАРНИХ СЕРВІСАХ

1.1. Огляд предметної області великих даних

Із розвитком інформаційних технологій, в результаті поширення соціальних мереж, сервісів обміну відео і аудіо інформацією, появі концепцій інтернету речей – потреби в обробці великих масивів даних безперервно ростуть. Зростання обсягу інформації супроводжується появою нових програмних та апаратних засобів, здатних виконувати їх обробку. У результаті такого зростання потреби зберігати і працювати з великими обсягами даних і появи технічних засобів з'явився окремий вид послуг, що отримав назву Big Data (Великі дані) [1]. Концепція великих даних почала розвиватись на початку 2000-х, коли аналітик Дуг Лейні (Doug Laney) сформулював визначення основних даних великих даних як три V: Volume – обсяг даних, Velocity – швидкість їх надходження та Variety – різноманітність інформації [2].

Найпопулярнішим на даний час інструментом для вирішення проблем у сфері великих даних є проект Apache Hadoop. До його складу входить набір утиліт, бібліотек та фреймворк з відкритою ліцензією для розробки і виконання розподілених програм, адаптованих для виконання на кластерах з великою кількістю вузлів.

Особливостями побудови інфраструктур для опрацювання великих даних є потреба недорогого зберігання великих об'ємів інформації. Для цього застосовуються архітектури розподіленого зберігання з горизонтальним масштабуванням, наприклад, на основі HDFS. Для великих даних є потреба використовувати сервери з великим об'ємом локальних дисків, оперативної пам'яті і числом процесорів. Потреби до такої інфраструктури постійно ростуть, оскільки неперервно збільшуються запити бізнесу до результатів, швидкості роботи, об'ємів даних і технологій що використовуються. Це робить інфраструктуру складною, дорогою і вимагає кваліфікованих працівників для її підтримки. Для розгортання інфраструктур опрацювання великих даних може бути доцільним використання моделі хмарних обчислень.

1.2. Огляд моделі хмарних обчислень

Хмарними обчисленнями називається модель надання мережевого доступу до спільного середовища комп'ютерних ресурсів, доступного для налаштування (мережі, сервери, сховища даних, сервіси і додатки). Такий доступ передбачає, що ресурси можуть бути швидко наданими і звільненими при мінімальних зусиллях для конфігурації та взаємодії з провайдером послуг. Використання такого підходу звільняє від необхідності будувати і підтримувати власні дата центри, сервери і від витрат пов'язаних з цими операціями.

Існує декілька типів хмарних обчислень: публічні хмари, приватні хмари та гібридні хмари.

Публічною хмарою називається такий тип хмарних обчислень, при якому постачальник послуг надає свої послуги (окремі програми, віртуальні машини, сховища даних) для користувачів через публічний інтернет. Такі ресурси можуть бути доступними для безкоштовного використання, оплачуватися за моделлю підписки чи згідно до фактичного об'єму використаної послуги. При використанні публічної хмари постачальник є власником усіх ресурсів, він керує та здійснює підтримку датацентрів, апаратного забезпечення та інфраструктури на якій працює користувач. У публічній хмарі ви використовуєте одне і те ж апаратне забезпечення, сховища та мережеві пристрої разом з іншими організаціями, "орендаторами" ("tenants") хмари. Доступ до керування обліковим засобом, створення та конфігурації ресурсів надається через використанням веб-інтерфейсу та програмного (API) інтерфейсу. Лідерами на ринку надання публічних хмарних ресурсів є Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure.

Перевагами використання публічних хмар нижча вартість у порівнянні з використанням власних комп'ютерних ресурсів. Кінцевому користувачу не потрібно купувати апаратне та програмне забезпечення, він платить тільки за сервіси якими користується. Користувач звільняється від необхідності здійснювати підтримку інфраструктури – постачальник послуг відповідає за технічне обслуговування ресурсів що використовуються. Постачальники послуг володіють мережею дата-

центрів розподілених по усьому світу, що відкриває для користувачів майже необмежену масштабованість, для них ці ресурси доступні за вимогою. Провайдери хмарних послуг надають можливості для конфігурації відмовостійкості та високої доступності ресурсів.

Приватною хмарою називається інфраструктура, ресурси якої використовується тільки для певної організації. Такі ресурси можуть фізично знаходитись у датацентрі, що належить організації або надаватись у користування третьою стороною. У приватній хмарі сервіси та інфраструктура завжди працюють у межах приватної мережі, а апаратне та програмне забезпечення виділяється тільки для конкретної установи. Використання приватних хмар дозволяє більш гнучке налаштування ресурсів для досягнення особливих вимог. Приватні хмари часто використовуються державними та фінансовими установами, організаціями середнього та великого розміру, що прагнуть мати повний контроль над своїм середовищем.

До переваг використання приватних хмар відноситься більша гнучкість налаштування у порівнянні з публічними. Оскільки ресурси є ізольованими в межах організації, це дозволяє досягнути більшого рівня контролю та безпеки у порівнянні з публічними хмарами. Модель забезпечує більшу масштабованість у порівнянні з використанням власних локальних ресурсів компанії.

Гібридною хмарою називається тип хмарних обчислень, який передбачає комбінацію локальної інфраструктури чи приватних хмар у поєднанні з публічними хмарами. Гібридні хмари дозволяють мігрувати дані та робоче навантаження між двома середовищами. Гібридна хмарна платформа надає організаціям переваги гнучкості, більше можливостей розгортання, безпеку, та забезпечує повноцінне використання існуючої інфраструктури. При зростанні навантаження на інфраструктуру, гібридні хмарні обчислення дозволяють масштабувати свої ресурси у публічну хмару, і таким чином уникнути перевантаження локальних ресурсів. Організації отримують гнучкість, яку надає публічна хмара, при цьому маючи можливість зберігати чутливі дані у власному центрі обробки даних для задоволення потреб клієнта або вимог регуляторних органів.

1.3. Аналіз можливих способів побудови інфраструктури для опрацювання великих даних у приватних хмарах

Для побудови інфраструктури опрацювання великих даних у приватній хмарі спочатку потрібно визначитися з тим, яка реалізація хмари буде використовуватись. Варіанти реалізації приватної хмари поділяються на пропрієтарні та рішення з відкритим кодом. Зокрема, розробники дистрибутивів операційної системи Linux, наприклад, Red Hat, SUSE та Ubuntu розповсюджують свої рішення для побудови приватних хмар на основі проекту з відкритим кодом OpenStack.

Платформа OpenStack – це рішення з відкритим кодом, яке використовує поширені механізми віртуалізації для побудови та керування приватними хмарами. Проект складається з набору компонентів, які разом надають функціонал хмарної платформи: створення віртуальних комп'ютерних ресурсів, мереж для їх взаємодії, сховищ даних, а також програмні та веб-інтерфейси для ідентифікації та роботи користувачів. OpenStack підтримує популярні механізми віртуалізації: KVM, Xen, VMware vSphere, Hyper-V, LXC, Docker. Перевагою використання проекту є легкість автоматизації: проект постачається з CLI для налаштування та роботи з ресурсами та має програмний інтерфейс API, що дозволяє розробляти власні інструменти для роботи з хмарою. OpenStack має велику кількість користувачів по всьому світу, над його розвитком працює понад 4000 розробників. Проект є готовим до використання – будь-хто може запустити OpenStack, але для його коректного налаштування, повноцінної реалізації усіх функцій потрібні кваліфіковані спеціалісти з досвідом.

До недоліків використання підходу створення власної приватної хмари на основі OpenStack відноситься складність його розгортання. Складність пов'язана з тим, що платформа складається з великого набору компонентів, кожен з яких конфігурується окремо відповідно до вимог користувача. Для автоматизації встановлення OpenStack існує проект openstack-ansible, що дозволяє автоматизувати встановлення компонентів системи за допомогою Ansible. Слабким місцем проекту OpenStack можна назвати якість документації. Через те, що платформа складається з

більше ніж 25 різних проектів, та не існує одного, спільного джерела документації, якість її покриттям варіюється в залежності до конкретного проекту.

Існують також варіанти реалізації приватних хмар з використанням механізмів віртуалізації на обладнані від певних постачальників. До таких пропрієтарних рішень можна віднести продукт Hewlett Packard Enterprise під назвою Helion Cloud Suite. Компанія VMware реалізує віртуалізацію за допомогою vSphere та пропонує Cloud Foundation Software-Defined Data Center для приватних хмар. Dell EMC – сервіс для організації приватних хмар, а також продукт для керування та забезпечення їх безпеки. Компанія Oracle пропонує своє рішення Private Cloud Appliance X8 для реалізації функцій комп'ютерних обчислень та сховища даних для приватних хмар. IBM пропонує апаратне забезпечення для приватних хмар та IBM Cloud Managed Services, інструменти для забезпечення безпеки та оркестрації елементів хмари.

Після реалізації інфраструктури приватної хмари наступним етапом є створення віртуальних машин, конкретних вузлів де, будуть запускатися обчислювальні завдання та налаштування необхідного для роботи ПЗ. Для розгортання кластера Apache Hadoop необхідно, щоб віртуальні машини працювали під управлінням операційних систем з сімейства Linux. Для роботи Hadoop на кожній машині повинна бути встановлені програмні компоненти для запуску Java додатків. Для встановлення та налаштування також необхідно забезпечити доступ по SSH до вузлів кластера та між ними. Після цього виконується завантаження бінарних файлів, налаштування конфігурації та виконуються скрипти встановлення кластера.

Використання Apache Hadoop у тому вигляді, в якому він надається Apache Software Foundation має кілька вагомих недоліків: за-замовчуванням його встановлення потребує попередньої конфігурації машин, ручного встановлення пакетів, редагування багатьох файлів конфігурації. Тому, на практиці частіше всього використовують дистрибутиви від приватних компаній, наприклад, Cloudera CDH і Hortonworks HDP.

Cloudera Distribution including Apache Hadoop (CDH) – зв'язка найбільш популярних інструментів з інфраструктури Hadoop під керуванням Cloudera Manager. Cloudera Manager має функціонал за розгортання кластера як на локальних, так і у

хмарних середовищах (Rackspace, Amazon EC2, Softlayer). Також він містить утиліти і можливості конфігурації автоматизації збірки засобами Apache Maven. Крім цього, Cloudera Manager включає в себе засоби для моніторингу над компонентами кластера, відстеження та аналізу ефективності виконання завдань, створення сповіщень, щодо стану працездатності інфраструктури та програмних компонентів.

Hortonworks Data Platform (HDP) – дистрибутив Hadoop компанії від компанії Hortonworks. Особливістю продукту цієї компанії є те, що замість розробки власних компонентів вона вкладається у розвиток відкритих рішень Apache Foundation. Для встановлення, налаштування та керування роботою кластера у HDP використовується Apache Ambari. Ambari надає веб та API інтерфейси для роботи з кластером. Дистрибутив містить додаткові рішення Apache: Zookeeper, Oozie для планування і координування розподіленої обробки завдань; підтримує інтеграцію з Kerberos, KNOX, Ranger для забезпечення безпеки. Дистрибутив підтримує інтеграцію з реляційними та NoSQL базами даних Hive, HCatalog, HBase, Apache Solr, що використовуються для зберігання та аналізу журналу роботи кластера.

1.4. Аналіз можливих способів побудови інфраструктури для опрацювання великих даних у публічних хмарах

1.4.1. Використання моделі PaaS для реалізації інфраструктури опрацювання великих даних.

У популярних публічних хмарах доступні реалізації функціоналу платформи Hadoop, доступ до яких надається за моделлю PaaS (Platform as a service). Використання моделі передбачає те, що усі ресурси, у тому числі обчислювальні мережі, сервери, сховища даних повністю керуються постачальником. Кінцевому користувачеві надається можливість використовувати функціонал платформи та встановлювати, розробляти і запускати прикладне програмне забезпечення, маючи можливість динамічно змінювати кількість обчислювальних ресурсів. Використання моделі PaaS дозволяє значно спростити трудозатратні задачі по створенню, налаштуванню кластерів Hadoop та керуванням ними і обчислювальними потужностями, на яких вони працюють. Перевагою також є майже миттєва

доступність новостворених ресурсів платформи. До таких PaaS рішень, зокрема, відносяться продукти Amazon EMR і Google Dataproc.

Amazon EMR використовує середовище Hadoop, яке знаходиться в інфраструктурі Amazon EC2 та сховище даних Amazon S3. З допомогою Amazon EMR можна миттєво виділити будь-яку кількість ресурсів, необхідних для обчислень. Після завершення роботи, якщо користувачем не заданий інший варіант, кластер автоматично припиняє роботу, тому не потрібно платити за ресурси, які вже не використовуються. Функціонал Amazon EMR базується на використанні продуктів Apache; Spark, Hive, HBase, Flink, Hudi та Presto [3]. На відміну від інфраструктури з використанням локальних кластерів, EMR розділяє ресурси обчислення та зберігання даних, що дає можливість масштабувати кожен елемент незалежно та використовувати можливості різних типів зберігання Amazon S3. Інфраструктура EMR легко масштабована, кількість ресурсів можна автоматично змінювати залежно від навантаження, і, відповідно, оплата знімається тільки за використані ресурси. Для моніторингу ресурсів EMR використовуються ті ж засоби Amazon, що і для інших типів ресурсів. Платформа підтримує функціонал автоматичної заміни обчислювальних машин, що погано працюють та відновлення при втраті вузлів кластера. Постачальник послуг також забезпечує оновлення програмних компонентів до останніх стабільних версії компонентів, тому користувачеві не потрібно керувати оновленнями, та виправляти помилки у них. При використанні Amazon EMR користувач має повний контроль над кожною інстанцією кластера. Є можливість використовувати користувацькі образи операційної системи, встановлювати додаткове програмне забезпечення на етапі ініціалізації кластера.

Google Dataproc – платформа, що надається Google Cloud Platform і дозволяє запускати Apache Spark та Hadoop кластери у хмарні інфраструктурі [4]. Google Dataproc інтегрується з іншими сервісами Google Cloud Platform. Це означає те, що користувач отримує повноцінну платформу з можливістю використовувати сервіси моніторингу Cloud Monitoring, логування Cloud Logging, масштабовану NoSQL базу даних Cloud Bigtable, хмарне сховище для даних Cloud Storage, сервіс для інтерактивного широкомасштабного аналізу великих наборів даних BigQuery [5].

До складу Google Dataproc входять наступні компоненти платформи Apache Hadoop: Spark, Hive, Pig, Tez, Druid, HBaze, Hive WebHCat, Jupyter Notebook, Kerberos, Presto, Zookeeper, мови програмування Python і Scala. При створенні кластера є можливість додати додаткові, програмні продукти, існують опції щодо конфігурації апаратних характеристик вузлів кластеру, їх високої доступності та відмовостійкості.

1.4.2. Використання моделі SaaS для реалізації інфраструктури опрацювання великих даних. SaaS (Software as a service) – модель доступу до хмарних ресурсів, при якій користувачу надається у використання програмне забезпечення, робота якого повністю обслуговується постачальником. Доступ користувачів до програми реалізовано за допомогою веб інтерфейсу і програмного інтерфейсу API. Обслуговування та оновлення такого ПЗ відбувається постачальником, вартість таких послуг включена в орендну плату.

Для роботи з великими даними компанія Amazon надає наступні SaaS продукти: Athena, Glue, Kinesis, сховище об’єктів Amazon S3.

Amazon Athena – сервіс для виконання інтерактивних запитів, який дозволяє аналізувати дані, що знаходяться у сховищі S3 з використанням стандартного SQL. Сервіс є високодоступним, для виконання запитів використовує розгалужену інфраструктуру Amazon у різних географічних локаціях. Запити Athena автоматично розпаралелюються щоб забезпечити максимальну швидкість обчислення результату.

AWS Glue – служба для виконання ETL операцій (extract, transform, load) яка дозволяє готувати дані до аналізу. Її особливістю є можливість створити ETL завдання у візуальному редакторі.

Amazon Kinesis – продукт для збирання, обробки та аналізу потокових даних в реальному часі. З використанням Kinesis можна обробляти відео, аудіо, інформацію, телеметрію і логи роботи програм та проводити їх аналіз.

До SaaS рішень для роботи з великими даними від Google можна віднести наступні продукти: BigQuery, Dataflow, Cloud Data Fusion.

Google BigQuery – безсерверне сховище даних від Google, яке дозволяє виконувати SQL запити та інтерактивний аналіз над великими наборами даних. Рішення підтримує інтеграцію між різними хмарами, аналітику над потоковими

даними в реальному часі. Для виконання запитів використовується ANSI:2011 SQL, надаються драйвери ODBC та JDBC для інтеграції з сторонніми додатками. BigQuery підтримує роботу із зовнішнім сховищем об'єктів Cloud Storage (файли у форматах Parquet та ORC), інтеграцію з SQL базами даних (Bigtable, Cloud SQL). Така робота відбувається без переміщення даних у сховище. BigQuery може виконувати Spark, TensorFlow, Dataflow, Apache Beam, MapReduce, Pandas задачі, використовуючи Storage API.

Cloud Data Fusion – рішення для інтерактивного створення ETL/ELT процесів без написання коду, побудоване на проекті з відкритим кодом CDAP. Постачається з бібліотекою 150+ передналаштованих конекторів до джерел даних та трансформаторів, що виконують перетворення даних. Продукт підтримує можливість створення власної бібліотеки конекторів та трансформаторів з використанням популярних мов програмування.

Dataflow – сервіс для потокової та послідовної (stream and batch) обробки даних з використанням Apache Beam SDK. Dataflow автоматично масштабує необхідну для роботи кількість віртуальних машин, для виконання конкретного завдання.

Dataprep – сервіс для візуального дослідження, очищення та підготовки структурованих і неструктурованих наборів даних для аналізу, побудови звітів та машинного навчання.

1.5. Проблеми, які виникають при розгортанні інфраструктури опрацювання великих даних

Розглянутим варіантам побудови інфраструктури для опрацювання великих даних у приватних та публічних хмарах властиві певні недоліки. Говорячи про приватні хмари, використання описаних способів для розгортання, налаштування та підтримки інфраструктури опрацювання великих даних вимагає великих затрат часу та значної кількості фахівців. Системні адміністратори повинні конфігурувати апаратне забезпечення для досягнення потрібної продуктивності комп'ютерних ресурсів і мережі. Після цього вручну встановлюються та налаштовуються операційні

системи, програмні компоненти та бібліотеки потрібні для роботи обчислювальних програм. Тільки після того, як проведені усі підготовчі кроки, спеціалісти готові для розгортання самого кластера Hadoop. Виконання усіх цих ручних процесів часто приводить до виникнення певних проблем.

Першою проблемою є вартість інфраструктури. Компанії повинні наймати спеціалістів для кожного етапу налаштування інфраструктури, від мережевих інженерів до техніків, які займаються підтримкою апаратного забезпечення. Усім цим людям потрібно оплачувати роботу, ефективно ставити завдання та контролювати їх виконання. Це веде до накладних витрат, ускладнення структури та комунікації в межах організації. Значним компонентом витрат є купівля і оновлення апаратного забезпечення потрібного для функціонування інфраструктури.

Наступною великою проблемою є масштабованість та доступність інфраструктури. При виникненні неочікуваного навантаження на ресурси, оскільки ручне конфігурування є повільним, можуть виникати проблеми у доступності ресурсів. Якщо організація не має у запасі серверів, на які можна розподілити навантаження, додаток може бути недоступним довгий період.

Також великою проблемою є непостійність при ручному конфігуруванні ресурсів. Редагування конфігураційних файлів, використання інтерактивних інструментів може призвести до виникнення непередбачуваних проблем, які важко знайти та відлагодити. Якщо ж в організації працює команда людей, що вручну конфігурують ресурси, уникнути розбіжностей пов’язаних з людським фактором стає ще важче.

Традиційні підходи до розгортання інфраструктури не надають можливості тестування функціоналу перед внесенням змін і повторного використання напрацювань.

Таким чином, для побудови інфраструктури системи опрацювання великих даних потрібно забезпечити можливості:

- масштабованості і високої доступності;
- доступності рішення за ціною;
- оптимізації витрат на управління персоналом;

- швидкого циклу розробки і впровадження змін;
- тестування перед внесенням змін у функціонал;
- повторного використання напрацювань;
- документованості створеного вирішення, керування його версіями;
- усунення конфліктів конфігурації при роботі в команді.

Для вирішення проблем, які стосуються масштабованості та доступності інфраструктури доцільно застосовувати підхід використання публічних та гібридних хмарних обчислень. Таке рішення не є панацеєю, вимагає наявності спеціалістів, проте дозволяє розгортати інфраструктуру швидко, і, при правильній конфігурації вирішує проблеми доступності та масштабованості.

Для вирішення проблеми непослідовності при конфігурації, можливості пришвидшення циклу розробки інфраструктури і впровадження до неї змін застосування знаходить підхід під назвою «Інфраструктура як код». «Інфраструктура як код» – це підхід, який передбачає створення і керування обчислювальними ресурсами за допомогою їх опису у програмному коді [6]. До переваг, які надає використання даного підходу відноситься швидкість створення, послідовність конфігурування, підзвітність описаної системи. Підхід дозволяє описати всю інфраструктуру за допомогою файлу, де міститься опис бажаного стану системи. Цей файл можна застосувати для розгортання окремих ізольованих середовищ роботи – від тестового до робочого, маючи можливість легко змінювати параметри конфігурації. Наприклад, для тестування використовувати менш потужні, дешевші машини, зберігаючи ідентичні налаштування мережі і доступу. Використання IaC дозволяє забезпечити послідовність у процесі конфігурації системи і уникнути помилок конфігурації пов’язаних з людським фактором. Оскільки конфігураційні файли є єдиним джерелом налаштувань, ви гарантуєте, що однакові конфігурації будуть розгортатися кожного разу, уникнувши можливих розбіжностей. Також підхід дозволяє мати звіт про те, хто і коли вносив зміни до конфігурації інфраструктури, оскільки файли зберігаються у системі контролю версій, як і при розробці програмного забезпечення.

При використанні «Інфраструктура як код» ефективність продемонстрували підходи із сфери розробки програмного забезпечення: неперервна інтеграція та неперервне розгортання (Continuous integration & Continuous delivery, CI/CD). Головні цілі впровадження CI/CD – звести до мінімуму помилки, пришвидшити збирання і підвищити якість кінцевого продукту. При використанні CI/CD тестування коду проводиться всякий раз, коли у нього вносяться зміни [7]. Виявлення помилок на ранній стадії розробки суттєво економить час і ресурси команди, тому що чим пізніше виявляється помилка програми, тим важче і дорожче її виправити. Що стосується пришвидшення, то автоматизація CI/CD дозволяє ефективно оптимізувати всі рутинні процеси збирання програми. В методології CI/CD існують три основні підходи, які використовують окремо або поєднують, в залежності від стратегії. Перший з них називається неперервною інтеграцією. Він передбачає, що зміни у програмний код вносяться регулярно, декілька разів на день. Інструменти CI, дозволяють створювати скрипти для автоматичного запуску збирання і тестування коду, після кожного завантаження у змін. Спочатку перевірку проходить нова зміна окремо, після чого вона вноситься у код попередньої версії ПЗ, яке тестується вже повністю. Іншими варіантами є неперервна доставка та неперервне розгортання. Після того, як зміни внесені і код протестований, ПЗ можна розгортати – запускати не на тестовому, а на робочих серверах. Неперервна доставка означає розгортання коду після кожної нової інтеграції, при цьому вона відбувається у ручному режимі. Неперервне розгортання – повністю автоматичний процес, який проходить без контролю команди [8].

1.6. Порівняння інструментів IaC

Інструменти для реалізації концепції IaC поділяються на декілька категорій:

- одноразові скрипти (Ad hoc scripts);
- інструменти керування конфігурацією (configuration management tools);
- інструменти шаблонування серверів (server templating tools);
- інструменти для оркестрації (orchestration tools);

– інструменти надання ресурсів (provisioning tools).

Ідея використання Ad hoc скриптів полягає у розбитті ручної процедури конфігурації на дискретні кроки, запису їх з використанням скриптової мови програмування (Bash, Ruby, Python) та виконанням на цільовому комп’ютері. Описаний підхід передбачає написання власного, кожного разу унікального коду на мовах загального призначення і може бути використаним для маленького завдання, але не підходить для підтримки комплексної інфраструктури.

Інструменти керування конфігурацією (Chef, Puppet, Ansible, і SaltStack) призначені для встановлення і налаштування програмного забезпечення на існуючих машинах. Такі інструменти дозволяють працювати зі структурованим кодом, що легко читається. Перевагою їх використання перед одноразовими скриптами є можливість реалізації ідемпотентного коду, який коректного працює незважаючи на те, скільки разів він виконувався. Такі інструменти спеціально розробляються для можливості легкої роботи з великою кількістю машин на відміну від одноразових скриптів, чия ніша – запуск на одній, локальній машині.

Альтернативою інструментам керування конфігурацією є рішення для створення шаблонів серверів (Docker, Packer, Vagrant). Інструменти шаблонування реалізують ідею створення образів серверів, які зберігають у собі знімок операційної системи, прикладного програмного забезпечення та інших файлів потрібних для роботи. Для розгортання таких образів на усі потрібні машини можна використати інші інструменти IaC. Шаблонування серверів – основний компонент для реалізації підходу незмінної (immutable) інфраструктури. Ця ідея бере початок у програмуванні де, якщо незмінному об’єкту призначити певне значення, то воно не може бути зміненими. Якщо потрібно оновити таке значення, то створюється новий об’єкт. Ідея незмінної інфраструктури схожа – після розгортання сервера з ним не виконується ніяких дій. Для внесення змін створюється новий образ з потрібними правками і розгортається на новий сервер. Оскільки в конфігурацію не вносяться зміни завжди можна бути впевненим про те, які версії програм і з якими налаштуваннями там виконується.

Інструменти оркестрації (Kubernetes, Marathon/Mesos, Amazon Elastic Container Service (Amazon ECS), Docker Swarm, Nomad) призначені для керування роботою кластерів віртуальних машин і контейнерів. Вони дозволяють розгортати такі кластери, поширювати оновлення, контролювати стан системи, масштабувати кількість ресурсів, керувати мережевим навантаженням та взаємодією між серверами.

Інструменти надання ресурсів (Terraform, CloudFormation, OpenStack Heat) призначені власне для створення потрібних комп'ютерних ресурсів. Вони відповідають не тільки за запуск серверів, але й за створення баз даних, балансувальників навантаження, черги, конфігурацію мережі та мережевих фільтрів та усі інші аспекти комп’ютерної інфраструктури.

Процес вибору необхідного IaC є комплексною задачею. Багато з них мають ідентичний функціонал, але можуть використовувати різні принципи, володіти нюансами реалізації. Для оптимального вибору, який не приведе до непередбачуваних наслідків і не потребуватиме додаткових витрат часу на дослідження особливостей потрібно вибирати інструмент треба під конкретну задачу, розуміючи різницю між ними [9].

IaC рішення поділяються на рішення з відкритим вихідним кодом та пропрієтарні. З наведених раніше продуктів CloudFormation і Deployment Manager є пропрієтарними інструментами Amazon та Google відповідно, і працювати тільки з власними хмарами. Критерій відкритості впливає на те, чи може кінцевий користувач вносити зміни у функціонал інструменту, наскільки швидко буде додаватись потрібний спільноті функціонал та виправлятися виявлені проблеми.

Вагомою відмінністю між згаданими інструментами є те, що Ansible, Chef і SaltStack є інструментами для керування конфігурацією, тобто призначені для встановлення та налаштування програмного забезпечення. Terraform, Deployment Manager, CloudForamtion – інструменти для створення ресурсів інфраструктури. Цей поділ не є категоричним, тому що більшість інструментів створення інфраструктури мають функціонал їх налаштування. Засоби керування конфігурацією часто підтримують створення певних ресурсів у хмарах. Проте, розробці кожного з них

закладалися певні акценти, тому конкретні інструменти краще виконують конкретні завдання.

IaC інструменти можуть працювати за однією з двох парадигм: змінної або незмінної інфраструктури. Інструменти керування конфігурацією за-замовчуванням реалізовують змінну інфраструктуру. Вони виконують зміни, які визначив користувач, не гарантуючи конкретний стан системи, а просто виконуючи їх крок за кроком. Наприклад, якщо потрібно оновити певний програмний пакет, у ході такого процесу не відслідковується стан системних файлів, робота інших програм. З часом це може привести до того, що на кожному сервері буде унікальна історія змін. У свою чергу це може викликати помилки конфігурації які важко діагностувати і відтворити. Декларативний підхід передбачає при внесенні кожної зміни у конфігурацію створення нової інстанції машини і видалення старої. Це дозволяє бути впевненим, що усі сервери містять ідентичні налаштування та версії програмних продуктів. За потреби можна легко повернути стан системи до попереднього.

Інструменти для опису інфраструктури у коді поділяють на такі, що використовують процедурний і декларативний стилі. Процедурний стиль є характерним для Ansible та Chef. При його використанні потрібна для досягнення результату послідовність дій описується крок-за-кроком. Використовуючи декларативний стиль CloudFormation, Terraform, Puppet, користувач вказує бажаний фінальний стан системи, а за його досягнення відповідає вже сам інструмент. За використання процедурного підходу система у конкретний момент часу може не відповідати описаному коду, оскільки тут також важливий порядок застосування змін. Декларативний підхід забезпечує повну відповідність коду та розгорнутої інфраструктури.

IaC інструменти також відрізняються тим, чи потрібен їм для роботи окремий мастер сервер. SaltStack, Puppet і Chef вимагають такого сервера. Він використовується для зберігання інформації про поточний стан інфраструктури і для поширення змін на агенти. Функціонал мастер сервера постійно відслідковує стан системи і може скасувати внесення змін, яке виконалось вручну.

Такі засоби як Chef, SaltStack, Puppet для роботи вимагають встановлення агента на цільову машину. Недоліком такого підходу є потреба попередньо доставити таке ПЗ, відслідковувати його стан та стежити за оновленнями програми-агента. Наявність такого компонента впливає на безпеку системи через необхідність відкривати мережевий доступ до ресурсів ззовні, крім того, агент сам може бути вразливим для зловмисників місцем.

Порівняння інструментів для опису інфраструктури за розглянутими критеріями наведено у табл. 1.1.

Таблиця 1.1

Порівняння характеристик інструментів IaC

Назва	Код	Хмара	Тип	Парадигма	Тип мови	Агент	Мастер
Chef	Відкритий	Усі	Керування конфігурацією	Змінна	Процедурна	Ні	Так
Puppet	Відкритий	Усі	Керування конфігурацією	Змінна	Декларативна	Ні	Так
Ansible	Відкритий	Усі	Керування конфігурацією	Змінна	Процедурна	Ні	Ні
SaltStack	Відкритий	Усі	Керування конфігурацією	Змінна	Декларативна	Так	Так
Cloud Formation	Закритий	AWS	Створення ресурсів	Незмінна	Декларативна	Ні	Ні
Deployment Manager	Закритий	GCP	Створення ресурсів	Незмінна	Декларативна	Ні	Ні
Heat	Відкритий	Open Stack	Створення ресурсів	Незмінна	Декларативна	Ні	Ні
Terraform	Відкритий	Усі	Створення ресурсів	Незмінна	Декларативна	Ні	Ні

Важливими характеристиками для вибору конкретного інструменту IaC також є розмір спільноти і зрілість технології. Дані критерії мають безпосередній вплив на те, наскільки детальною і вичерпною є документація на конкретний інструмент, чи доступні сторонні інтеграції та розширення, як легко знайти спеціалістів знайомих з технологією. Для вибору інструменту, особливо якщо це стосується рішень з відкритим кодом, варто врахувати активність розробки продукту, активність користувачів щодо повідомлень про виявлення помилок та шляхи їх вирішення. Оцінюючи зрілість технології, здійснено порівняння дат виходу першої версії

інструменту та номер поточної версії. Для порівняння розміру спільноти користувачів конкретного інструменту було оцінено кількість контриб'юторів проекту на GitHub, кількість запитань на StackOverflow, а також як багато доступно відкритих бібліотек і модулів. Оцінка зрілості та розмірів спільноти інструментів IaC наведено у табл. 1.2.

Таблиця 1.2

Порівняння розмірів спільноти та зрілості для інструментів IaC

Назва	Дата виходу	Поточна версія	Контриб'ютори	Бібліотеки	Розмір спільноти	Зрілість технології
Chef	2009	16.6	636	3958	Велика	Висока
Puppet	2005	7.0	547	6110	Велика	Висока
Ansible	2012	2.9	5156	26692	Дуже велика	Середня
SaltStack	2011	3000.5	2288	346	Велика	Середня
Cloud Formation	2011	2020-11-24	-	560	Мала	Середня
Deployment Manager	2015	2020-04-15	-	35	Мала	Низька
Heat	2012	15.0.0-34	372	29	Мала	Низька
Terraform	2014	0.13.5	1524	4548	Дуже велика	Низька

На практиці, зазвичай, доводиться використовувати одночасно декілька інструментів для побудови своєї інфраструктури. Найпоширеніші комбінації спільного застосування інструментів:

- Розгортання інфраструктури та керування конфігурацією.
- Розгортання інфраструктури та створення шаблону сервера.
- Розгортання, шаблонування і оркестрація.

Поєднання інструментів для створення інфраструктури та керування конфігурацією дозволяє використати один інструмент для розгортання всієї базової інфраструктури, включаючи топологію мережі, сховища і бази даних, сервери. Потім застосовується інший для розгортання та налаштування своїх програм на цих машинах. Прикладом є поєднання Terraform та Ansible. Наприклад, Terraform може додавати спеціальні теги на створені ним сервери, а потім Ansible їх використає для пошуку потрібних машин та роботи з ними.

Прикладом поєднання інструментів розгортання інфраструктури та створення шаблонів є використання Terraform та Packer. Можна використати Packer, щоб упакувати свої програми як образи віртуальних машини. Після цього застосовується Terraform для розгортання серверів на основі цих образів та решти інфраструктури.

Прикладом комбінації з використанням інструментів розгортання, шаблонування та оркестрації є застосування Terraform, Packer, Docker та Kubernetes для створення Kubernetes кластера і запуску додатків на ньому. За допомогою Packer створюється образ віртуальної машини, на якій встановлені Docker та Kubernetes. Потім використовується Terraform для розгортання кластера серверів, на кожному з яких запускається цей образ. Нарешті, коли усі сервери завантажуються, вони об’єднуються у кластер Kubernetes, на якому запускаються користувацькі програми.

1.7. Висновки до розділу

У розділі здійснено огляд предметної області великих даних та особливостей, характерних для побудови інфраструктури для їх обробки. Розглянуто модель хмарних обчислень та можливих способів побудови інфраструктури для опрацювання великих даних з її використанням. Описано проблеми, які виникають при розгортанні інфраструктури для опрацювання великих даних та підходи, що допомагають їх мінімізувати. Сформовано критерії, характерні для побудови оптимальної інфраструктури опрацювання великих даних. Здійснено порівняння інструментів для опису інфраструктури у коді.

РОЗДІЛ 2
ВИБІР СЕРЕДОВИЩА ДЛЯ РОЗГОРТАННЯ ІНФРАСТРУКТУРИ ОПРАЦЮВАННЯ ВЕЛИКИХ ДАНИХ

2.1. Вибір оптимального варіанту середовища для розгортання інфраструктури опрацювання великих даних

Для вибору оптимального варіанту розгортання інфраструктури опрацювання великих даних у хмарних сервісах з урахуванням типових вимог замовника було використано метод аналізу ієрархій (MAI) Томаса Л. Сааті. Основою методу є розбиття проблеми на прості складові, для яких поетапно встановлюються пріоритети з використанням попарних порівнянь. Метод MAI знайшов застосування у широкому спектрі галузей, особливо при груповому прийнятті рішень [10].

Першим етапом використання MAI є структурування проблеми вибору у вигляді ієрархії або мережі. Побудову ієрархії починають з вершини, яка виступає у ролі цілі дослідження, через проміжні рівні, тобто критерії оцінювання. Останній і найнижчий рівень – набір альтернатив, які потрібно оцінити. Для нашого випадку ціллю дослідження виступає вибір оптимального варіанту для розгортання інфраструктури опрацювання великих даних. Вибрано наступні критерії оцінювання:

- вартість інфраструктури;
- можливості масштабованості;
- можливість реалізації високої доступності;
- гнучкість налаштування та можливість внесення змін;
- швидкість розгортання і внесення змін до стану інфраструктури;
- документованість, можливість повторного використання рішення;
- приватність даних.

Розглянуті альтернативи розгортання інфраструктури опрацювання великих даних у середовищах:

- на власних ресурсах (On-premise);

- приватної хмари;
- приватної хмари з використанням IaC;
- публічної хмари за моделлю PaaS та використанням IaC;
- публічної хмари за моделлю SaaS та використанням IaC;
- гібридної хмари.

Наступним етапом дослідження є встановлення пріоритетів для критеріїв і оцінка кожної з альтернатив за ними. При використанні MAI елементи дослідження оцінюються попарно за відношенням їх впливу на спільну характеристику. Система парних порівнянь приводить до результату, який може бути представлений у вигляді обернено-симетричної матриці. Елементом матриці a_{ij} є інтенсивність впливу елементу ієрархії i відносно елементу j, що оцінюється за шкалою інтенсивності від 1 до 9, запропонованою Т. Сааті (табл. 2.1).

Таблиця 2.1

Шкала оцінювання парних порівнянь

Оцінка у балах	Характеристика оцінки	Коментар
1	рівні за важливістю елементи	вклад обох елементів оцінювання однаковий
3	помірна перевага одного елемента над іншим	існує легка перевага одного елементу за іншим
5	суттєва перевага	існує сильна перевага одного елементу над іншим
7	значна перевага	очевидна перевага одного елементу над іншим
9	абсолютна перевага	перевагу одного елементу над іншим не піддається сумніву
2, 4, 6, 8	проміжні оцінки між двома твердженнями	компромісне рішення

Результати порівнянь формують матрицю, у якій занесено порівняння відносної важливості елементів рядка з елементами стовбця. Якщо при порівнянні одного фактору i з іншим j перший є важливішим, то у табл. 2.2 заносилося ціле число b, якщо навпаки – $1/b$.

Таблиця 2.2

Матриця порівнянь критеріїв оцінювання

i	Критерій	Номер стовбця, j							W_i	НО	λ_{max}
		1	2	3	4	5	6	7			
1	Вартість	1,00	2,00	2,00	3,00	4,00	1,00	2,00	1,92	24,50 %	1,000
2	Масштабованість	0,50	1,00	1,00	1,00	5,00	0,33	2,00	1,08	13,73 %	1,194
3	Висока доступність	0,50	1,00	1,00	2,00	3,00	1,00	2,00	1,29	16,48 %	1,044
4	Гнучкість розгортання	0,33	1,00	0,50	1,00	3,00	1,00	1,00	0,91	11,56 %	1,079
5	Швидкість розгортання	0,25	0,20	0,33	0,33	1,00	0,33	1,00	0,41	5,19 %	1,039
6	Приватність даних	1,00	3,00	1,00	1,00	3,00	1,00	3,00	1,60	20,44 %	1,022
7	Документованість	0,50	0,50	0,50	1,00	1,00	0,33	1,00	0,64	8,10 %	0,973
Сума		4,08	8,7	6,3	9,3	20	5	12	7,83	100%	7,35

Власне значення матриці, λ_{max}	7,35
Індекс узгодженості, ІУ	0,0584256
Усереднений індекс узгодженості, УІУ	1,32
Відношення узгодженості, ВУ	0,0442 = 4%

У результаті попарних порівнянь отримано матрицю порівнянь, де по головній діагоналі знаходяться одиниці, елементи $a_{j,i} = \frac{1}{a_{i,j}}$, $a_{j,k} = \frac{a_{i,k}}{a_{i,j}}$.

Визначення вагових коефіцієнтів попарних порівнянь використовується для оцінки пріоритетів критеріїв, так само як і для оцінки альтернатив за результатами попарних порівнянь. Значенням вагових коефіцієнтів є вектор пріоритетів W_i, який обчислюється як нормоване середнє геометричне порівнянь кожного рядка:

$$W_i = \frac{\sqrt[n]{\prod_{j=1}^{n} a_{i,j}}}{\sum_{i=1}^{n} \sqrt[n]{\prod_{j=1}^{n} a_{i,j}}}.$$

Нормована оцінка (НО) для елементу v_i вектору пріоритету рівна:

$$v_i = \frac{w_i}{\sum_{i=1}^{n} w_i}.$$

Сума нормованих оцінок повинна бути рівною 1 або 100%.

Оскільки MAI належить до методів експертного дослідження, додатково обчислюється узгодженість оцінок методу. Спочатку знаходяться суми за всіма стовбцями матриці:

$$B = \{b_{i_2}\},\ i_2 = \overline{1, M},\ b_{i_2} = \sum_{i_1=1}^{M} a_{i_1 i_2}.$$

Далі обчислюється власне значення матриці λ_{max}:

$$\lambda_{\max} = \left(b_1, b_2, \ldots, b_{i_2}, \ldots, b_M\right) \times \begin{pmatrix} w_1 \\ w_2 \\ \cdots \\ w_i \\ \cdots \\ w_M \end{pmatrix}.$$

Індекс узгодженості експертної оцінки (ІУ) обчислюється як:

$$\text{ІУ} = \frac{\lambda_{max} - n}{n - 1},$$

де n – розмірність матриці.

Відношення узгодження (ВУ) отримаємо, якщо поділити ІУ на число, що відповідає усередненому індексу узгодженості (УІУ) матриці того ж порядку. Значення (УІУ) для матриць різних розмірів наведено у табл. 2.3.

Таблиця 2.3

Значення усередненої узгодженості для матриць різних порядків

Розмір матриці (n)	3	4	5	6	7	8	9	10
УІУ	0,58	0,9	1,12	1,24	1,32	1,41	1,45	1,49

Величина відношення узгодження повинна бути меншою за 10%, у окремих випадках дозволяється значення не більше за 20%. Якщо значення більше за критичне – проводиться перегляд експертних оцінок.

Для розглянутого порівняння критеріїв оцінювання отримані наступні значення якості експертних оцінок:

- Власне значення матриці, λ_{max}= 7,836.
- Індекс узгодженості, ІУ = 0,0584256.
- Відношення узгодженість, ВУ = 0,0442 = 4%.

Наступний етап проведення дослідження – оцінювання альтернативних варіантів середовищ для опрацювання великих даних. Його виконання передбачає визначення векторів пріоритетів альтернатив щодо усіх критеріїв, а також вектора глобальних пріоритетів, який і використовується для вибору оптимального рішення. Кращим варіантом вважається альтернатива з найбільшим показником пріоритету.

Оцінювання альтернатив за критерієм «Вартість інфраструктури» наведено у табл. 2.4. З результатів дослідження видно, що найкращий результат характерний для варіанту розгортання у середовищі публічної хмари з використанням моделі PaaS. Показник відносної узгодженості оцінок експерта становить 11%.

Таблиця 2.4

Матриця порівнянь альтернатив за критерієм «Вартість»

№	Альтернатива	Матриця парних порівнянь альтернатив						W_i	НО	λ_{max}
		1	2	3	4	5	6			
1	Власні ресурси	1,00	0,25	0,20	0,11	0,33	0,14	0,253	2,59%	0,752
2	Приватна хмара	4,00	1,00	2,00	0,14	6,00	0,20	1,054	10,79%	1,501
3	Приватна хмара, IaC	5,00	0,50	1,00	0,17	6,00	0,20	0,890	9,12%	1,310
4	Публічна хмара PaaS	9,00	7,00	6,00	1,00	8,00	3,00	4,566	46,76%	0,878
5	Публічна хмара SaaS	3,00	0,17	0,17	0,13	1,00	0,17	0,346	3,55%	0,970
6	Гібридна хмара	7,00	5,00	5,00	0,33	6,00	1,00	2,654	27,18%	1,280
Сума		29,00	13,92	14,37	1,88	27,33	4,71	9,766	100,00%	6,693

λ_{max}	6,693
ІУ	0,1387555
УІУ	1,24
ВУ	0,1118 = 11,19%

Найгіршим відношенням вартості характеризуються використання власних ресурсів для розгортання платформи опрацювання великих даних, показник нормованої оцінки якого рівний 2,59%.

Матриця порівнянь альтернатив за критерієм «Масштабованість інфраструктури» наведено у табл. 2.5. Зважаючи на отримані значення нормованої оцінки для критерію найкращими характеристиками володіє варіант розгортання інфраструктури у публічній хмарі з використанням моделі SaaS. Найменше значення – у варіанту розгортання на власних ресурсах. Значення нормованої оцінки для даного порівняння становить 12,95%.

Таблиця 2.5

Матриця порівнянь альтернатив за критерієм «Масштабованість»

№	Альтернатива	Матриця парних порівнянь альтернатив						W_i	НО	λ_{max}
		1	2	3	4	5	6			
1	Власні ресурси	1,00	0,33	0,33	0,11	0,11	0,20	0,25	2,38%	0,71
2	Приватна хмара	3,00	1,00	1,00	0,11	0,11	0,14	0,42	3,89%	1,06
3	Приватна хмара, IaC	3,00	1,00	1,00	0,11	0,11	0,14	0,42	3,89%	1,06
4	Публічна хмара PaaS	9,00	9,00	9,00	1,00	0,33	5,00	3,27	30,46%	1,38
5	Публічна хмара SaaS	9,00	9,00	9,00	3,00	1,00	7,00	4,98	46,47%	0,84
6	Гібридна хмара	5,00	7,00	7,00	0,20	0,14	1,00	1,38	12,90%	1,74
Сума		30,00	27,33	27,33	4,53	1,81	13,49	10,72	100,00%	6,80

λ_{max}	6,8029511
ІУ	0,1605902
УІУ	1,24
ВУ	0,1295 = 12,95%

Порівняння альтернатив за критерієм «Висока доступність» наведено у табл. 2.6. Як видно з отриманих значень, найвищий показник доступності забезпечують два варіанти: використання публічної хмари за моделлю доступу SaaS та PaaS. Найменше значення – при використанні власних локальних ресурсів. Значення нормованої оцінки для даного порівняння становить 11,48%.

Таблиця 2.6

Матриця порівнянь альтернатив за критерієм «Висока доступність»

№	Альтернатива	Матриця парних порівнянь альтернатив						W_i	НО	λ_{max}
		1	2	3	4	5	6			
1	Власні ресурси	1,00	0,14	0,14	0,11	0,11	0,20	0,19	1,80%	0,68
2	Приватна хмара	7,00	1,00	1,00	0,11	0,11	0,20	0,51	4,76%	1,20
3	Приватна хмара, IaC	7,00	1,00	1,00	0,11	0,11	0,20	0,51	4,76%	1,20
4	Публічна хмара PaaS	9,00	9,00	9,00	1,00	1,00	7,00	4,15	38,86%	0,96
5	Публічна хмара SaaS	9,00	9,00	9,00	1,00	1,00	7,00	4,15	38,86%	0,96
6	Гібридна хмара	5,00	5,00	5,00	0,14	0,14	1,00	1,17	10,95%	1,71
Сума		38,00	25,14	25,14	2,48	2,48	15,60	10,68	100,00%	6,71

λ_{max}	6,7115326
ІУ	0,1423065
УІУ	1,24
ВУ	0,11476 = 11,48%

Порівняння альтернатив за критерієм «Гнучкість налаштування» наведено у табл. 2.7. Як видно з отриманих значень, найвищий показник гнучкості дозволяє досягти розгортання інфраструктури на власних ресурсах. Найменшу гнучкість дає використання моделі SaaS. Значення якості експертної оцінки для даного порівняння становить 14,17%.

Таблиця 2.7

Матриця порівнянь альтернатив за критерієм «Гнучкість налаштування»

№	Альтернатива	Матриця парних порівнянь альтернатив						W_i	НО	λ_{max}
		1	2	3	4	5	6			
1	Власні ресурси	1,00	3,00	3,00	5,00	9,00	4,00	3,43	39,69%	0,88
2	Приватна хмара	0,33	1,00	1,00	4,00	8,00	5,00	1,94	22,47%	1,25
3	Приватна хмара, IaC	0,33	1,00	1,00	4,00	8,00	5,00	1,94	22,47%	1,25
4	Публічна хмара PaaS	0,20	0,25	0,25	1,00	9,00	0,50	0,62	7,17%	1,15
5	Публічна хмара SaaS	0,11	0,13	0,13	0,11	1,00	7,00	0,33	3,85%	1,35
6	Гібридна хмара	0,25	0,20	0,20	2,00	0,14	1,00	0,38	4,36%	0,98
Сума		2,23	5,58	5,58	16,11	35,14	22,50	8,64	100,00%	6,88

λ_{max}	6,8785074
ІУ	0,1757015
УІУ	1,24
ВУ	0,1416947 = 14,17%

Матриця порівняння альтернатив розгортання за критерієм «Швидкість розгортання та внесення змін» наведено у табл. 2.8. Згідно обрахунків, перевагою у порівнянні володіє інфраструктура, що використовує модель хмарних послуг SaaS. Найгірший показник швидкості – при використання власних ресурсів та приватної хмари без IaC. Показник нормованої оцінки експерта становить 9,61%.

Таблиця 2.8

Матриця порівнянь альтернатив за критерієм «Швидкість розгортання»

№	Альтернатива	Матриця парних порівнянь альтернатив						W_i	НО	λ_{max}
		1	2	3	4	5	6			
1	Власні ресурси	1,00	0,50	0,20	0,13	0,13	0,20	0,26	2,73%	0,79
2	Приватна хмара	2,00	1,00	0,13	0,11	0,11	0,13	0,27	2,82%	1,00
3	Приватна хмара, IaC	5,00	8,00	1,00	0,14	0,13	1,00	0,95	9,89%	1,71
4	Публічна хмара PaaS	8,00	9,00	7,00	1,00	0,50	2,00	2,82	29,52%	1,14
5	Публічна хмара SaaS	8,00	9,00	8,00	2,00	1,00	3,00	3,89	40,69%	0,89
6	Гібридна хмара	5,00	8,00	1,00	0,50	0,33	1,00	1,37	14,35%	1,05
Сума		29,00	35,50	17,33	3,88	2,19	7,33	9,56	100,00%	6,60

λ_{max}	6,5959409
ІУ	0,1191882
УІУ	1,24
ВУ	0,0961195 = 9,61%

Наступний проаналізований критерій – «Приватність даних», дані щодо якого показані у табл. 2.9. Це оцінювання говорить про те, що найвищу конфіденційність даних забезпечує інфраструктура, побудована з використання власних ресурсів. Показник ВУ при цьому оцінюванні становить 11,1%.

Таблиця 2.9

Матриця порівнянь альтернатив за критерієм «Приватність даних»

№	Альтернатива	Матриця парних порівнянь альтернатив						W_i	НО	λ_{max}
		1	2	3	4	5	6			
1	Власні ресурси	1,00	3,00	3,00	4,00	9,00	4,00	3,30	38,86%	0,89
2	Приватна хмара	0,33	1,00	1,00	3,00	7,00	4,00	1,74	20,51%	1,17
3	Приватна хмара, IaC	0,33	1,00	1,00	3,00	7,00	4,00	1,74	20,51%	1,17
4	Публічна хмара PaaS	0,25	0,33	0,33	1,00	9,00	0,20	0,61	7,14%	1,15
5	Публічна хмара SaaS	0,11	0,14	0,14	0,11	1,00	0,13	0,18	2,09%	0,86
6	Гібридна хмара	0,25	0,25	0,25	5,00	8,00	1,00	0,92	10,88%	1,45
Сума		2,28	5,73	5,73	16,11	41,00	13,33	8,50	100,00%	6,69

λ_{max}	6,6926612
ІУ	0,1385322
УІУ	1,24
ВУ	0,1117195 = 11,1%

Щодо оцінювання документованості і можливості повторного використання рішення, однакові оцінки отримали усі варіанти, що передбачають використання методу «інфраструктура як код». Матриця альтернатив за цим критерієм наведено у табл. 2.10.

Таблиця 2.10

Матриця порівнянь альтернатив за критерієм «Документованість рішення»

№	Альтернатива	Матриця парних порівнянь альтернатив						W_i	НО	λ_{max}
		1	2	3	4	5	6			
1	Власні ресурси	1,00	0,33	0,11	0,11	0,11	0,20	0,21	2,25%	0,81
2	Приватна хмара	3,00	1,00	0,11	0,11	0,11	0,20	0,31	3,24%	1,08
3	Приватна хмара, IaC	9,00	9,00	1,00	1,00	1,00	5,00	2,72	28,80%	0,99
4	Публічна хмара PaaS	9,00	9,00	1,00	1,00	1,00	5,00	2,72	28,80%	0,99
5	Публічна хмара SaaS	9,00	9,00	1,00	1,00	1,00	5,00	2,72	28,80%	0,99
6	Гібридна хмара	5,00	5,00	0,20	0,20	0,20	1,00	0,76	8,10%	1,33
Сума		36,00	33,33	3,42	3,42	3,42	16,40	9,44	100,00%	6,18

λ_{max}	6,1750861
ІУ	0,0350172
УІУ	1,24
ВУ	0,0282397 = 2,82%

Для розрахунку вектору глобальних пріоритетів нормовані оцінки кожного з оцінювань та числове значення локальних векторів пріоритетів перенесені у табл. 2.11. Значення глобального пріоритету обчислюється як сума добутків значення вектору пріоритету для критерію і значення вектору локального пріоритету даної альтернативи.

Таблиця 2.11

Обчислення вектора глобальних пріоритетів

Альтернативи	Критерії							Глобальний вектор
	Вартість	Масштабованість	Висока доступність	Гнучкість налаштування	Швидкість розгортання	Приватність	Документованість	
	Числове значення вектору пріоритету							
	0,245	0,137	0,165	0,116	0,052	0,204	0,081	
Власні ресурси	0,026	0,024	0,018	0,397	0,027	0,389	0,022	0,141
Приватна хмара	0,108	0,039	0,048	0,225	0,028	0,205	0,032	0,112
Приватна хмара, IaC	0,091	0,039	0,048	0,225	0,099	0,205	0,288	0,132
Публічна хмара PaaS	0,468	0,305	0,389	0,072	0,295	0,071	0,288	0,282
Публічна хмара SaaS	0,035	0,465	0,389	0,038	0,407	0,021	0,288	0,190
Гібридна хмара	0,272	0,129	0,109	0,044	0,144	0,109	0,081	0,144

Зважаючи на отримане значення глобального вектору, найкращим варіантом для розгортання платформи для опрацювання великих даних за сукупністю розглянутих критеріїв є використання публічної хмари за моделлю надання послуг PaaS.

2.2. Висновки до розділу

За допомогою методу аналізу ієрархій проаналізовано альтернативні варіанти розгортання платформ для опрацювання великих даних. Враховуючи критерії вартості, доступності, масштабованості інфраструктури, гнучкості налаштування, приватності даних, документованості і можливості повторного використання напрацювань оптимальним варіантом було вибрано використання приватної хмари за моделлю доступу до ресурсів PaaS.

РОЗДІЛ 3
РОЗРОБКА КОДУ ОПИСУ ІНФРАСТРУКТУРИ ДЛЯ ОПРАЦЮВАННЯ ВЕЛИКИХ ДАНИХ

3.1. Розробка коду для опису інфраструктури

Перед безпосередньою розробкою програмного коду для опису інфраструктури було створено блок-схему алгоритму роботи системи, наведену у додатку Б. Згідно неї, після початку роботи відбувається відстеження змін коду, що зберігається у репозиторії. Якщо виявлені зміни – виконуємо етап планування, виводимо зміни, які плануються до внесення та очікуємо вказівки користувача. Якщо користувач підтверджує внесення змін, то виконуємо їх, якщо ж ні – ігноруємо їх та чекаємо на наступну зміну стану коду. При виникненні помилки вона буде виводитись, а ще не внесені зміни – ігноруватися.

Для проведення дослідження по автоматизації розгортання платформи для опрацювання великих даних Google Dataproc вибрано інструмент опису інфраструктури у коді – Terraform. Terraform – програмний продукт з відкритим кодом, призначений для опису інфраструктури як коду. Його використання дозволяє користувачеві описувати та використовувати інфраструктуру постачальників хмарних послуг за допомогою декларативної мови конфігурації високого рівня HCL. У ході виконання Terraform генерує план-файл, в якому описуються кроки, які будуть виконані для досягнення бажаного стану інфраструктури, а потім виконує його для створення описаних. У міру внесенні змін до конфігурації Terraform може визначати, що змінилося, і створювати додаткові плани виконання. Terraform будує граф залежностей для ресурсів і проводить паралельно створення та модифікацію будь-яких незалежних ресурсів. Через це створення інфраструктури відбувається максимально ефективно, а користувачі отримують представлення про існуючі зв’язки в інфраструктурі [14].

Для забезпечення роботи кластера Dataproc потрібно створити наступні ресурси:

– приватну віртуальну мережу для зв’язку між усіма комп’ютерними ресурсами;

– правила для налаштування мережевого екрану;

– сховище даних для зберігання конфігураційних файлів кластера та результатів виконання обчислень;

– кластер обчислювальних ресурсів.

Для створення віртуальної приватної мережі Google Cloud ми оголошуємо тип ресурсу — google_compute_network, задаємо ім’я та вказуємо параметр auto_create_subnetworks, який дозволяє створити підмережі в автоматичному режимі, як показано на рис. 3.1. Підмережі будуть створені в кожному доступному географічному регіоні у мережі 10.128.0.0/9.

```
resource "google_compute_network" "dataproc_network" {
  name                    = "${var.gcp_project}-dataproc-network"
  auto_create_subnetworks = true
}
```

Рис. 3.1. Лістинг коду для створення віртуальної мережі

Для налаштування мережевого фільтру використовується ресурс google_compute_firewall, як показано на рис. 3.2. Описані правила використовуються у тестових цілях і дозволяють отримати доступ до усіх TCP і UDP портів, а також по протоколу ICMP з будь-якої публічної адреси.

```
resource "google_compute_firewall" "dataprocopen" {
  name    = "${var.gcp_project}-dataprocopen"
  network = google_compute_network.dataproc_network.name

  allow {
    protocol = "icmp"
  }

  allow {
    protocol = "tcp"
    ports    = ["1-65535"]
  }
  allow {
    protocol = "udp"
    ports    = ["1-65535"]
  }
```

Рис. 3.2. Лістинг налаштування мережевого фільтру

Для створення файлового сховища необхідно оголосити ресурс google_storage_bucket, лістинг коду для роботи з яким показано на рис. 3.3. Оскільки усі ресурси типу bucket на платформі GCP повинні мати унікальне ім'я, створимо його, використавши ресурс random_id, що генерує унікальну послідовність символів при створенні. Задаємо локацію для ресурсу сховища як "EU", що дозволяє оголосити ресурс як мультирегіональний, з розташуванням файлів у датацентрах Європейського Союзу. Параметр force_destroy говорить про те, що ми хочемо при видалені сховища також видалити усі файли що, там знаходяться.

```
resource "random_id" "name_suffix" {

  byte_length = 4
}

resource "google_storage_bucket" "dataproc_bucket" {

  name          = "dataproc-bucket-${random_id.name_suffix.hex}"
  location      = "EU"
  force_destroy = "true"
}
```

Рис. 3.3. Лістинг ресурсів для створення сховища даних

Перед безпосереднім створенням обчислювального кластеру потрібно налаштувати необхідні для його функціювання API інтерфейси. Для налаштування API використовуємо ресурс google_project_service, як показано на рис. 3.4.

```
resource "google_project_service" "services" {
  count              = length(var.project_services)
  project            = var.gcp_project
  service            = element(var.project_services, count.index)
  disable_on_destroy = false
}

variable "project_services" {
  default = [
    "cloudresourcemanager.googleapis.com",
    "cloudbilling.googleapis.com",
    "iam.googleapis.com",
    "compute.googleapis.com",
    "oslogin.googleapis.com",
    "storage-api.googleapis.com",
    "dataproc.googleapis.com",
    "servicenetworking.googleapis.com"
  ]
}
```

Рис. 3.4. Лістинг активації API інтерфейсів необхідних для роботи

На вхід цього блоку передається кількість окремих API, яка обчислюється за допомогою функції length() автоматично, відповідно вмісту змінної project_services. Також задаємо змінну, що містить назву поточного проекту GCP, вказуємо назву конкретного API зі змінної project_services. Параметр disable_on_destroy говорить про те, що не слід вимикати сервіси після видалення ресурсу.

Значення змінних, що будуть використовуватися при створенні кластера, визначається в окремому файлі, як показано на рис. 3.5.

```
variable "master_num_instances" {
  default = 1
}
variable "master_machine_type" {
  default = "n1-standard-2"
}
variable "boot_disk_size" {
  default = 30
}
variable "boot_disk_type" {
  default = "pd-ssd"
}
```

Рис. 3.5. Лістинг оголошення змінних

Така конфігурація говорить про те, що у кластері буде одна мастер інстанція типу n1-standard-2 — два віртуальних процесора, 7.5 GB оперативної пам’яті. Тип інстанції також визначає максимальну пропускну здатність мережі для даної машини. В нашому випадку вона рівна 10 Gbps. Розмір диску описаної машини становить 30 GB, тип SSD.

Для створення обчислювального кластера Dataproc використовується ресурс google_dataproc_cluster, як показано на рис. 3.6. Для його оголошення задаємо йому ім’я і регіон, в якому будуть розміщені мастер та робочі інстанції. Також потрібно вказати параметр staging_bucket, в який у нашому випадку передаємо назву раніше створеного сховища. Даний бакет буде використовуватись для обміну файлами між інстанціями кластера, збереження тимчасових файлів та результатів виконання обчислювальних завдань. Конфігурація мастер нод кластера відбувається у блоці

коду master_config. Тут задано кількість таких інстанцій, їх тип — конфігурацію процесора та оперативної пам’яті, обсяг та тип дискового сховища.

```
resource "google_dataproc_cluster" "mycluster" {
  name   = "mycluster"
  region = var.gcp_region
  labels = {
    env = "testing"
  }
  cluster_config {
    staging_bucket = google_storage_bucket.dataproc_bucket.name
    master_config {
      num_instances = var.master_num_instances
      machine_type  = var.master_machine_type
      disk_config {
        boot_disk_type    = var.boot_disk_type
        boot_disk_size_gb = var.boot_disk_size
      }
    }
    worker_config {
      num_instances = var.worker_num_instances
      machine_type  = var.worker_machine_type
      disk_config {
        boot_disk_size_gb = var.boot_disk_size
        num_local_ssds    = 1
      }
    }
    preemptible_worker_config {
      num_instances = var.preemptible_worker_num_instances
    }
```

Рис. 3.6. Лістинг ресурсу кластера Google Dataproc

Конфігураційний блок worker_config визначає характеристики обчислювальних одиниць кластера. Задані у ньому параметри відповідають тим же, як у конфігурації master, окрім параметру num_local_ssds, що підключає до кожної машини локальний SSD диск розміром 375 GB. Перевага використання таких дисків полягає в тому, що local ssd фізично з’єднані з сервером, на якому працює віртуальна машина. Інші типи дисків є мережевими, тому мають довший час затримок. Параметр preemptible_worker_config.num_instances дозволяє вказати бажану кількість обчислювальних машин типу preemptible — дешевших, але з недовгим часом доступності. Їх використання підходить для певних типів обчислювальних завдань, які швидко виконуються.

На рис. 3.7 наведено продовження лістингу конфігурації кластера. Блок під назвою software_config використовується для налаштування програмного

забезпечення кластера Dataproc. Параметр image_version вказує версію операційної системи кластера, від якої також залежить наявність певних прикладних програмних продуктів. У нашій конфігурації цей параметр рівний 1.5-debian10. У блоці gce_cluster_config вказано мережеві теги, що присвоюються ресурсам кластера, саму мережу у якій вони працюють та список Google Cloud API, який доступний кожного вузла кластера.

```
software_config {
  image_version = var.image_version
  override_properties = {
    "dataproc:dataproc.allow.zero.workers" = "true"
  }
}

gce_cluster_config {
  tags    = ["testing", "dataproc"]
  network = google_compute_network.dataproc_network.name
  service_account_scopes = [
    "https://www.googleapis.com/auth/monitoring",
    "useraccounts-ro",
    "storage-rw",
    "logging-write",
  ]
}
```

Рис. 3.7. Лістинг налаштування програмного забезпечення кластера

Також налаштування кластера передбачає використання декількох блоків initialization_action, як показано на рис. 3.8. Їх оголошення говорять про те, що при ініціалізації кластера потрібно виконати встановлення наступні програмні компонентів: Stackdriver Monitoring Agent, Apache ZooKeeper, Apache Livy, Kafka. Параметр depends_on використовується для того, щоб процес створення кластера розпочинався тільки після завершення налаштування GCP сервісів, та створення файлового сховища.

```
    initialization_action {
      script      = "gs://dataproc-initialization-
actions/stackdriver/stackdriver.sh"
      timeout_sec = 500
    }
    initialization_action {
      script      = "gs://dataproc-initialization-
actions/zookeeper/zookeeper.sh"
      timeout_sec = 5000
    }
    initialization_action {
      script      = "gs://dataproc-initialization-
actions/livy/livy.sh"
      timeout_sec = 500
    }
    initialization_action {
      script      = "gs://dataproc-initialization-
actions/kafka/kafka.sh"
      timeout_sec = 500
    }
  }
  depends_on = [google_storage_bucket.dataproc_bucket,
google_project_service.services]
  timeouts {
    create = "30m"
    delete = "30m"
  }
}
```

Рис. 3.8. Лістинг встановлення додаткових компонентів

Для демонстрації роботи кластера будуть запущені два обчислювальних завдання, що перевіряють функціонал фреймворків Apache Hadoop та Spark і також оголошені кодом, як показано на рис. 3.9 та 3.10.

```
resource "google_dataproc_job" "hadoop" {
  region       = google_dataproc_cluster.mycluster.region
  force_delete = true
  placement {
    cluster_name = google_dataproc_cluster.mycluster.name
  }
  hadoop_config {
    main_jar_file_uri = "file:///usr/lib/hadoop-mapreduce/hadoop-
mapreduce-examples.jar"
    args = [
      "wordcount",
      "file:///usr/lib/spark/NOTICE",

"gs://${google_dataproc_cluster.mycluster.cluster_config.0.bucket}/had
oopjob_output",
    ]
  }
}
```

Рис. 3.9. Лістинг запуску обчислювального з використанням фреймворку Hadoop

Як видно на рис. 3.10, для запуску обчислювального завдання потрібно передати регіон та назву кластера, де проходитимуть обчислення. Для конфігурації конкретних завдань слід вказати шлях до виконуваних файлів у файловій системі, та необхідні параметри. Проведене тестування передбачає виконання програми Hadoop Wordcount для розподіленого обчислення кількості повторень слів у заданому текстовому файлі та SparkPi, що обчислює значення числа Pi з використанням Spark.

```
resource "google_dataproc_job" "spark" {
  region       = google_dataproc_cluster.mycluster.region
  force_delete = true
  placement {
    cluster_name = google_dataproc_cluster.mycluster.name
  }
  spark_config {
    main_class    = "org.apache.spark.examples.SparkPi"
    jar_file_uris = ["file:///usr/lib/spark/examples/jars/spark-
examples.jar"]
    args          = ["1000"]
    properties = {
      "spark.logConf" = "true"
    }
    logging_config {
      driver_log_levels = {
        "root" = "INFO"
      }
    }
  }
```

Рис. 3.10 Лістинг обчислювального завдання SparkPi

3.2. Розгортання розробленої інфраструктури

Для роботи з Terraform існує декілька основних команд: terraform init, terraform apply, terraform destroy. Команда terraform init використовується для ініціалізації робочої директорії що містить конфігураційні файли Terraform. Дана команда є першою, що повинна виконуватись після розробки нової Terraform конфігурації або завантаження з системи контролю версій. Команда terraform plan виконується для створення плану виконання. Terraform виконує оновлення та визначає які дії потрібно виконати для досягнення стану системи, що вказаний у файлах конфігурації. Команда використовується для перевірки чи внесені користувачем зміни є коректними, не

вносячи змін до реального стану інфраструктури. Команда terraform apply використовується для внесення бажаних змін до стану інфраструктури відповідно до файлів конфігурації.

У результаті виконання Terraform будує граф залежності ресурсів. Для описаного проекту він наведений у додатку В. Використання графу дозволяє візуально переглянути залежності між окремими ресурсами у проекті, виявити можливі помилки, зациклення процесу створення ресурсів. У нашому випадку ми можемо виявити потенційну проблему у описаній конфігурації – між налаштуванням мережевого екрану і створенням ресурсу кластера немає прямої залежності, теоретично їх створення може відбуватись паралельно і це приведе до помилок доступу. Оскільки створення кластера виконується на порядок довше, і на встановлення зв’язку між компонентами задано певний поріг часу, то таких ситуацій на практиці не виникає.

Для реалізації підходів CI/CD при роботі з Terraform було використано сервіс Terraform Cloud. Цей засіб дозволяє виконувати Terraform у надійному і послідовному середовищі, включає засоби налаштування доступу до файлів стану, керування правами окремих користувачів та команд. Використання підходів «Інфраструктура як коду» та CI/CD вимагають, щоб конфігурації зберігалися у системах контролю версій. Terraform Cloud підтримує інтеграцію з такими засобами. Зокрема, кожен робочий простір може бути прив’язаним до окремої вітки чи коміту системи контролю версій git. Terraform Cloud автоматично отримує файли конфігурації зі сховища та відслідковує за змінами у ньому. При появі нових комітів, пов’язані з ними робочі простори запускають планування змін до стану інфраструктури (виконання описаної раніше команди terraform plan), що відповідає парадигмі Continuous Integration. При виконанні pull чи merge запитів, пов’язані робочі простори спекулятивно виконують планування. Член команди, що відповідає за підтвердження змін бачить, чи виконання було успішним, які конкретні кроки будуть виконані. Після цього він вручну підтверджує виконання плану – якщо команда використовує Continuous Delivery. Також є можливість налаштувати роботу сервісу таким чином, що зміни у

стан інфраструктури будуть вноситись після безпосереднього внесення коміту, без ручного підтвердження – як реалізація Continuous Deployment.

У результаті проведення дослідження конфігураційні файли з налаштуванням кластеру Google Dataproc поміщені у систему контролю версій git. З даним репозиторієм інтегрований сервіс Terraform Cloud, де і буде проходити виконання інструменту Terraform та керування ним. Створене під проект робоче середовище показане на рис. 3.11.

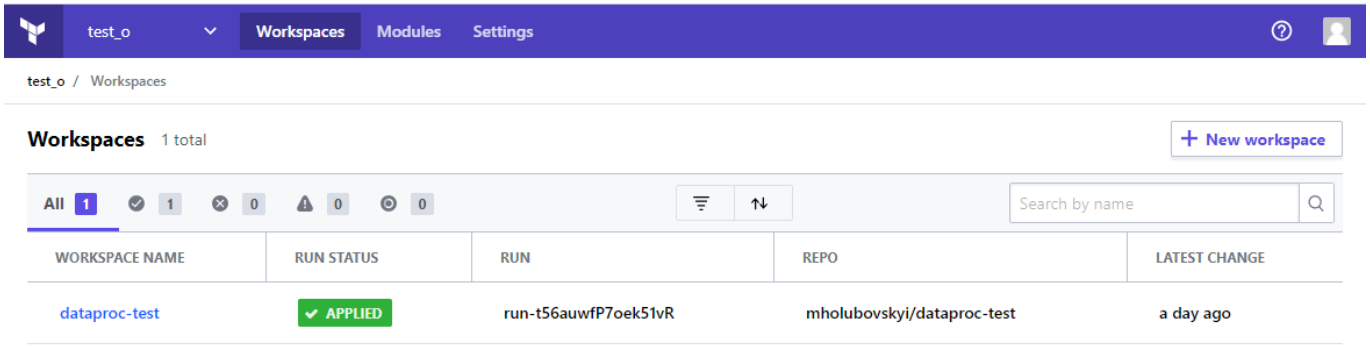


Рис. 3.11. Робоче середовище для запуску Terraform-коду

Для того, щоб запустити виконання Terraform потрібно створити коміт зі змінами до стану системи чи натиснути кнопку “Queue plan”. У результаті цих дій буде запущено процес створення плану змін, як показано на рис. 3.12. Після цього є можливість підтвердити зміни – натиснути кнопку “Confirm & Apply”, відмінити їх – “Discard Run” та додати коментар за допомогою “Add Comment”.

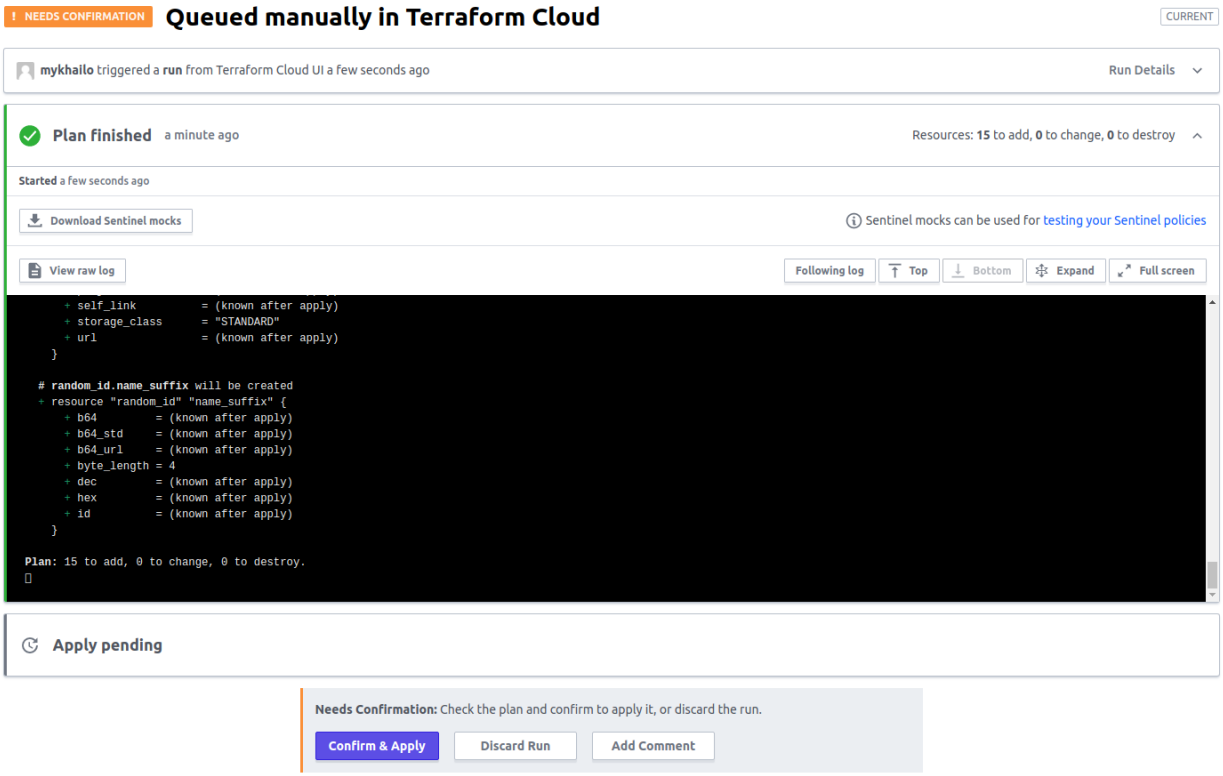


Рис. 3.12. Ініціалізація процесу планування внесення змін у стан інфраструктури

Після перегляду того, які зміни у стан інфраструктури планують підтверджуємо їх. Консольний вивід процесу впровадження змін показано на рис. 3.13.

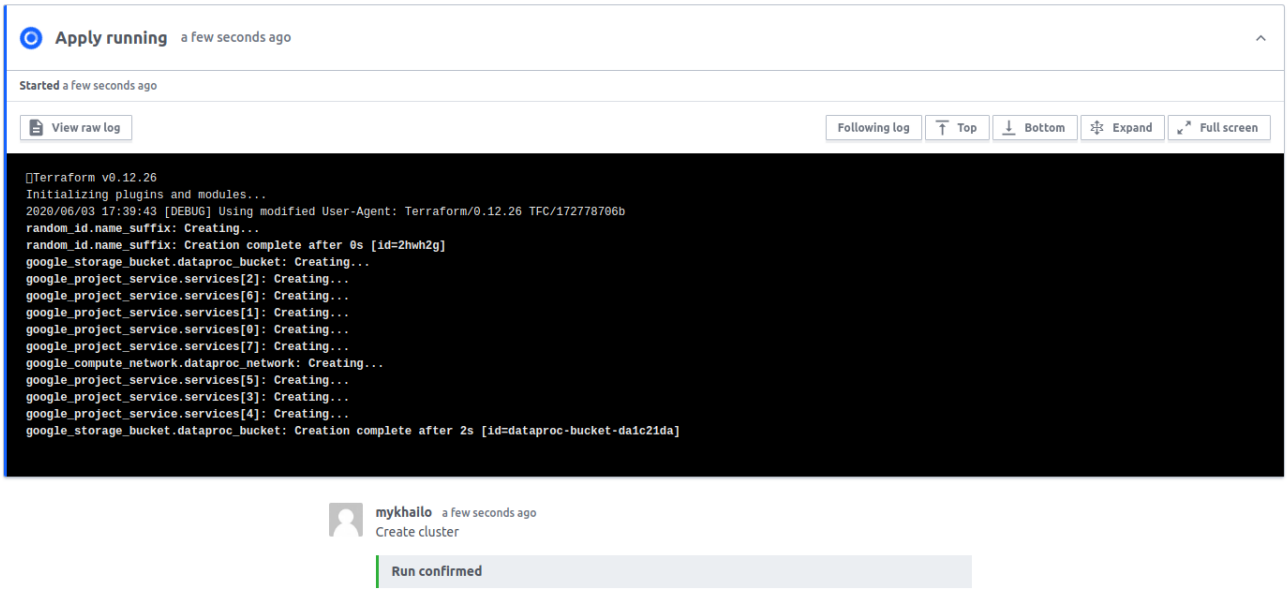


Рис. 3.13. Впровадження змін у стан інфраструктури

За умови успішного виконання виводиться інформація про це та повідомлення про кількість внесених змін. У нашому випадку додатково передбачено повідомлення про статус обчислювальних робіт haddopjob_status та spark_status, як показано на рис. 3.14.

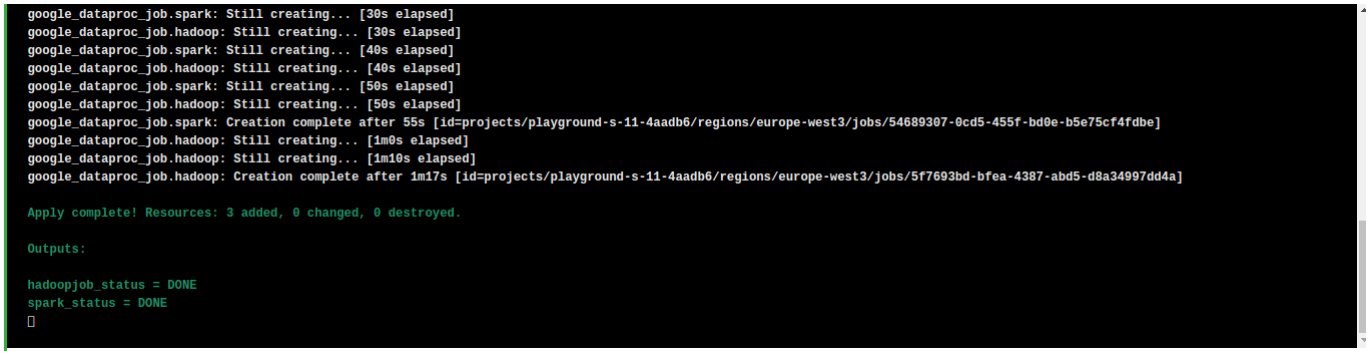


Рис. 3.14. Повідомлення про успішне виконання

У разі виникнення помилок terraform виводить назву помилки, назву ресурсу, з яким вона пов’язана та можливі способи її усунення. Після успішного створення ресурсів можна пересвідчитись у їх роботоздатності, відкривши, наприклад, меню VM instances у веб-інтерфейсі GCP. Тут ми бачимо, що для потреб кластера було створено чотири віртуальні машини, як і було вказано у наших конфігураційних файлах – рис. 3.15.

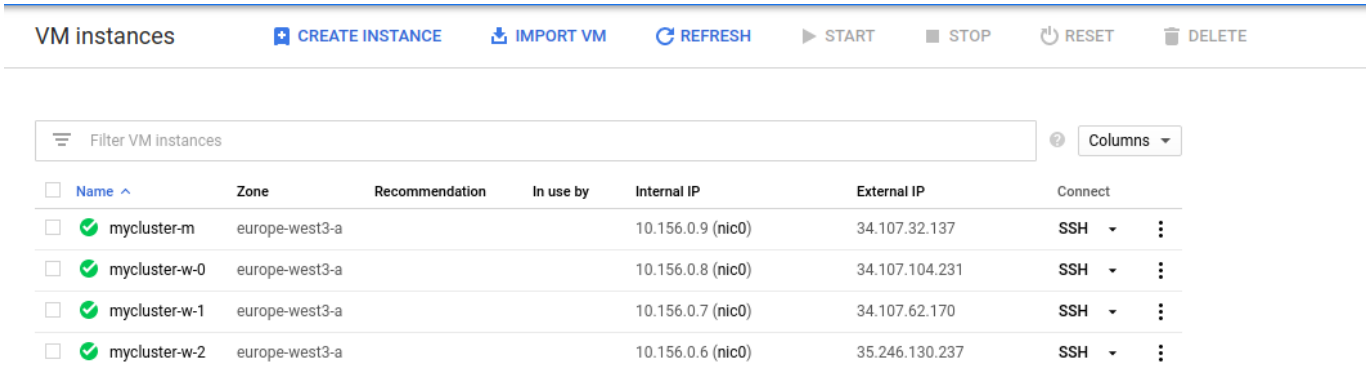


Рис. 3.15. Перелік створених обчислювальних ресурсів

Переглянути статус роботи кластера Dataproc можна в однойменному меню. Нам відображається назва кластера, регіон і зона, в яких він заходиться, назва

пов’язаного сховища об’єктів та його статус. Описана інформація наведена на рис. 3.16.

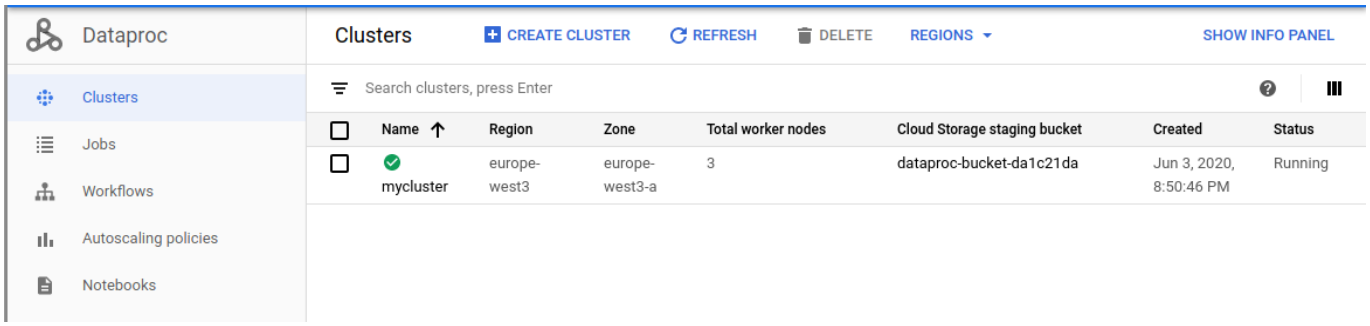


Рис. 3.16. Статус роботи новоствореного кластера

Для перегляду інформації про обчислювальні завдання, що виконувались кластером, потрібно перейти у підменю Dataproc -> Jobs. На рис. 3.17 ми бачимо два раніше описані обчислювальні завдання, їх статус, тип і час виконання.

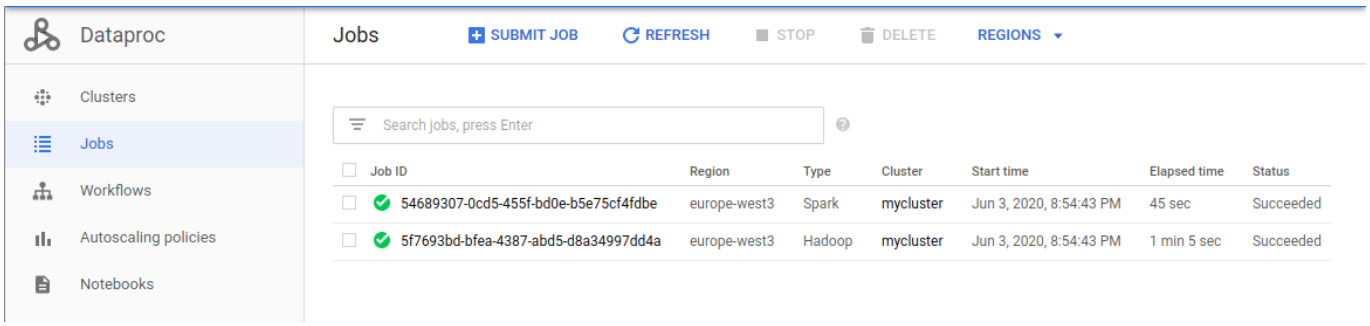


Рис. 3.17. Статус виконання обчислювальних завдань

Вибравши ID роботи, можна переглянути додаткову інформацію про неї та результат виконання. На рис. 3.18 показаний результат виконання завдання, що використовує фреймворк Hadoop MapReduce для обчислення кількість повторення слів у текстовому файлі.

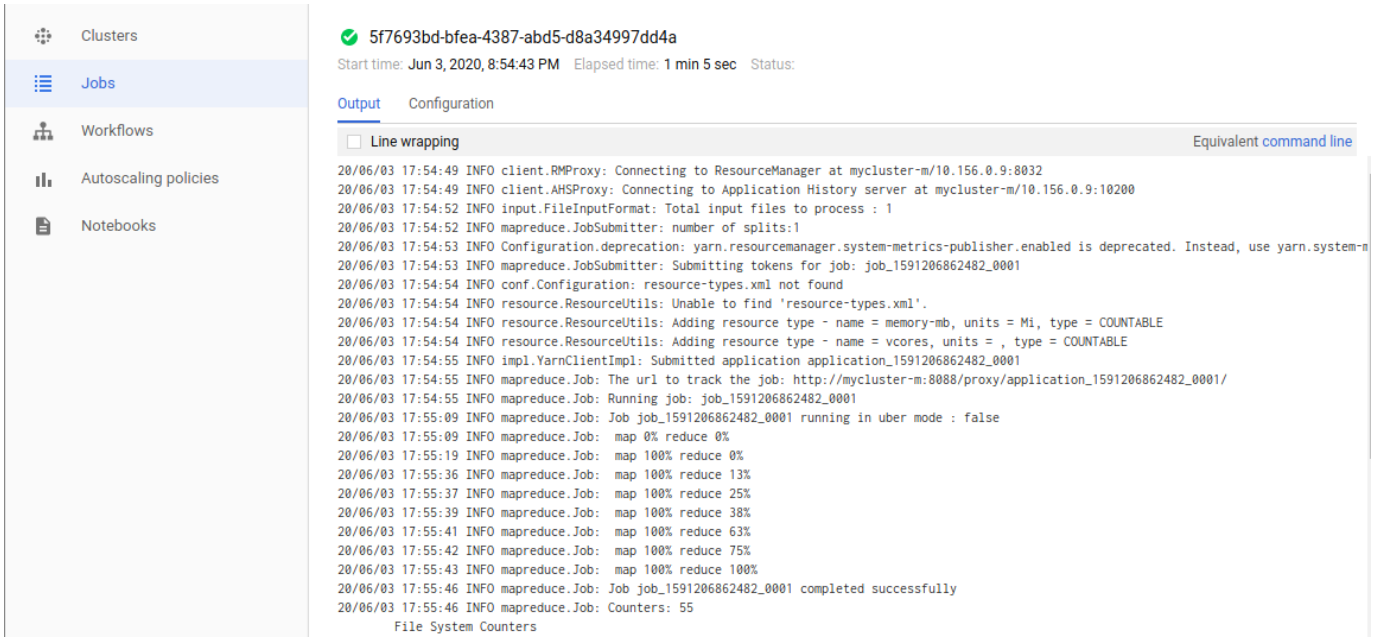


Рис. 3.18. Результат виконання програми MapReduce WordCount

Виконання роботи передбачало збереження результатів в сховище об’єктів Google Bucket. У пов’язаному файлі знаходиться інформація про кількість повторень слів з тексту – див. рис. 3.19.

```
Commerce,       1
License 1
Software        4
The     6
across  1
code.   1
eligible        1
following       1
for     3
object  1
of      6
on      2
regulations     1
restrictions    1
software        4
the     6
```

Рис. 3.19. Вміст файлу з результатами виконання програми WordCount

Вибравши ID Spark завдання бачимо інформацію про те, який фрагмент паралельного коду на якій машині виконувався та результат обчислення числа Pi, з

використанням паралельних обчислень на ресурсах кластера, що показано на рис 3.20.

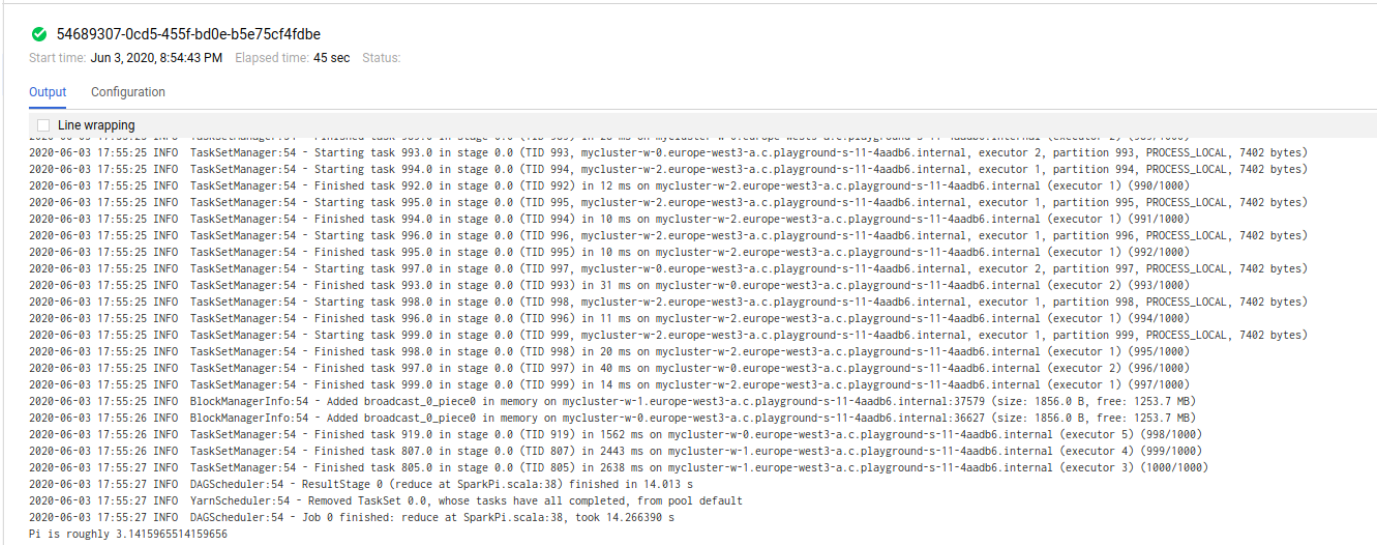


Рис. 3.20 Результат виконання паралельної програми SparkPi

3.3. Висновки до розділу 3

У ході виконання дослідження розроблено код опису інфраструктури платформи опрацювання великих даних Google Dataproc з використанням інструменту IaC Terraform. Описано процес розгортання та внесення змін у стан інфраструктури. Для перевірки роботоздатності платформи запущено та представлено результат виконання двох обчислювальних завдань, що використовують технології паралельних розподілених обчислень.

РОЗДІЛ 4
ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1. Охорона праці

Охорона праці – це система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів та засобів, спрямованих на збереження життя, здоров'я і працездатності людини у процесі трудової діяльності [11]. Важливим елементом охорони праці є робоче місце, в межах якого здійснюється діяльність працівників. З розвитком виробничих процесів та інформаційні технології все більше робочих місць працівників оснащуються персональним комп’ютером. Однак їх використання загострило проблеми збереження власного та суспільного здоров'я, вимагає удосконалення існуючих та розробки нових підходів до організації робочих місць, проведення профілактичних заходів для запобігання розвитку негативних наслідків впливу на здоров'я робітників [12].

Дослідження технологій розгортання інфраструктур опрацювання великих даних у хмарних сервісах здійснювалося з дотриманням усіх норм та правил охорони праці і техніки безпеки. Оскільки виконання роботи вимагає використання обчислювальної техніки, необхідно забезпечити дотримання правил охорони праці, техніки безпеки та протипожежної безпеки при її використанні. При роботі з комп'ютером працівник піддається дії ряду небезпечних і шкідливих виробничих факторів: електромагнітних полів (діапазон радіочастот: ВЧ, НВЧ), інфрачервоного і іонізуючого випромінювання дії статичної електрики.

До діючих нормативних документів, що забезпечують охорону праці користувачів ЕОМ належать:

– НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» [13];

– ДСанПіН 3.3.2.007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» [14];

– Закон України «Про охорону праці» [11].

Заходи з охорони праці користувачів ЕОМ розглядають в трьох аспектах: соціальному, психологічному та медичному. У соціальному плані розв'язання цих проблем пов'язане з оптимізацією умов життя і праці. Значне місце у профілактиці розладів здоров'я належить психології праці. Заходи, пов'язані з забезпеченням у виробничих колективах здорової психологічної атмосфери, сприяють зменшенню нервово-психічного перенапруження, підвищенню працездатності та ефективності праці.

Найповнішим нормативним документом щодо забезпечення охорони праці користувачів персонального комп'ютера є «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами (ВДТ) електронно-обчислювальних машин ДСанПіН 3.3.2.007-98. У процесі роботи з комп'ютером необхідно дотримувати правильний режим праці та відпочинку. Недотримання режиму праці може приводити до значної напруги зорового апарату, головної болі, порушення сну, втоми і хворобливі відчуття в очах, в попереку, в області шиї і руках і незадоволеності роботою.

Службові приміщення, в яких розташовані ПЕОМ, не повинні межувати з приміщеннями, де рівні шуму та вібрації перевищують норму (механічні цехи, майстерні тощо) [15].

Відповідно до ДСанПіН 3.3.2.007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» площа приміщення, у яких розташовують відеотермінали, визначається, виходячи з розрахунку на одне робоче місце - не менше 6,0 кв. м, об'єм - не менше 20,0 куб. м, з урахуванням максимальної кількості осіб, які одночасно працюють у зміні [12].

Заземлені конструкції, що знаходяться в приміщеннях (батареї опалення, водопровідні труби, кабелі із заземленим відкритим екраном тощо), мають бути надійно захищені діелектричними щитками або сітками від випадкового дотику. У приміщеннях з ПЕОМ слід щоденно проводити вологе прибирання, повинні бути медичні аптечки першої допомоги, а також система автоматичної пожежної

сигналізації. [14]. Заходи дотримання протипожежної безпеки передбачають, що в приміщення, де експлуатуються ЕОМ повинні бути забезпечені від виникнення короткого замикання, а також від перепадів напруги, що може викликати збої в роботі електронно–обчислювальної техніки. Приміщення повинні бути оснащені системою автоматичної пожежної сигналізації та вогнегасниками. Під час експлуатації ліній електромережі необхідно повністю виключити можливість виникнення електричного джерела загоряння внаслідок короткого замикання та перевантаження проводів, не допускати застосування проводів з легкозаймистою ізоляцією і, за можливості, застосовувати негорючу ізоляцію. Будівлі і ті їх частини, в яких розташовуються ЕОМ, повинні мати не нижче II ступеня вогнестійкості.

Штучне освітлення в приміщеннях з робочим місцем, обладнаним візуальними дисплейними терміналами має здійснюватися системою загального рівномірного освітлення. Як джерело штучного освітлення мають застосовуватись люмінесцентні або світлодіодні лампи. Загальне освітлення має бути виконане у вигляді суцільних або переривчастих ліній світильників, що розміщуються збоку від робочих місць (переважно зліва) паралельно лінії зору працівників.

Рівень освітленості на робочому столі в зоні розташування документів має бути в межах 300-500 лк. У разі неможливості забезпечити даний рівень освітленості системою загального освітлення допускається застосування світильників місцевого освітлення, але при цьому не повинно бути відблисків на поверхні екрана та збільшення освітленості екрана більше ніж до 300 лк.

При дотриманні описаних вимог, виконання дослідження технологій розгортання інфраструктур опрацювання великих даних у хмарних сервісах буде безпечним з точки зору охорони праці, техніки безпеки та протипожежної безпеки.

4.2. Безпека в надзвичайних ситуаціях

4.2.1. Фактори, що впливають на функціональний стан користувачів комп’ютерів. Функціональним станом людини називають оцінку рівня здатності ефективно виконувати поставлені завдання, її працездатність. Розглядаючи

користувача комп'ютерної системи, незадовільний функціональний стан може привести до виникнення помилок у роботі таких систем, які у залежності від відповідальності посади працівника здатні привести до значними економічних втрат. Неприйнятний функціональний стан впливає на виникнення професійних, хронічних хвороб, втоми, емоційного напруження та стресу [16].

На якість функціонального стану людини, яка працює за комп'ютером, значною мірою впливає середовище роботи. До фізичних факторів середовища належить якість освітлення робочого простору, температура і вологість повітря, вплив електромагнітного випромінювання, шумовий фон. Такі фактори безпосередньо впливають на здоров'я і самопочуття, стимулюючи органи чуття людини. До хімічних факторів, які впливають на стан користувачів комп'ютерів, належать вплив пилу та шкідливих хімічних речовин, що можуть виділятися від друкарської і копіювальної техніки, поломці чи загорянні електроніки. Мають вплив на таку категорію людей і біологічні факторів робочого середовища, до яких відноситься підвищений рівень вірусів та інших патогенних організмів. Ці питання є особливо характерним для приміщень, де знаходиться велика кількість працівників, та при забезпеченні недостатньої вентиляції, у період епідемій.

Користувачі комп'ютерів піддаються дії психофізіологічних факторів, які впливають на їх функціональний стан. Робота за комп'ютером характеризується високим навантаженням на зоровий апарат, яке виникає через необхідність концентрувати фокус погляду на дисплеї монітора і здійснені рухів очних яблук в обмеженому діапазоні. Під час роботи з комп'ютером користувач піддається впливу статичного фізичного навантаження на опорно-руховий апарат, що пов'язане з напруженням при роботі з клавіатурою та мишею, довгим сидінням у одній позі. Професії, пов'язані з роботою за комп'ютером, у своїй більшості передбачають діяльність, яка характеризуються значним напруження вищих нервових центрів мозку людини, що відповідають за функції концентрації уваги, мислення, вирішення аналітичних проблем. На функціональний працівників також має вплив високе інформаційне навантаження, що виникає через необхідність опрацювання документації, проведенні досліджень та опануванні нових технологій.

До факторів, що впливають на стан працівника, належать зовнішні засоби діяльності, такі як ергономічні характеристики робочого місця та обладнання, зручність його розташування і можливість налаштування під індивідуальні показники користувача.

Функціональний стан користувачів комп'ютерів залежить і від внутрішніх засобів діяльності, до яких відносяться професійні характеристики та досвід роботи працівників. Ці показники також корелюють з частотою виникнення помилок та форс-мажорів при роботі з комп'ютерними системами. Досвідчений і кваліфікований працівник отримує значно менше стресу та емоційного навантаження при розв'язанні нетипових завдань чи у нестандартних ситуаціях.

Працівники, виконання завдань яких передбачає роботу з комп'ютерами, частіше за все, звичайно, є членами певного колективу. Таким чином, до факторів, які мають вплив на функціональний стан слід віднести соціально-психологічні фактори міжособових відносин. До таких факторів відносяться питання довіри, доброзичливості, психологічної культури колективу, стиль керівництва, якого дотримуються в організації, ступень задоволення роботою та її результатом, умови праці і побуту.

Отже, на функціональний стан користувачів комп'ютерної техніки має вплив ціла низка факторів. Для кожного конкретного працівника потрібно окремо вираховувати ступень впливу певного чинника, оскільки усі вони мають безпосереднє відношення до самопочуття, фізичного і психологічного здоров'я людини. Підтримка задовільного функціонального стану користувача дозволяє йому якісно та безпомилково виконувати поставлені завдання, знаходити оптимальне вирішення у нестандартних ситуаціях та не шкодити здоров'ю.

4.3. Висновки до розділу 4

В результаті дослідження технологій розгортання інфраструктур опрацювання великих даних у хмарних сервісах було визначено та описано шкідливі і небезпечні чинники, що мають вплив на розробника. Здійснено аналіз нормативних документів

з охорони праці, техніки безпеки та протипожежної безпеки, на основі яких описано параметри і характеристики робочого приміщення, та проведення заходів, необхідних для безпечного виконання роботи.

Під час виконання дослідження безпеки в надзвичайних ситуаціях розглянуто вплив фізичних, хімічних, психофізичних факторів робочого середовища на функціональний стан користувачів комп'ютерів. Описано виплив зовнішніх і внутрішніх засобів діяльності, а також соціально-психологічних факторів міжособових відносин на стан самопочуття, фізичного і психологічного здоров'я працівника.

ВИСНОВКИ

За час виконання кваліфікаційної роботи магістра проведено дослідження технологій розгортання інфраструктур опрацювання великих даних у хмарних сервісах. У підсумку проведення теоретичних і експериментальних досліджень отримані такі результати:

– здійснено огляд предметної області великих даних та виокремлено особливості інфраструктур для їх опрацювання;

– розглянуто основні моделі хмарних обчислень, що дозволяють побудувати інфраструктуру для роботи з великими даними;

– проаналізовано публічні хмарні сервіси для роботи з великими даними та засоби для побудови приватних хмарних сервісів;

– виокремлено та класифіковано проблеми, які виникають при розгортанні інфраструктур опрацювання великих даних та описано підходи для їх вирішення;

– здійснено огляд підходу «Інфраструктура як код» та порівняння інструментів, що використовуються для його реалізації;

– адаптовано практики розробки ПЗ для опису хмарних інфраструктур опрацювання великих даних;

– описано алгоритм вибору оптимальної моделі для реалізації інфраструктури з урахуванням їх характеристик та типових вимог замовника;

– розроблено код для опису інфраструктури опрацювання великих даних на платформі Dataproc засобами інструменту Terraform і використанням підходу «інфраструктура як код»;

– підтверджено працездатність описаної інфраструктури шляхом запуску обчислювальних завдань.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Big Data What it is and why it matters [Електронний ресурс] // SAS – Режим доступу до ресурсу: https://www.sas.com/en_us/insights/big-data/what-is-big-data.html.
2. Min Chen, Shiwen Mao, Yin Zhang, Victor C.M. Leung. Big Data. Related Technologies, Challenges, and Future Prospects. — Spinger, 2014. — 100 с.
3. Amazon EMR Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.aws.amazon.com/emr
4. Google Dataproc documentation [Електронний ресурс] – Режим доступу до ресурсу: https://cloud.google.com/dataproc/docs.
5. Create a cluster [Електронний ресурс] // Dataproc Documentation – Режим доступу до ресурсу: https://cloud.google.com/dataproc/docs/guides/create-cluster#dataproc-create-cluster-gcloud.
6. Schults C. What Is Infrastructure as Code? How It Works, Best Practices, Tutorials [Електронний ресурс] / Carlos Schults // Stackify. – 2019. – Режим доступу до ресурсу: https://stackify.com/what-is-infrastructure-as-code-how-it-works-best-practices-tutorials/.
7. Grady B. Object Oriented Design: With Applications / Booch Grady. – Boston, MA: Pearson Education, 2007. – 551 с.
8. Мельников Е. Методология разработки CI/CD [Електронний ресурс] / Евгений Мельников // itglobal.com. – 2019. – Режим доступу до ресурсу: https://itglobal.com/ru-ru/company/blog/development-method-ci-cd/.
9. Brikman Y. Terraform: Up & Running: Writing Infrastructure as Code / Yevgeniy Brikman. – Sebastopol, CA: O'Reilly Media, 2019. – 368 с.
10. Саати Т. Принятие решений. Метод анализа иерархий / Томас Саати. – Москва: Радио и связь, 1993. – 278 с.
11. Закон України Про охорону праці Відомості Верховної Ради України (ВВР), № 49, 1992. 668 с.
12. Про захист людини від впливу іонізуючих випромінювань: Закон України від 14 січня 1998 р. Київ, 1998. 24 с.

13. Наказ міністерства соціальної політики України № 207 Про затвердження вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями від 14.02.2018.
14. ДСанПІН 3.3.2.007-98. Гігієнічні вимоги до організації роботи з візуальними дисплейними терміналами електронно-обчислювальних машин №7 від 10.12.98.
15. НПАОП 0.00-7.15-18. Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями від 14.02.2018.
16. Конспект лекцій з курсу «Охорона праці в галузі» / Укладачі: Яскілка В.Я., Олійник М.З. – Тернопіль: ТНТУ імені Івана Пулюя, 2016. – 56 с.

ДОДАТКИ

Додаток А Тези конференцій

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний технічний університет імені Івана Пулюя (Україна)
Національна академія наук України
Університет імені П'єра і Марії Кюрі (Франція)
Маріборский університет (Словения)
Технічний університет у Кошице (Словаччина)
Вільнюський технічний університет ім. Гедимінаса (Литва)
Шяуляйська державна колегія (Литва)
Жешувський політехнічний університет ім. Лукасевича (Польща)
Білоруський національний технічний університет (Республіка Білорусь)
Міжнародний університет цивільної авіації (Марокко)
Національний університет біоресурсів і природокористування України (Україна)
Наукове товариство ім. Шевченка
ГО «Асоціація випускників Тернопільського національного технічного університету імені Івана Пулюя»

АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ

Збірник

тез доповідей

Том II

IX Міжнародної науково-технічної конференції молодих учених та студентів

25-26 листопада 2020 року

УКРАЇНА
ТЕРНОПІЛЬ – 2020

ЗМІСТ

Секція: КОМПЮТЕРНО-ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ТА СИСТЕМИ ЗВ'ЯЗКУ

Матеріали IX Міжнародної науково-технічної конференції молодих учених та студентів.
Актуальні задачі сучасних технологій – Тернопіль 25-26 листопада 2020.

УДК 004.272.3
А. М. Луцків, канд .техн. наук, доц., М. П.Голубовський
Тернопільський національний технічний університет ім. Івана Пулюя, Україна

КРИТЕРІЇ ВИБОРУ ІНСТРУМЕНТІВ IAC

A.M. Lutskiv, Ph.D., Assoc. Prof, M.P. Holubovskyi

IAC TOOLS SELECTION CRITERIA

Процес вибору необхідного інструменту для опису інфраструктури у вигляді коду є комплексною задачею. Усі популярні рішення дозволяють для описати інфраструктуру та з використанням будь-якого можна реалізувати поставлене завдання, але це не означає що вибір буде оптимальним і не приведе до непередбачуваних наслідків у майбутньому, не потребуватиме додаткових витрат часу на дослідження нюансів роботи та підводних каменів. До популярних інструментів належать Chef, Puppet, Ansible, SaltStack, CloudFormation, Deployment Manager, Terraform.

IaC рішення поділяються на рішення з відкритим вихідним кодом та пропрієтарні. З наведених вище продуктів CloudFormation і Deployment Manager є пропрієтарними інструментами Amazon та Google відповідно і можуть працювати тільки з їх хмарними середовищами. Решта – працюють з усіма популярними постачальниками хмарних послуг.

Першою вагомою відмінністю між описаними інструментами є те, що Chef, Puppet, Ansible і SaltStack є інструментами для керування конфігурацією (configuration management tools), і призначені для встановлення та налаштування програмного забезпечення на існуючому апаратному забезпеченні (віртуальних машинах, серверах). CloudFormation і Terraform – інструменти для створення (provisioning) інфраструктури. Вони розроблені для створення ресурсів машин та інших елементів інфраструктури: баз даних, сховищ інформації, налаштування мережі тощо, залишаючи роботу по налаштуванню для інших рішень. Ці категорії не є взаємовиключними: більшість інструментів керування конфігурацією можуть забезпечити певні етапи створення інфраструктури, так само як інструменти створення інфраструктури можуть виконувати налаштування. Проте, акценти закладені розробниками приводять то того, що конкретні інструменти краще підходять для виконання конкретних завдань.

Інструменти керування конфігурацією Chef, Puppet, Ansible та SaltStack працюють за парадигмою змінної інфраструктури. Наприклад, якщо вказати такому інструменту оновити версію програмного пакету, він виконає оновлення програмного забезпечення на існуючих, робочих серверах і зміни одразу будуть застосовані. У ході такого процесу не гарантується конкретний стан системи, інструмент відповідає тільки за виконання процедури, не враховуючи стан службових, тимчасових системних файлів та інших програм. Якщо з часом буде виконуватися більше і більше оновлень кожна машина буде містити унікальну історію змін. Це може призвести до такого феномену як зсув конфігурації (configuration drift), при якому кожен сервер стає трохи іншим за інші, що призводить до тонких помилок конфігурації, які важко діагностувати і майже неможливо відтворити. При використанні інструментів створення інфраструктури, для внесення змін у конфігурацію при кожному оновленні буде створюватися нова машина, а існуюча – видалятися. Такий підхід зменшує ризик проблем пов'язаних з конфігураційним зсувом, дозволяє точно знати яке програмне забезпечення запущене та дозволить за потреби легко розгорнути попередню версію у будь-який час.

Chef і Ansible використовують процедурний стиль, де код крок-за-кроком описується послідовність дій потрібну для досягнення бажаного стану конфігурації.

Для Terraform, CloudFormation, SaltStack і Puppet є характерним декларативний стиль, коли код описує бажаний кінцевий стан системи, а інструмент сам відповідальний для виконання дій, потрібних для його досягнення. Недоліком процедурного підходу є те, що стан такої системи може не повністю відповідати коду, щоб з'ясувати кінцевий стан системи потрібно знати порядок, з яким вносились та застосовувалися зміни. Повторне використання процедурного коду є обмеженим, тому що код написаний раніше може бути більше не придатним до використання, оскільки він був призначений для модифікації стану інфраструктури яка більше не існує. З декларативним підходом код завжди відповідає останньому стану інфраструктури. Таким чином, описаний підхід забезпечує самодокументованість розгорнутої інфраструктури й дає змогу здійснювати аналіз історії її змін у системі контролю версій.

Недоліком декларативного підходу є те, що деякі типи інфраструктурних змін важко описати у чисто декларативних термінах, уникнувши логічних операторів, циклів. Тому, зазвичай такі інструменти надають примітиви, зокрема: змінні, модулі, цикли, які дають змогу створювати функціональний, багаторазовий код навіть з використанням декларативної мови.

За замовчуванням для роботи Chef, Puppet, SaltStack потребують запуску мастер сервера для зберігання інформації про стан інфраструктури та розповсюдження змін. Для внесення зміни використовуються клієнт (CLI) який передає команди на мастер сервер, а він уже виконує потрібні дії. Перевагою є те, що мастер є єдиним, центральним місцем звідки можна переглянути та керувати станом інфраструктури. Він постійно працює у фоновому режимі і відстежує стан конфігурації. Таким чином, якщо відбувається внесення змін вручну, головний сервер може скасувати цю зміну, щоб запобігти дрейфу конфігурації на цільовій системі. Ansible, CloudFormation, Heat, Terraform за-замовчуванням не потребують використання мастер сервера. Такий функціонал уже вбудований у інструмент і не потребує додаткового налаштування. Ansible, наприклад, працює, під'єднуючись до кожної машини через SSH, тому не потребує додаткової інфраструктури для запуску чи налаштування інших механізмів автентифікації.

Chef, Puppet, SaltStack вимагають встановлення програмного забезпечення агента на кожну машину яка буде налаштовуватися. Для цього можуть використовуватися образи віртуальних машин, де вже буде потрібне ПЗ, скрипти які виконуються при створенні ресурсу, доступ до віддалених машин по SSH. Недоліком є необхідність моніторити стан, оновлювати та підтримувати роботу програми-агента на цільових машинах. Використання агентів впливає на налаштування безпеки, викликаючи необхідність відкривати необхідні мережеві порти для доступу чи інші механізми, що можуть бути вразливими до дій зловмисників. Ansible, CloudFormation, Heat і Terraform не вимагають встановлення агентів для своєї роботи. Точніше, агенти, зазвичай, вже встановлені. Для прикладу, постачальники хмарних послуг встановлюють свої агенти на кожен фізичний сервер. Користувач Terraform виконує команди, а агент постачальника виконує необхідні дії.

При виборі інструменту IaC також відіграє роль спільнота та зрілість технології. Ці фактори визначають, скільки людей бере участь у розвитку проекту, скільки доступно інтеграцій та розширень, наскільки легко знайти документацію, літературу та вирішення проблем. Для вибору варто врахувати кількість та активність розробки продукту, активність контриб'юторів щодо повідомлення про помилки та їх виправлення, повноту документації.

Література

1. Brikman Y. Terraform: Up & Running: Writing Infrastructure as Code / Yevgeniy Brikman. – Sebastopol, CA: O'Reilly Media, 2019. – 368 c.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

VIII НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

9–10 грудня 2020 року

ТЕРНОПІЛЬ
2020

УДК 004.272.3

канд. техн. наук, доц Луцків А. М, Голубовський М. П.
(Тернопільський національний технічний університет ім. Івана Пулюя, Україна)

ПРОБЛЕМИ, ЯКІ ВИНИКАЮТЬ ПРИ РОЗГОРТАННІ ІНФРАСТРУКТУР ДЛЯ ОПРАЦЮВАННЯ ВЕЛИКИХ ДАНИХ

UDC 004.272.3

Ph.D., Assoc. Prof Lutskiv A, Holubovskyi M.

PROBLEMS THAT ARISE DURING DEPLOYMENT OF BIG DATA PROCESSING INFRASTRUCTURES

Використання традиційних, ручних способів розгортання, налаштування та підтримки інфраструктури для опрацювання великих даних вимагає великих затрат часу та значної кількості фахівців. Спочатку здійснюється конфігурація апаратного забезпечення – обчислювальних ресурсів та мережі для їх взаємодії. Наступними кроками є встановлення та налаштування операційних систем, програм та бібліотек від яких залежить робота додатків. Тільки після виконання усіх етапів підготовки спеціалісти виконують встановлення компонентів комплексу для опрацювання великих даних. Оскільки, до виконання таких завдань використовується ручний підхід, це спричиняє виникнення певних проблем та наслідків.

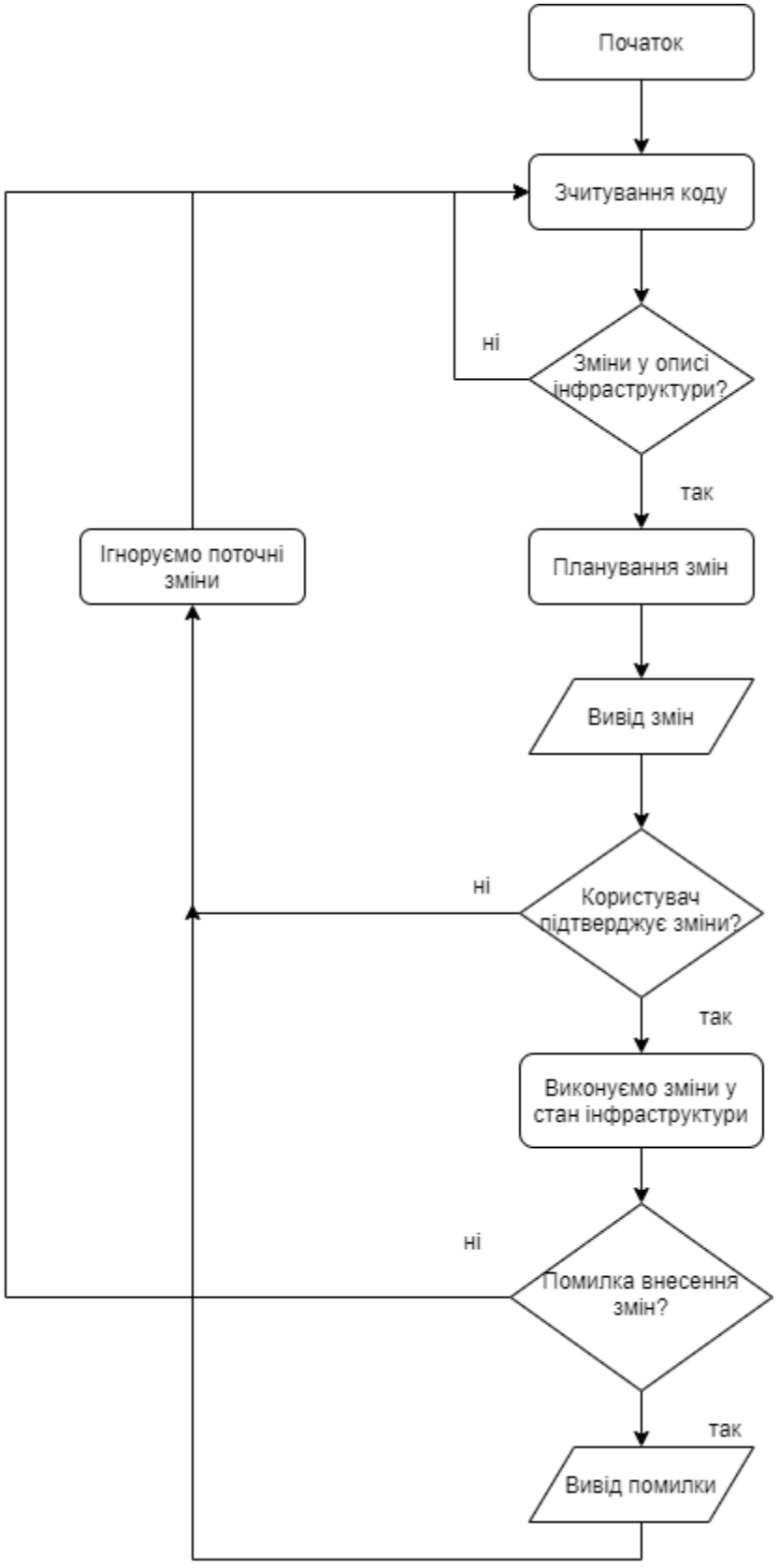
Першою проблемою є швидкість створення та підготовки до роботи такої інфраструктури. Також проблемою є непостійність при конфігуруванні ресурсів, виникнення розбіжностей при ручному конфігурування, зокрема, при роботі команди спеціалістів. Традиційний підхід не надає можливості здійснювати тестування перед внесенням змін у конфігурацію, перегляду історії змін та зручної командної роботи.

Для вирішення описаних проблем ефективним рішенням є використання підходу опису інфраструктури у вигляді коду. Такий спосіб створення та керування обчислювальними ресурсами передбачає їх опис у вигляді програмного коду, на відміну від ручного редагування конфігураційних файлів чи використання інтерактивних інструментів. Така інфраструктура може охоплювати фізичні сервери, віртуальні машини, а також пов'язані з ними ресурси і програмні компоненти. Таким чином, при використання підходу "інфраструктура як код" можемо використовувати практики притаманні типовій розробці ПЗ. До переваг, які надає використання даного підходу відноситься швидкість створення, послідовність конфігурування, підзвітність описаної системи. Швидкість роботи досягається завдяки можливості створити всю потрібну інфраструктуру, використовуючи лише одну команду над конфігураційним файлом. Такий файл може використовуватися для створення ідентичних ізольованих систем та надає змогу легко проводити тестування перед внесенням змін. Вже описані елементи такої інфраструктури можуть бути повторно використані при роботі над іншими проектами. Підхід дозволяє уникнути непослідовності, помилок пов'язаних з людським фактором, оскільки, єдиним джерелом для внесення змін є конфігураційні файли, виключається необхідність внесення ручних змін. Перевагою також є наявність звітності про те, які зміни, коли і ким були внесені. Таку конфігурацію можна зберігати у системі контролю версій. Використання підходу документує стан інфраструктури, дає змогу мати задокументований стан того, як вона налаштована у поточний момент часу.

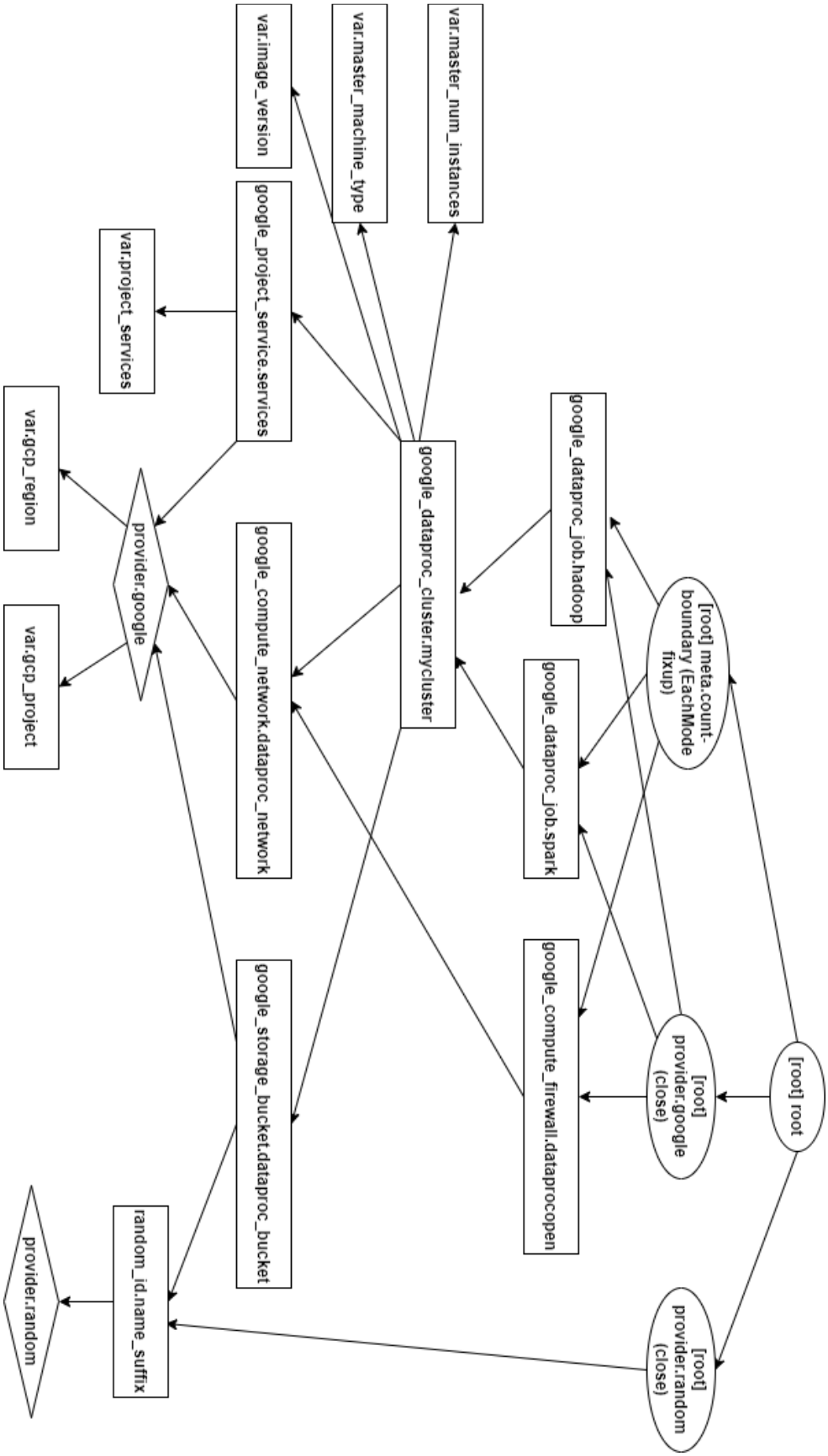
Література.

1. Morris K. Infrastructure as Code: Managing Servers in the Cloud / Kief Morris. – Sebastopol, CA: O'Reilly Media, 2016. – 362 с.

Додаток Б Блок-схема алгоритму роботи системи



Додаток В Граф залежності ресурсів Terraform



Додаток Д Лістинг коду опису інфраструктури

```
provider "google" {
  project = var.gcp_project
  region  = var.gcp_region
}

resource "random_id" "name_suffix" {
  byte_length = 4
}

resource "google_storage_bucket" "dataproc_bucket" {
  name          = "dataproc-bucket-${random_id.name_suffix.hex}"
  location      = "EU"
  force_destroy = "true"
}

resource "google_dataproc_cluster" "mycluster" {
  name   = "mycluster"
  region = var.gcp_region
  labels = {
    env = "testing"
  }

  cluster_config {
    staging_bucket = google_storage_bucket.dataproc_bucket.name

    master_config {
      num_instances = var.master_num_instances
      machine_type  = var.master_machine_type
      disk_config {
        boot_disk_type    = var.boot_disk_type
        boot_disk_size_gb = var.boot_disk_size
      }
    }

    worker_config {
      num_instances = var.worker_num_instances
      machine_type  = var.worker_machine_type
      disk_config {
        boot_disk_size_gb = var.boot_disk_size
        num_local_ssds    = 1
      }
    }

    preemptible_worker_config {
      num_instances = var.preemptible_worker_num_instances
    }

    # Override or set some custom properties
    software_config {
```

```
      image_version = var.image_version
      override_properties = {
        "dataproc:dataproc.allow.zero.workers" = "true"
      }
    }

    gce_cluster_config {
      tags    = ["testing", "dataproc"]
      network = google_compute_network.dataproc_network.name
      service_account_scopes = [
        "https://www.googleapis.com/auth/monitoring",
        "useraccounts-ro",
        "storage-rw",
        "logging-write",
      ]
    }

    # You can define multiple initialization_action blocks
    initialization_action {
      script      = "gs://dataproc-initialization-
actions/stackdriver/stackdriver.sh"
      timeout_sec = 500
    }
    initialization_action {
      script      = "gs://dataproc-initialization-
actions/ganglia/ganglia.sh"
      timeout_sec = 500
    }

    initialization_action {
      script      = "gs://dataproc-initialization-
actions/zookeeper/zookeeper.sh"
      timeout_sec = 5000
    }

    initialization_action {
      script      = "gs://dataproc-initialization-
actions/docker/docker.sh"
      timeout_sec = 500
    }

    initialization_action {
      script      = "gs://dataproc-initialization-
actions/livy/livy.sh"
      timeout_sec = 500
    }
  }

  depends_on = [google_storage_bucket.dataproc_bucket,
google_project_service.services]

  timeouts {
```

```
    create = "30m"
    delete = "30m"
  }
}

resource "google_dataproc_job" "hadoop" {
  region       = google_dataproc_cluster.mycluster.region
  force_delete = true

  placement {
    cluster_name = google_dataproc_cluster.mycluster.name
  }

  hadoop_config {
    main_jar_file_uri = "file:///usr/lib/hadoop-mapreduce/hadoop-
mapreduce-examples.jar"

    args = [
      "wordcount",
      "file:///usr/lib/spark/NOTICE",

"gs://${google_dataproc_cluster.mycluster.cluster_config.0.bucket}/had
oopjob_output",
    ]
  }
}

resource "google_dataproc_job" "spark" {
  region       = google_dataproc_cluster.mycluster.region
  force_delete = true

  placement {
    cluster_name = google_dataproc_cluster.mycluster.name
  }

  spark_config {
    main_class    = "org.apache.spark.examples.SparkPi"
    jar_file_uris = ["file:///usr/lib/spark/examples/jars/spark-
examples.jar"]
    args          = ["1000"]

    properties = {
      "spark.logConf" = "true"
    }

    logging_config {
      driver_log_levels = {
        "root" = "INFO"
      }
    }
  }
}
```

```
resource "google_project_service" "service_usage" {
  project            = var.gcp_project
  service            = "serviceusage.googleapis.com"
  disable_on_destroy = false
}

resource "google_project_service" "services" {
  for_each           = var.project_services
  project            = var.gcp_project
  service            = each.value
  disable_on_destroy = false
  depends_on         = [google_project_service.service_usage]
}

resource "google_compute_network" "dataproc_network" {
  name                    = "${var.gcp_project}-dataproc-network"
  auto_create_subnetworks = true
}

resource "google_compute_firewall" "dataprocopen" {
  name    = "${var.gcp_project}-dataprocopen"
  network = google_compute_network.dataproc_network.name

  // Allow ping
  allow {
    protocol = "icmp"
  }
  // Allow incoming connections
  allow {
    protocol = "tcp"
    ports    = ["1-65535"]
  }
  allow {
    protocol = "udp"
    ports    = ["1-65535"]
  }
}

variable "gcp_project" {
  default = "playground-s-11-13e95c6e"
}
variable "gcp_region" {
  default = "europe-west3"
}

variable "project_services" {
  description = "Required Apis & Services"
  type        = set(string)
  default = [
    "cloudresourcemanager.googleapis.com",
    "cloudbilling.googleapis.com",
    "iam.googleapis.com",
```

```
    "compute.googleapis.com",
    "oslogin.googleapis.com",
    "storage-api.googleapis.com",
    "dataproc.googleapis.com",
    "servicenetworking.googleapis.com"
  ]
}

variable "master_num_instances" {
  default = 1
}
variable "master_machine_type" {
  default = "n1-standard-2"
}
variable "boot_disk_size" {
  default = 30
}
variable "boot_disk_type" {
  default = "pd-ssd"
}

variable "worker_num_instances" {
  default = 3
}
variable "preemptible_worker_num_instances" {
  default = 0
}
variable "worker_machine_type" {
  default = "n1-standard-2"
}
variable "image_version" {
  default = "1.5-debian10"
}
```