Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління
(повна назва)
Кафедра Автоматизації проектування обчислювальної техніки
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

другий (магістерський)
(рівень вищої освіти)

ГЮІК.ХХХХХХ.001
(позначення документа)

Моделі архітектури високонавантажених web-застосунків на платформі хмарних сервісів Amazon
(тема)

Виконав: студент 2 курсу, групи СКСм-18-1
Богатиренко А.М
(прізвище, ініціали)
Спеціальність
123 Комп'ютерна інженерія
(код і повна назва спеціальності)
Тип програми Освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма Спеціалізовані комп'ютерні системи
(повна назва спеціалізації)
Керівник професор кафедри АПОТ
д.т.н Литвинова Є.І.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Чумаченко С.В.
(підпис) (прізвище, ініціали)

2019 р.

Харківський національний університет радіоелектроніки

Факультетв Комп'ютерної інженерії та управління

Кафедра Автоматизації проектування обчислювальної техніки

Рівень вищої освіти другий (магістерський)

Спеціальність 123 Комп'ютерна інженерія

Освітня програма Спеціалізовані комп'ютерні системи

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 20 ___ р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові Богатиренко Анатолію Михайловичу

(прізвище, ім'я, по батькові)

1. Тема роботи Моделі архітектури високонавантажених web-застосунків на платформі хмарних сервісів Amazon

затверджена наказом по університету від 04 листопада 2019 р. № 1624 Ст

2. Термін подання студентом роботи до екзаменаційної комісії ____ _____ 20__ р.

3. Вихідні дані до роботи
Terraform CLI
Amazon Web Services
Infrastructure as Code
Керування інфраструктурою

4. Перелік питань, що потрібно опрацювати в роботі
Моделі розгортання хмарних технологій
Аналіз хмарної інфраструктури
Аналіз високонавантажених веб-застосунків
Розробка хмарної інфраструктри за допомогою Terraform

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслеників, плакатів)___20_слайдів_____
_____
_____
_____

6. Консультанти розділів роботи (проекту)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи (проекту)	Термін виконання етапів проекту (роботи)	Примітка
1	Видача теми проекту, узгодження і затвердження	02.09.2019 – 08.09.2019	
2	Аналіз предметної області	09.09.2019 – 15.09.2019	
3	Аналіз джерел з галузі, постановка задачі, вибір інструментальних засобів	16.09.2019 – 29.11.2019	
4	Розробка моделі хмарної архітектури для веб-додатку	14.10.2019 – 20.10.2019	
5	Розробка алгоритму керування хмарною інфраструктурою	04.11.2019 – 01.12.2019	
6	Оформлення пояснювальної записки	02.12.2019 – 08.12.2019	
7	Перевірка виконаного проекту керівником,	08.12.2019 – 14.12.2019	
8	Захист проекту	20.12.2019 – 20.12.2019	

Дата видачі завдання _____

Студент _____
(підпис)

Керівник роботи (проекту) _____ проф. каф. Литвинова Є.І.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка містить 64 сторінок, 16 рисунків, 2 таблиці, 6 джерел за переліком посилань.

ХМАРНІ ТЕХНОЛОГІЇ, КЕРУВАННЯ ІНФРАСТРУКТУРОЮ AMAZON WEB SERVICES, TERRAFORM

Розглянуто питання одного з перспективних напрямків розвитку DevOps методолгій – Infrastucture as code із застосуваннямїх на хмарній платформі Amazon Web Services. Проаналізовані тендеції розвитку сучасних хмарних платформ. Реалізація алгоритму управління хмарою виконується за допомогою Terraform на мові опису YAML, що спрощує керування інфрастрктурою та зменшує час на розгортання та міграцію між хмарними провайдерами. Для оптимізації процесу доставки коду та його розробки використовується система контролю версій.

ABSTRACT

Bachelor's thesis contains 64 pages, 16 figures, 2 tables, 6 sources according to the reference list.

CLOUD TECHNOLOGIES, AMAZON WEB SERVICES, TERRAFORM INFRASTRUCTURE MANAGEMENT

Infrastucture as Code and an application based on the Amazon Web Services cloud platform are perspective directions of the development of DevOps methodologies. Trends in the development of modern cloud platforms are analyzed. The cloud management algorithm is implemented using Terraform with the YAML description language, which simplifies infrastructure management and reduces the time for deployment and migration between cloud providers. A version control system is used to optimize the code delivery and development process.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ ……7
ВСТУП ……8
1 ХМАРНІ ТЕХНОЛОГІЇ: ОСНОВНІ ПОНЯТТЯ, ПРОВАЙДЕРИ ТА ТЕНДЕНЦІЇ РОЗВИТКУ ……10
1.1 Визначення……10
1.2 Характеристики хмарних технологій……12
1.3 Моделі розгортання хмарних технологій……14
1.4 Моделі обслуговування хмарних технологій (IaaS, PaaS, SaaS)……18
1.5 Хмарні тенденції……20
1.6 Хмарний провайдер Amazon Web Services……22
1.6.1 Amazon Elastic Compute Cloud (Amazon EC2)……22
1.6.2 Amazon Simple Storage Service (Amazon S3)……24
1.6.3 Amazon Elastic Block Store (Amazon EBS)……25
1.6.4 Amazon Elastic File System (Amazon EFS)……26
1.6.5 Amazon Machine Image (Amazon AMI)……26
1.6.6 Amazon Relational Database Service (Amazon RDS)……27
1.6.7 Amazon Elastic Load Balancer (Amazon ELB)……27
1.6.8 Amazon Virtual Private Cloud (Amazon VPC)……28
1.6.9 Amazon Route 53……29
1.7 Git……30
1.7.1 Репозіторій для зберігання інфраструктури……31
1.8 Terraform як інструмент керування інфраструктурою……32
1.8.1 Переваги використання Terraform……33
1.8.2 Управління Terraform……34
1.8.3 Робота з Terraform……36
2 ПЛАНУВАННЯ ТА РОЗРОБКА АРХІТЕКТУРИ……38
2.1 Визначення із хмарним провайдером……38

2.2 Керування інфраструктурою ……..38
2.3 Інфраструктура для веб-додатку……..…40
2.3.1 Масштабування для високого навантаження……..…..41
2.3.2 База даних RDS…….……..……...…..41
2.3.3 Балансування навантаження БД……..……43
2.3.4 Балансування навантаження EC2 за допомогою Amazon ELB та Auto Scaling Group……44
2.3.5 Route 53……..…47
3 РОЗРОБКА МОДЕЛІ КЕРУВАННЯ ДЛЯ AMAZON WEB SERVICES……49
3.1 Веб-складова……..……....49
3.2 Описання архітектури AWS..……..52
ВИСНОВКИ ……..……...…….69
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ ……71
ДОДАТОК А Графічна частина проекту ……...72

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

IaaS – Infrastructure as a service

PasS – Platform as a service

SaaS – Software as a service

IaC – Infrastructure as code

AWS – Amazon web services

EC2 – Elastic compute cloud

AMI – Amazon machine image

RDS – Relational database service

ELB – Elastic load balancer

CIDR – Classless Inter-Domain Routing

SSH – Secure Shell

YAML – Yet another markup language

ЦОД – Центр обробки даних

ВСТУП

В даний час хмарні сервіси і провайдери поступово нарощують популярність і вплив в самих різних життєвих сферах. Для збереження і підвищення ефективності бізнесу багато підприємців все частіше звертаються до хмарних рішенньь.

Хмарні обчислення – технологія розподіленої обробки даних, в якій комп'ютерні ресурси і потужності надаються користувачеві як Інтернет-сервіс. Суть концепції хмарних обчислень полягає в наданні кінцевим користувачам віддаленого динамічного доступу до послуг, обчислювальних ресурсів і додатків (у тому числі операційних систем і інфраструктури) через інтернет. Розвиток сфери хостингу було обумовлено виниклою потребою в програмному забезпеченні і цифрових послугах, якими можна було б управляти зсередини, але які були б при цьому більш економічними і ефективними за рахунок економії на масштабі.

Впровадження хмари IaaS на підприємствах часто починається з сучасних робочих навантажень, підкреслюючи продуктивність розробників і гнучкість бізнесу. Але все більше і більше хмарних IaaS купується для традиційних IT з упором на зниження витрат, безпеку і захищеність. Лідери інфраструктури та операцій зазвичай керують пошуком, коли хмарні IaaS повинні використовуватися для традиційних IT. На відміну від цього, пошук джерел для сучасних робочих навантажень зазвичай здійснюється корпоративними архітекторами, лідерами розробки додатків і лідерами цифрового бізнесу.

Ринок хмарної інфраструктури як послуги IaaS продовжує консолідуватися, оскільки клієнти і постачальники прагнуть до більшої безпеки, інновацій та масштабування. Уже дев'ятий рік поспіль AWS оцінюється як лідер в хмарі IaaS.

Нарівні з питанням про вибір хмарного провайдера також актуальним є питання ефективного і безпечного використання хмарних ресурсів, автоматизації, збереження та резервного копіювання інфраструктури. Рішенням є застосування нових практик та інструментів, наприклад застосування практик IaC.

Infrastructure as Code – один з логічних напрямків DevOps, що дозволяє описувати, моделювати і змінювати інфраструктуру в форматі коду із застосуванням багаторічних практик розробки додатків. На практиці допомогає спростити масштабування і контроль інфраструктури в цілому, а також скоротити цикл розгортання і доставки додатків користувачеві. Одним з основних інструментів застосування IaC є Terraform, за допомогою якого буде розгорнута і описана вся інфрастркутура за допомогою YAML.

Метою атестаційної роботи є розробка хмарної архітектури на базі Amazon Web Services для високонавантажених веб-застосунків за допомогою Terraform.

Задачі дослідження:

– розробка базового конфігураційного файлу для забезпечення взаємодії бази даних PHP та Wordpress;

– опис архітектури AWS в Terraform;

– розробка модулів забезпечення безпеки інфраструктури;

– розробка алгоритму керування інфраструктурою.

1 ХМАРНІ ТЕХНОЛОГІЇ: ОСНОВНІ ПОНЯТТЯ, ПРОВАЙДЕРИ ТА ТЕНДЕНЦІЇ РОЗВИТКУ

1.1 Визначення

На сьогоднішній день хмарні обчислення є одним з найпопулярніших напрямків розвитку інформаційних технологій. Сучасні умови інформаційного світу вимагають вирішення багатьох завдань, які можуть бути ефективно виконані за допомогою хмарних технологій. Багато великих світових ІТ-компаній застосовують хмарні обчислення, що є підтвердженням ефективності даних технологій.

Хмарні технології представляють універсальну середу для зберігання і обробки інформації, яка об'єднує в собі апаратні засоби, ліцензійне програмне забезпечення, канали зв'язку, а також технічну підтримку користувачів. Під хмарними технологіями також мають на увазі можливість отримання необхідних обчислювальних потужностей за запитом з мережі. На сьогоднішній день великі ЦОД дозволяють не тільки зберігати і обробляти дані в своїх центрах, а й дають можливість створювати власні віртуальні дата-центри. Це дозволяє компаніям не витрачати ресурси на створення своєї інфраструктури з нуля.

Головною особливістю хмарних технологій є масштабованість: у даних технологій немає жорсткої прив’язки до апаратної платформи, немає прив’язки до географічної території. Використання хмарних технологій в компаніях направлено на зниження витрат і підвищення ефективності бізнес-процесів. Хмарні технології – це технології обробки даних, в яких комп'ютерні ресурси надаються Інтернет-користувачеві як онлайн-сервіс. Завдяки цьому користувач працює з хмарними сервісами з будь-якого місця і з будь-якого пристрою: головне мати доступ до мережі Інтернет. Доступ до «хмари» можна мати не тільки через Інтернет, а й через локальну мережу. В

такому випадку, комп'ютер користувача – це термінал, підключений до мережі. Ті комп'ютери, які здійснюють хмарні обчислення, є «обчислювальним хмарою». Навантаження між такими комп'ютерами розподіляється автоматично за допомогою спеціального фізичного та програмного обладнання.

«Хмара» має три основні складові:

1. Хмарні обчислення. Під хмарними обчисленнями мається на увазі архітектура комп'ютерної обробки даних. Хмарна архітектура надає можливості для самообслуговування, масштабування і гнучких процесів. Подібне архітектурне рішення замінює постійні витрати на змінні і надає широкі можливості для аналізу даних.

2. Хмарні платформи. Хмарні платформи включають в себе інструменти, програмні та інформаційні моделі, системне програмне забезпечення та інші технології, які виконують поставлені завдання.

3. Хмарні послуги. Хмарні послуги - це моделі надання інформаційних послуг.

Розвиток хмарних технологій має величезний вплив на бізнес. Для того щоб мати переваги перед конкурентами, компаніям необхідно враховувати сучасні тенденції в інформаційних технологіях. Компанії, що використовують хмарні технології в своїх бізнес-процесах, отримують ряд переваг. Хмарні обчислення - це підхід в управлінні бізнес-процесами, який дозволяє знизити складність інформаційних систем. Це досягається за рахунок застосування хмарних обчислень, керованих самостійно і доступних на вимогу в рамках віртуальної інфраструктури.

Таким чином, компанії мають такі плюси від використань хмарних обчислень: зниження витрат на IT, підвищення якості надання сервісу, динамічність бізнесу. Зниження витрат на IT обумовлено тим, що хмарні технології знижують оперативні та капітальні витрати: завдяки «хмарі» IT-фахівці компанії можуть зосередитися на стратегічних проектах, не витрачаючи час на управління власним ЦОД.

Робота хмарних технологій відбувається наступним чином: компанії, замість того щоб купувати, встановлювати і управляти власним серверами для запуску додатків, орендують сервера (наприклад, у Amazon або Microsoft). Користувач управляє цими серверами через мережу інтернет. Оплата включає тільки фактичне використання серверів для обробки і зберігання даних.

Обчислювальні хмари складаються з величезного числа серверів, які розміщені в дата-центрах. ЦОД забезпечують роботу десятків тисяч додатків, які одночасно використовуються мільйонами користувачів одночасно. Повна автоматизація є умовою ефективного управління інфраструктури таких масштабів.

Таким чином, використання хмарних технологій входить в тенденцію, і компаніям необхідно знати про них і ефективно застосовувати для поліпшення бізнес-процесів.

1.2 Характеристики хмарних технологій

Для того щоб інформаційні ресурси ставилися до хмарних технологій вони повинні володіти такими ключовими властивостями: мати високу доступність і масштабованість, бути економічно вигідними для клієнта. Для того щоб розрізняти хмарних технології від інших більш ранніх підходів до надання апаратних і програмних ресурсів, виділяють наступні основні характеристики хмарних обчислень:

- широка мережева доступність;
- легка масштабованість, еластичність;
- можливість моніторингу;
- облік споживання;
- самообслуговування на вимогу;
- об'єднання ресурсів;

Під широкої мережевий доступністю мається на увазі те, що користувачеві доступні програмні продукти, ресурси і послуги через мережу, при цьому неважливо, який пристрій використовується. Користувач може використовувати персональний комп'ютер, ноутбук, планшет, мобільний телефон або будь-яке інше термінальне пристрій – головне мати доступ до мережі.

Легка масштабованість полягає в підключенні (або відключенні) додаткових апаратних або програмних пристроїв. Це відбувається без додаткових затримок з постачальником, в автоматичному режимі. «Хмари» оснащені системою моніторингу, що дозволяє стежити за стабільністю роботи і оцінювати доступність.

Наступна характеристика особливо важлива для бізнесу, тому що безпосередньо впливає на грошові ресурси, що витрачаються на IT. Економічно вигідним є те, що при використанні хмарних технологій є облік споживання. Клієнт не витрачає кошти на ресурси, які не використовуються. Йде облік спожитих ресурсів (наприклад, кількість користувачів і транзакцій, використовуваний обсяг зберігання даних), і на основі цього обліку постачальник оцінює в грошовому еквіваленті надані клієнту послуги. Самообслуговування на вимогу дає можливість клієнту самому управляти обчислювальними потребами. До таких потреб відносять серверний час, швидкість доступу і обробки даних, обсяг збережених даних. Клієнт може здійснювати таке управління без безпосереднього контакту з постачальником послуг. Нарешті, під об'єднанням ресурсів мається на увазі те, що провайдер об'єднує ресурси для обслуговування великого числа споживачів в єдиний пул для того, щоб динамічно розподіляти потужності між споживачами при постійній зміні попиту на потужності.

Таким чином, клієнти стежать тільки за основними параметрами (обсяг даних, швидкість доступу і т.п.), а за фактичним розподілом ресурсів стежить постачальник послуг.

1.3 Моделі розгортання хмарних технологій

Як правило, виділяють наступні моделі розгортання хмари: приватна, публічна і гібридна хмара. Основними моделями є публічне хмара (public cloud) і приватна хмара (private cloud).

Публічне хмара надає хмарні служби та ресурси великій кількості клієнтів, використовуючи загальнодоступні ЦОД. Приватне ж хмара надає власну інфраструктуру організації. Говорячи про публічну хмарі, варто відзначити, що вона дозволяє перевести всі відповідні витрати в операційні витрати і забезпечує швидкий і бюджетний запуск IT-рішення. При застосуванні приватної хмари капітальні вкладення зберігаються, але при цьому зберігається повний контроль IT-інфраструктури.

Якщо говорити докладніше про приватну хмару, то потрібно зазначити, що приватна хмара - це інфраструктура, яка розташовується в межах однієї організації. Приватна хмара створено для того, щоб задовольнити потреби внутрішнього робочого персоналу, забезпечуючи високий рівень безпеки даних.

На сьогоднішній день бізнес пред'являє все більше вимог до IT-технологій. Приватна хмара дозволяє вирішити такі завдання, як надання великого числа бізнес-сервісів і оптимізація витрат. Розгорнувши приватну хмару, компанія зменшує ризики, пов'язані з інформаційною безпекою, і гарантує високу доступність IT-ресурсів, незважаючи на можливе велике завантаження серверів.

Говорячи про плюси приватної хмари, можна відзначити наступне.

1. У порівнянні з публічною хмарою компанія має більше можливостей для контролю за IT-інфраструктурою, тому що всі її компоненти залишаються на стороні компанії.

2. Високий рівень безпеки. Це забезпечується тим, що сервіс споживає одна організація, тому інфраструктура може бути оптимально налаштована під вимоги до захисту даних в даній організації.

3. Висока продуктивність. Вона пов'язана, зокрема, з тим, що всі операції відбуваються в рамках внутрішніх мережевих екранів і засобів захисту периметра корпоративної мережі. Завдяки цьому, передача даних проходить швидко.

4. При впровадженні приватної хмари підвищується оперативність роботи IT-відділу – в будь-який момент він може швидко розгорнути потрібний сервіс. IT-фахівцям варто тільки «підняти» віртуальну машину з шаблону і встановити необхідний сервіс.

Незважаючи на переваги, приватні хмари мають ряд недоліків:

1. Значні витрати на всіх етапах життєвого циклу хмари. На етапі розгортання потрібні інвестиції в обладнання і ПЗ.

Крім цього, приватною хмарою необхідно управляти, що тягне за собою витрати на адміністрування і залучення фахівців.

2. У порівнянні з публічною хмарою, ризики втрати працездатності сервісів або втрати даних через фізичних загроз набагато більш істотні.

3. Компанія може зіткнутися з нестачею місця в хмарі, коли ресурсів інфраструктури може виявитися недостатньо.

Аналізуючи вищезазначене, можна зробити висновок, що найбільш істотним недоліком приватної хмари є необхідність витрати значної кількості людських і матеріальних ресурсів для його створення і подальшої роботи. Це істотно впливає на фінансовий стан організації.

Переходячи до розгляду публічної хмари, варто сказати, що публічна хмара – це інфраструктура, призначена для вільного використання декількома організаціями. Ця модель хмари може перебувати у власності кількох компаній. Незважаючи на це, слово «публічна» не означає, що дані користувачів доступні абсолютно всім. У публічній хмарі реалізуються механізми безпеки для контролю доступу. Простота настройки і низька

вартість – це основні переваги розгортання публічної хмари. Провайдер робить всю роботу, пов'язану зі створенням хмари, а клієнт тільки налаштовує необхідну йому кількість ресурсів.

У разі використання публічного хмари споживач використовує інфраструктуру стороннього провайдера, що створює безліч можливостей для ефективного використання та перерозподілу ресурсів. Публічні хмарні сервіси відрізняються простотою і ефективністю використання, так як для доступу до додатків клієнтам не потрібно нічого, крім стабільного підключення до Інтернету.

Говорячи про переваги публічного хмари, варто відзначити наступне:

1. Простота і ефективність використання.

2. Щоб отримати доступ до додатків потрібно тільки стабільне інтернет-з'єднання.

3. Використання публічної хмари дає можливість скоротити витрати на IT за рахунок відсутності витрат на обладнання і ПЗ.

4. Гнучкість і масштабованість: публічне хмара дозволяє оплачувати саме стільки ресурсів, скільки потрібно в даний момент, та регулювати цей параметр в більшу або меншу сторону.

5. Скорочення часу на обслуговування інфраструктури.

6. Виключається ризик простою бізнес-процесів через серверних аварій, так як сервера додатків знаходяться в хмарі. Віртуальні сервера провайдерів найчастіше налаштовані на потужної фізичної базі і розміщені в великих дата-центрах, де можливий час простоїв обчислюється хвилинами в рік.

7. Використання публічних хмар і відсутність контакту користувачів зі складним комп'ютерним обладнанням дозволяє відмовитися від послуг додаткових IT-фахівців.

Однак модель публічного хмари має деякі недоліки:

1. Головним мінусом публічної хмари є відсутність можливостей для контролю з боку організації, так як працездатність послуг повністю

підпорядкована постачальнику послуг.

2. Повільна швидкість: продуктивність публічних хмарних сервісів безпосередньо залежить від стабільності інтернет-з'єднання, тому в деяких випадках передача даних може бути досить повільним. При оперуванні великими обсягами даних публічні хмари поступаються приватним по продуктивності.

3. Слабка захищеність даних є характерною рисою публічних хмарних середовищ. Захист приватного хмари на порядок надійніше.

Крім приватної і публічної хмари існує гібридна хмара (hybrid cloud).

Гібридна хмара – це модель розгортання хмарної інфраструктури, що забезпечує комбінацію приватної і публічної хмар і поєднує в собі переваги кожної окремо. Поєднання цих двох моделей дозволяє компанії, яка вже має приватну хмару, використовувати ресурси публічної хмари.

Таким чином, у організації є можливість при необхідності розширити власну інфраструктуру за рахунок обчислювальних ресурсів публічної хмари. Отже, при виборі гібридної хмари, компанія отримує контроль і безпеку приватної хмари з масштабами і перевагами публічної хмари. Особливості гібридної хмари:

1. Розширення можливостей приватного хмари. Гібридна хмара дозволяє користувачам мережі організувати доступ до необхідних додатків в приватної хмарі через публічну хмару, при цьому безпеку приватної хмари залишається тією ж.

2. Перерозподіл навантаження. Гібридна хмара дозволяє при необхідності переносити частину навантаження з приватного хмари в публічне, що забезпечує високий рівень продуктивності.

3. Збереження даних. Для підвищення рівня збереження даних гібридна хмара дозволяє при необхідності зберігати в зашифрованому вигляді в публічної хмарі «бекапи» з приватної хмари.

4. Мобільність. Завдяки можливості організації доступу до певних додатків з приватного хмари через публічне хмара, забезпечується робота з

даними додатками з будь-якої точки світу за наявності підключення до інтернету.

1.4 Моделі обслуговування хмарних технологій (IaaS, PaaS, SaaS)

Концепція хмарних обчислень характеризується моделями (рівнями) обслуговування, які виконують певні функції. Хмара надає такі рівні обслуговування:

- інфраструктура як сервіс (Infrastructure as a Service, IaaS);
- платформа як сервіс (Platform as a Service, PaaS);
- програмне забезпечення як сервіс (Software as a Service, SaaS).

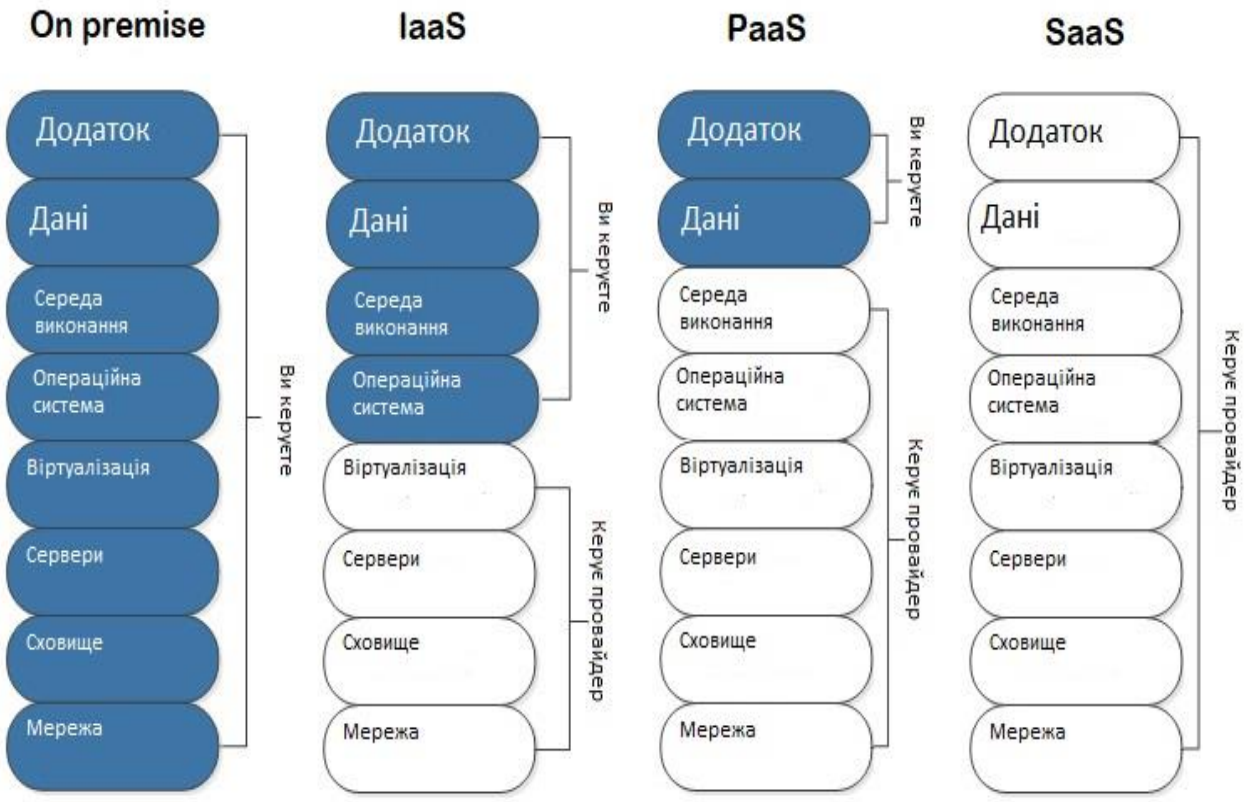

Рисунок 1.1 – Моделі керування хмарних технологій

Інфраструктура як сервіс, IaaS – це надання комп'ютерної інфраструктури як послуги на основі концепції хмарних обчислень. Ця модель обслуговування складається з фізичних активів – мережевих

пристроїв, серверів, дисків і т.п. При взаємодії з IaaS користувач не керує базовою інфраструктурою. Він керує сховищами даних, операційною системою, розгортає додатки і елементи мережі.

IaaS позбавляє компанію від необхідності підтримки складних IT-інфраструктур, центрів обробки даних, клієнтських і мережевих інфраструктур. Це дозволяє зменшити пов'язані з цим капітальні витрати і поточні витрати.

Платформа як сервіс, PaaS – це надання інтегрованої платформи для розробки, тестування, розгортання і підтримки веб-додатків як послуги. PaaS – це модель обслуговування, коли клієнт прямо отримав можливість використовувати хмарну інфраструктуру для розміщення базового програмного забезпечення і для подальшого розміщення на ньому програм. До складу такої платформи входять інструменти для створення і тестування прикладного програмного забезпечення. Ці інструменти надаються хмарним провайдером. PaaS, як інтегрована платформа для розробки, тестування, розгортання і підтримки веб-додатків, дозволяє весь перелік операцій по розробці, тестування і розгортання веб-додатків виконувати в одній інтегрованому середовищі. Такий підхід виключає витрати на підтримку окремих середовищ для кожного етапу розробки додатків. Здатність створювати вихідний код і надавати його в загальний доступ всередині команди розробки значно підвищує продуктивність по створенню додатків на основі PaaS.

Сервіс як послуга, SaaS – модель розгортання програми, яка має на увазі надання додатки кінцевому користувачеві як послуги на вимогу. Концепція SaaS дає можливість використовувати програмне забезпечення як послугу і робити це віддалено через Інтернет. Це дозволяє клієнтові не купувати програмний продукт, а лише тимчасово користуватися ним при виникненні необхідності. В даному випадку, основна перевага моделі SaaS для клієнта полягає у відсутності витрат, пов'язаних з установкою, оновленням і підтримкою працездатності обладнання і програмного

забезпечення, що працює на ньому.

Для моделі SaaS характерно наступне:

– додаток можна використовувати віддалено;
– оплата за додаток стягується або як щомісячна абонентська;
– плата, або на основі сумарного обсягу транзакцій;
– за підтримку програми не потрібна додаткова оплата;
– регулярне автоматичне оновлення;
– додатком можуть користуватися одночасно кілька клієнтів;
– цільова аудиторія SaaS - кінцеві споживачі.

Для більш точного опису трьох моделей обслуговування в таблиці 1.1 описані їх характеристики, переваги, недоліки і ризики.

1.5 Хмарні тенденції

Світовий ринок інфраструктури як послуги (IaaS) зріс на 31,3% в 2018 році до 32,4 мільярда доларів, що збільшилося з 24,7 мільярда доларів у 2017 році, за даними Gartner, Inc. Amazon знову став постачальником № 1 на ринку IaaS у 2018 році, після чого від Microsoft, Alibaba, Google та IBM. У 2018 році п'ятірка найбільших постачальників послуг IaaS становила майже 77% світового ринку IaaS, що склало менше ніж 73% у 2017 році. Консолідація ринку триватиме до 2019 року, обумовлена високими темпами зростання для провідних постачальників, які зазнали сукупності зростання на 39% з 2017 по 2018 рік порівняно зі скромнішим зростанням на 11% для всіх інших постачальників за той же період.

Amazon продовжує лідирувати на світовому ринку IaaS з прогнозованим доходом у 15,5 мільярдів доларів у 2018 році, що на 27% більше, ніж у 2017 році. Найбільший з постачальників послуг IaaS, Amazon,

Таблиця 1.1 Моделі обслуговування. Характеристики, переваги, недоліки та ризики

Моделі сервісів	Характеристики	Переваги	Недоліки та ризики
IaaS	1. Як правило, не залежить від платформи; 2. Зниження витрат на інфраструктуру; 3. Оплата за фактом використання; 4. Автоматичне масштабування.	1. Зниження витрат на апаратне забезпечення та трудові ресурси; 2. Зниження ризику втрати інвестицій; 3. Низький поріг впровадження; 4. Плавне масштабування.	1. Ефективність і продуктивність залежать від постачальника послуги; 2. Потенційно великі довгострокові витрати; 3. Централізація вимагає нових методів захисту та безпеки.
PaaS	1. Використовує інфраструктуру хмари; 2. Забезпечує методи динамічного управління проектами.	Плавне розгортання версій.	Централізація вимагає нових методів захисту та безпеки, які гарантують, що шкідливі програми не зможуть використовувати уразливості в програмній платформі.
SaaS	1. Інтерфейс; 2. Взаємодія за допомогою API (інтерфейс прикладного програмування); 3. Семантична сумісність.	1. Зниження витрат на апаратне забезпечення та трудові ресурси; 2. Зниження ризику втрати інвестицій; 3. Регулярне оновлення.	Централізація вимагає нових методів захисту та безпеки, які пов'язані з конфіденційністю даних клієнта.

становить майже половину всього ринку IaaS. Він продовжує агресивно

розширюватися на нові IT-ринки через нові сервіси, а також придбання, розширюючи основний хмарний бізнес.

Microsoft забезпечила позицію №2 на ринку IaaS з доходом понад 5 мільярдів доларів у 2018 році, порівняно з 3,1 мільярда доларів у 2017 році. Microsoft надає свої можливості IaaS завдяки інноваційному та відкритому пропозиціям Microsoft Azure, що продовжує зміцнювати свою позицію провідного IaaS постачальник.

Домінуючий постачальник IaaS у Китаї Alibaba Cloud зазнав найсильнішого зростання серед провідних постачальників, зроставши 92,6% у 2018 році. Компанія побудувала екосистему, що складається з керованих постачальників послуг (MSP) та незалежних постачальників програмного забезпечення (ISV). Його успіх минулого року був зумовлений агресивними інвестиціями в НДДКР у його портфоліо пропозицій, особливо порівняно з його високопоставленими постачальниками. Alibaba має фінансові можливості продовжувати цю тенденцію та інвестувати в глобальну експансію. Google увійшов на місце №4, зроставши на 60,2% доходів від 2017 року.

1.6 Хмарний провайдер – Amazon Web Services

1.6.1 Amazon Elastic Compute Cloud (Amazon EC2)

Amazon Elastic Compute Cloud (Amazon EC2) – це веб-сервіс, який забезпечує безпечні, масштабовані обчислювальні ресурси в хмарі. Він розроблений, щоб зробити хмарні обчислення простіше для розробників. Простий веб-інтерфейс (рис 1.2) сервісу Amazon EC2, який дозволяє отримати і налаштувати інстанси з мінімальною кількістю дій. Надається повний контроль над обчислювальними ресурсами і дозволяє працювати на перевірених обчислювальному середовищі Amazon. Сервіс Amazon EC2 дозволяє скоротити час, необхідний для отримання та завантаження нових екземплярів сервера до хвилин, що дозволяє швидко масштабувати ресурси,

як вгору, так і вниз, як з урахуванням мінливих вимог. Amazon EC2 змінює економіку обчислень, що дозволяє платити тільки за ресурси, які ви реально використовували. Сервіс Amazon EC2 надає розробникам інструменти, щоб створювати відмовостійкі додатки і ізолювати їх.

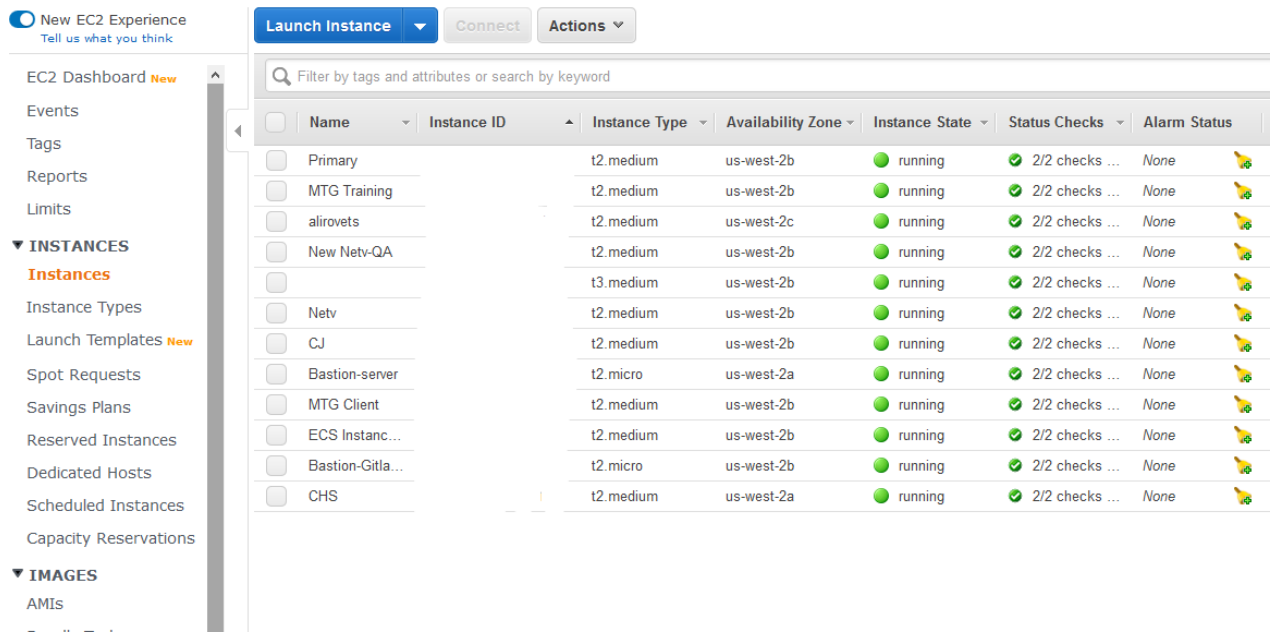

Рисунок 1.2 – Інтерфейс Amazon EC2

Є три основних типи інстанси: On-Demand, Reserved, Spot Instances. On-Demand – ви платите за обчислювальні потужності погодинну оплату, без довгострокових зобов'язань або авансових платежів. Ви можете збільшувати або зменшувати обчислювальні потужності в залежності від вимог вашої програми і платити тільки зазначену погодинну ставку для примірників, яку ви використовуєте.

Reserved Instances – надається значна знижка (до 75%) у порівнянні з цінами інстанси On-Demand. Крім того, коли Reserved Instances призначені конкретній зоні доступності, вони забезпечують резерв потужності, що дає вам додаткову впевненість у вашій здатності до запуску тих випадках, коли ви в них потребу. Але необхідно проплатити наперед від 1 до 3 років використання.

Spot Instances – дозволяє економити до 90% від ціни On-Demand.

У Amazon дуже багато машин, і щоб вони не простоювали в холосту – продають їх за дуже низькими цінами, але при цьому не гарантують час використання вам і можуть в будь-який момент відключити вас. Також застосовується алгоритм ставок. Якщо на даний момент є 10 вільних машин, ви готові заплатити максимально 5 центів за машину і взяти 5 машин – вам дадуть 5 інстанси, але через 10 хвилин приходить інший користувач і робить ставку 6 центів і хоче взяти 8 машин – то у вас заберуть 3 машини. У таких випадках Amazon оповіщає вас, відправляю на машину сигнал, за 2 хвилини до вимкнення. Що значить максимально 5 центів? Це означає, що якщо зараз ціна за ці інстанси 2 центи, то з вас будуть знімати 2 ціна, а не 5, але якщо ціна почне зростати то з вас будуть брати більше грошей, але як тільки ціна стане більше 5 центів – у вас заберуть інстанси.

Spot Fleet Instances – об'єднання декількох Spot Instances і підтримання кількості машин. Повертаючись до минулих ситуації, якби ви користувалися Spot Fleet Instances – то як тільки звільнилося б ще 3 машини – їх би додали в ваш флот. Але має обмеження за кількістю машин у флоті рівне 2000 і ціна за машину не може перевищувати On-Demand ціну такого ж інстанси. В обох варіантах – платите за годину використання, не дивлячись на те що могли використовувати інстанси всього 1 хвилину з цієї години [1].

1.6.2 Amazon Simple Storage Service (Amazon S3)

Amazon Simple Storage Service – це об'єктне сховище з простим веб-інтерфейсом для зберігання і отримання будь-якої кількості даних з інтернету. Він гарантує 99.9% відмовостійкості, і зберігання більше трильйонів об'єктів по всьому світу.

Використовується як зовнішній сервіс через надається API, його не можна підключити до робочої машині як звичайний диск. Платіть за використовуваний обсяг пам'яті і кожну операцію – отримання, переміщення, копіювання, додавання даних і т.д.

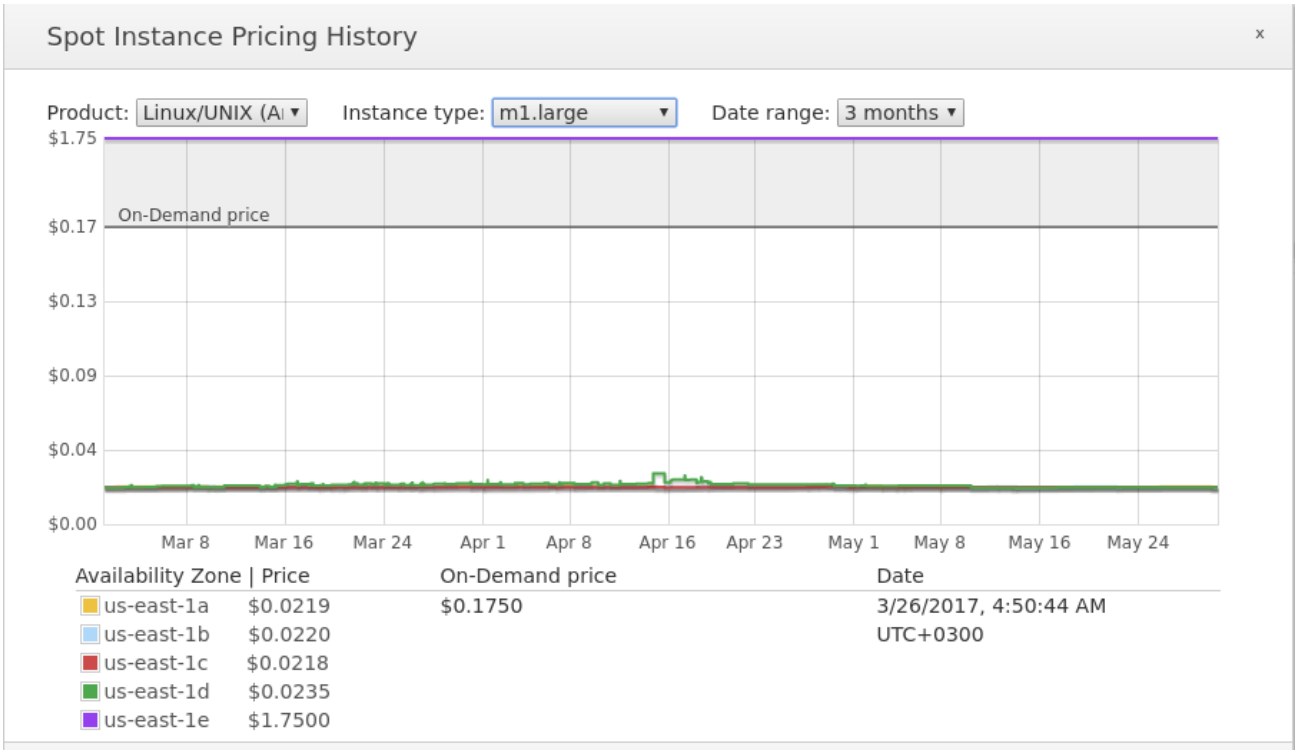

Рисунок 1.3 – Аналіз історії цін Amazon Spot Instances

1.6.3 Amazon Elastic Block Store (Amazon EBS)

Amazon Elastic Block Store – надає постійні обсяги для зберігання даних для використання з Amazon EC2 в хмарі AWS. Кожен Amazon EBS тому автоматично дублюється в своїй зоні доступності, щоб захистити вас від збою компонентів, забезпечуючи високу доступність і надійність. Тома Amazon EBS забезпечують послідовне читання з низькою затримкою, необхідної для виконання вашої роботи. З Amazon EBS, можна масштабувати використання вгору або вниз за лічені хвилини і при цьому платити тільки за використовувані ресурси.

Amazon EBS призначений для робочих навантажень додатків, які отримують вигоду від тонкої настройки продуктивності, вартості і потужності. Типові випадки використання включають в себе аналіз великих даних (як в Hadoop / HDFS), реляційних і NoSQL баз даних (Microsoft, таких як SQL Server і MySQL або Cassandra і MongoDB), трансляція і обробка журналів (як Kafka і Splunk) і зберігання даних додатки (такі як Vertica і

Teradata). Має обмеження, до одного EBS в один момент часу може бути підключений тільки один інстанс EC2.

1.6.4 Amazon Elastic File System (Amazon EFS)

Amazon Elastic File System – забезпечує просте, масштабується сховище файлів для використання з Amazon EC2 в хмарі AWS. Amazon EFS пропонує простий інтерфейс, який дозволяє створювати і налаштовувати файлові системи швидко і легко. З Amazon EFS, ємність для зберігання еластична і автоматично змінюється, коли файли додаються або видаляються. При установці на Amazon EC2 сервера, файл системи Amazon EFS надає стандартний інтерфейс файлової системи, дозволяє легко інтегрувати Amazon EFS з існуючими додатками і інструментами. Кілька Amazon EC2 сервера можуть отримати доступ до файлової системи Amazon EFS в той же час. Amazon EFS дозволяє забезпечити єдиний джерело даних для робочих навантажень і додатків, що працюють на більш ніж одній Amazon EC2 інстанси.

1.6.5 Amazon Machine Image (AMI)

Amazon Machine Image – являє собою особливий тип віртуального пристрою, який використовується для створення віртуальної машини в Amazon Elastic Compute Cloud (EC2). Він служить в якості основної одиниці застосування послуг, що надаються з використанням EC2. Основним компонентом AMI є образ файлової системи (тільки для читання), який включає в себе операційну систему (наприклад, Linux, UNIX або Windows) і додаткове програмне забезпечення, необхідне для надання служби або її частини. Файлова система AMI стиснута, зашифрована, підписана, розділена на групи по 10 МБ і завантажена в Amazon S3 для зберігання. Файл маніфесту XML зберігає інформацію про AMI, включаючи назву, версію архітектури, ідентифікатор ядра за замовчуванням, ключ дешифрування і дайджести для всіх частин файлової системи.

AMI не включає в себе образ ядра, тільки покажчик на ідентифікатор ядра за замовчуванням, який може бути обраний із затвердженого списку безпечних ядер, підтримуваних Amazon і його партнерами (наприклад, Red Hat, Canonical, Microsoft). Користувачі можуть вибрати інші ядра при завантаженні AMI, а не тільки ті, що надані за замовчуванням.

1.6.6 Amazon Relational Database Service (Amazon RDS)

Код програмам та інструментам, які використовуються при роботі з існуючими базами даних, будуть ефективно інтегровані з сервісом Amazon RDS. Amazon RDS може автоматично створювати резервну копію бази даних і підтримувати ПО бази даних в актуальному стані шляхом оновлення до останньої версії. Завдяки гнучкості сервісу можна без особих зусиль масштабувати обчислювальні ресурси і ємність сховища, пов'язані з використовуваним інстанси реляційної БД. Крім того, Amazon RDS спрощує процес реплікації, що підвищує доступність БД і надійність зберігання даних, а також забезпечує масштабування ресурсів для виконання робочих навантажень з великою кількістю операцій читання, знімаючи обмеження, які накладає використання одного інстансу БД. Як і при роботі з іншими сервісами Amazon Web Services, попередні капіталовкладення не потрібні. Оплата нараховується тільки за використовувані ресурси.

1.6.7 Amazon Elastic Load Balancer (Amazon ELB)

Amazon Elastic Load Balancer – це сервіс що автоматично розподіляє вхідний трафік додатків по декількох цільових об'єктів, таким як інстанси Amazon EC2, контейнери, IP-адреси і функції Lambda. Він може розподіляти трафік додатки з мінливою навантаженням в одній зоні доступності або між декількома зонами доступності. Elastic Load Balancing пропонує три типи балансувальник навантаження, які забезпечують високу доступність, автоматичне масштабування і надійний захист, необхідну для забезпечення відмовостійкості додатків.

1. Appliaction Load Balancer.
2. Network Load Balancer.
3. Classic Load Balancer.

Application Load Balancer найкраще підходить для балансування навантаження трафіку HTTP і HTTPS і забезпечує розширену маршрутизацію запитів, орієнтовану на доставку додатків, які побудовані на базі сучасних архітектур, включаючи мікросервіси і контейнери. Працюючи на рівні окремих запитів (рівень 7), Application Load Balancer направляє трафік на цільові об'єкти в Amazon Virtual Private Cloud (Amazon VPC), спираючись на вміст запиту [2].

Network Load Balancer оптимально підходить для балансування трафіку по протоколах Transmission Control Protocol (TCP), User Datagram Protocol (UDP) і Transport Layer Security (TLS), коли потрібна висока продуктивність. Працюючи на рівні з'єднання (рівень 4), Network Load Balancer направляє трафік на цільові об'єкти в Amazon Virtual Private Cloud (Amazon VPC) і може обробляти мільйони запитів в секунду при збереженні наднизьких затримок. Classic Load Balancer забезпечує базову балансування навантаження між декількома інстанси Amazon EC2 і працює як на рівні запитів, так і на рівні з'єднання. Classic Load Balancer призначений для додатків, які були побудовані в мережі EC2 Classic.

1.6.8 Amazon Virtual Private Cloud (Amazon VPC)

Amazon Virtual Private Cloud дозволяє надати логічно ізольований розділ AWS Cloud (рис 1.4), де ви можете запускати ресурси AWS у віртуальній мережі, яку ви визначаєте. Ви маєте повний контроль над своїм віртуальним мережним середовищем, включаючи вибір власного діапазону IP-адрес, створення підмереж та конфігурацію таблиць маршрутів та мережевих шлюзів. Ви можете використовувати як IPv4, так і IPv6 у своєму VPC для безпечного та легкого доступу до ресурсів та програм. Ви можете легко налаштувати мережеву конфігурацію для вашого Amazon VPC.

Наприклад, ви можете створити загальнодоступну підмережу для своїх веб-серверів, які мають доступ до Інтернету, і розмістити свої резервні системи, такі як бази даних або сервери додатків, у приватній підмережі без доступу до Інтернету. Ви можете використовувати кілька рівнів безпеки, включаючи групи безпеки та списки контролю доступу до мережі, щоб допомогти контролювати доступ до примірників Amazon EC2 у кожній підмрежі.

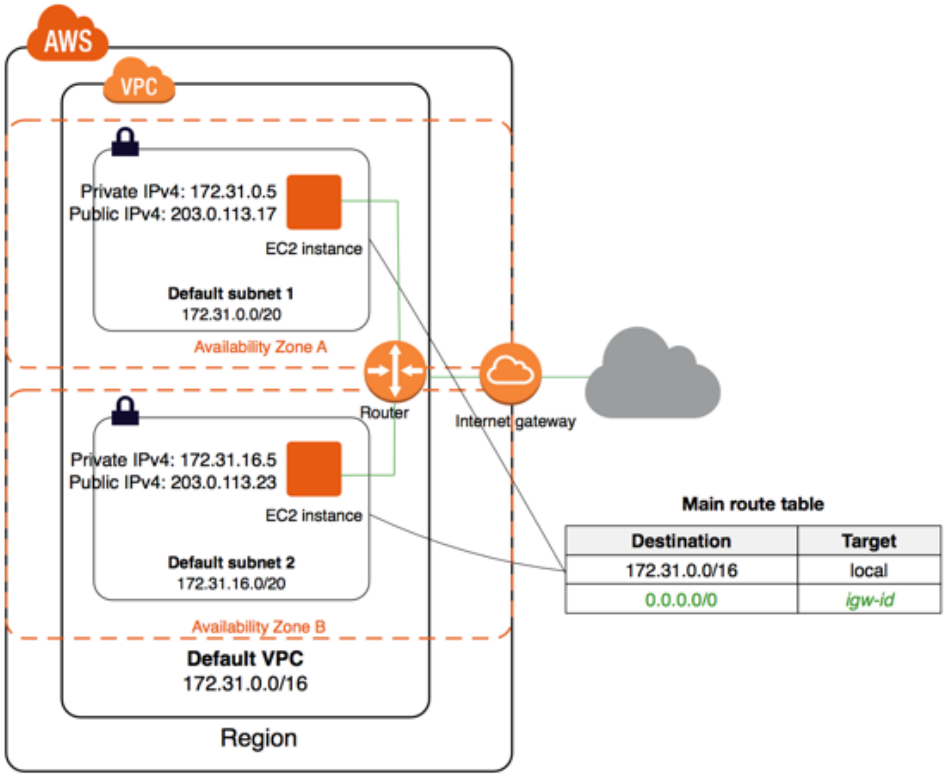

Рисунок 1.4 – Віртуальні мережі Amazon VPC

1.6.9 Amazon Route 53

Amazon Route 53 – це високодоступних масштабований хмарний веб-сервіс системи доменних імен (DNS). Розробники та власники веб-сервісів використовують його як виключно надійний і економічний спосіб перенаправлення кінцевих користувачів до інтернет-додатків за рахунок перетворення доменних імен (наприклад, www.example.com) в формат цифрових IP-адрес (наприклад, 192.0.2.1), використовуваних комп'ютерами.

При цьому Amazon Route 53 повністю сумісний з протоколом IPv6.

Сервіс Amazon Route 53 направляє запити користувачів до інфраструктури AWS, наприклад до інстанси Amazon EC2, балансувальник навантаження Elastic Load Balancing або кошиках Amazon S3. Крім того, він може використовуватися для перенаправлення користувачів в інфраструктуру за межами AWS.

За допомогою сервісу Amazon Route 53 Traffic Flow можна просто управляти глобальним трафіком, використовуючи різні типи маршрутизації (такі як маршрутизація на базі затримки, DNS з урахуванням географічного положення, географічна близькість і циклічний зважений алгоритм), поєднуючи їх з можливістю перекидання сервісу DNS і створюючи в внаслідок відмовостійкі архітектури з низькою затримкою. Використовуючи нескладний візуальний редактор Amazon Route 53 Traffic Flow, можна просто управляти маршрутизацією кінцевих користувачів до адрес додатків як в рамках одного регіону AWS, так і при розподілі трафіку по всьому світу. Крім того, в сервісі Amazon Route 53 можна зареєструвати доменне ім'я: при покупці доменів (наприклад, example.com) і управлінні ними Amazon Route 53 автоматично налаштує для них параметри DNS.

1.7 Git

Git (Global Information Tracker) – це розподілена система керування версіями яка є різновидом VCS (Version Control System). А VCS – це програма для роботи з інформацією що постійно змінюється. Таке ПЗ може зберігати безліч версій одного і того ж файлу (документа) і повертатися до попереднього стану (версії). На рисунку 1.4 можно побачити схему роботи з віддаленим репозіторієм одночасно декількох розробників.

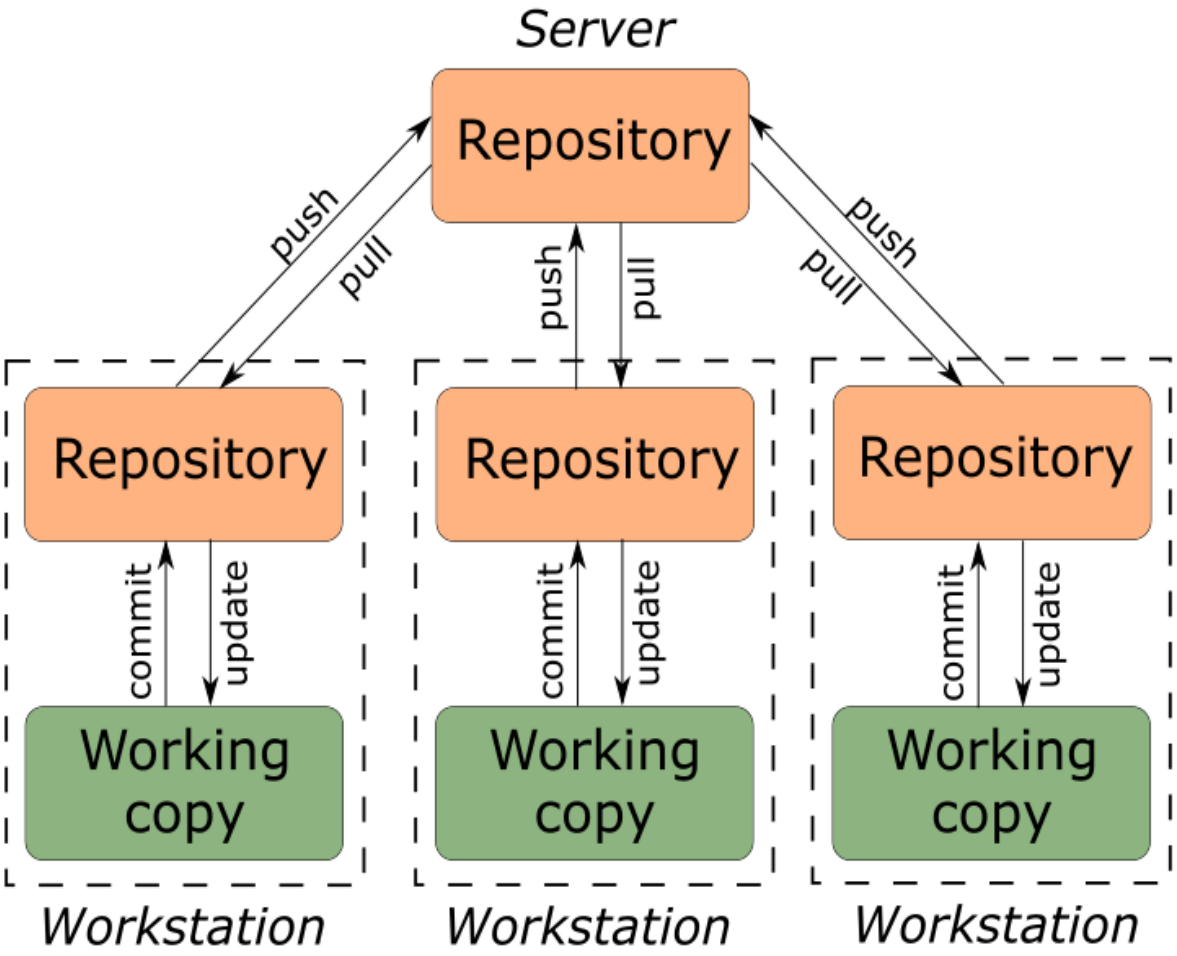

Рисунок 1.4 – Схема роботи Git

1.7.1 Репозіторій для зберігання інфраструктури

Для того, шоб ефективно розробляти, тестувати, керувати, та модернізувати інфраструктуру проекту як код потрібно використовувати надійне сховище з системою контролю версій. Одними з найяскравіших представників даних послуг виступають такі платформи як GitHub і GitLab.

Зберігання інфраструктури проекту в репозіторії (рис. 1.5) значно спрощує роботу над проектом одночасно декількома розробниками, та надає доступ до актуальної та тестових версій інфраструктури [5].

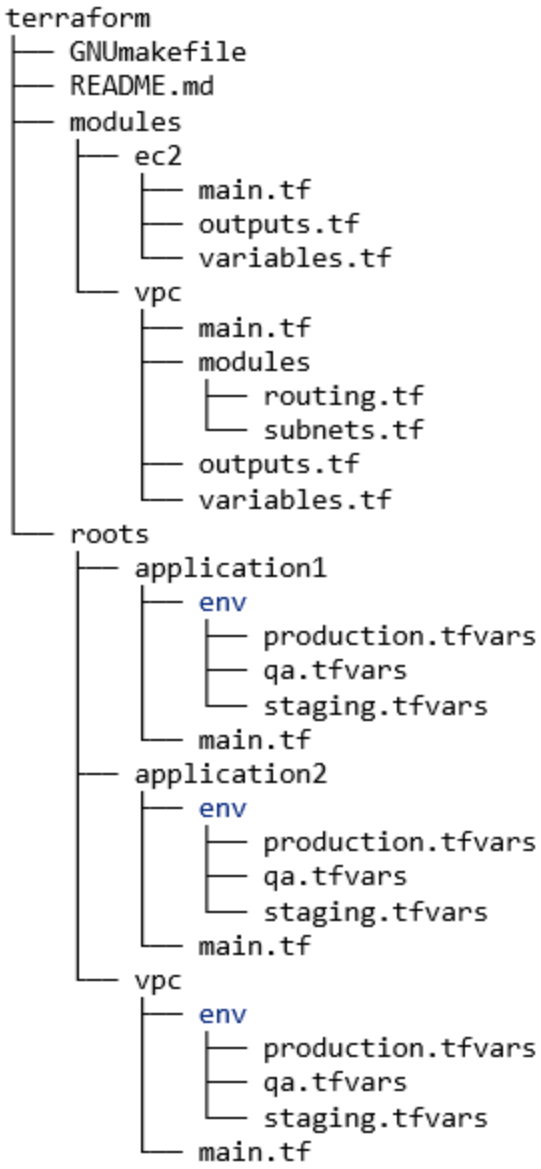

Рисунок 1.5 – Структура зберігання типового проекту Terraform

1.8 Terraform як інструмент керування інфраструктурою

Terraform – це багатохмарний інструмент (IaC) від HashiCorp, написаний на Go Language, використовуючи (HCL) HashiCorp Config Language. Інструмент командного рядка з відкритим кодом, який може використовуватися для забезпечення інфраструктури на багатьох різних платформах і службах, таких як IBM, AWS, GCP, Azure, OpenStack, VMware та багато іншого. Використовуючи модель на основі плагінів для підтримки провайдерів та постачальників послуг, надаючи їй можливість підтримувати

майже будь-яку службу, що виставляє API.

Це хмарно-агностичний інструмент, який дозволяє будувати складні, керовані версіями, спільні, неоднорідні та одноразові системи з дуже високою продуктивністю. Він забезпечує вбудовані залежності між кожним рівнем, які обробляють автоматичне забезпечення інфраструктури, як у публічній, так і в приватній хмарі [3]. Опис коду здійснюється за допомогою людино-читаної мови розмітки даних (рис 1.6), орієнтованій на зручність введеня-виведеняе типових структур даних – YAML.

```
terraform {
 required_version = ">= 0.11"
 backend "azurerm" {
  storage_account_name = "__terraformstorageaccount__"
    container_name       = "terraform"
    key                  = "terraform.tfstate"
  access_key  ="__storagekey__"
  }
  }
resource "azurerm_resource_group" "dev" {
  name     = "PULTerraform"
  location = "West Europe"
}

resource "azurerm_app_service_plan" "dev" {
  name                = "__appserviceplan__"
  location            = "${azurerm_resource_group.dev.location}"
  resource_group_name = "${azurerm_resource_group.dev.name}"

  sku {
    tier = "Free"
    size = "F1"
  }
}

resource "azurerm_app_service" "dev" {
  name                = "__appservicename__"
  location            = "${azurerm_resource_group.dev.location}"
  resource_group_name = "${azurerm_resource_group.dev.name}"
  app_service_plan_id = "${azurerm_app_service_plan.dev.id}"
```

Рисунок 1.6 – Приклад коду у форматі YAML

1.8.1 Переваги використання Terraform

Використання інфраструктури як коду в рамках процесу розгортання має ряд переваг для робочого процесу:

– швидкість – автоматизація перевершує ручну роботу з навігацією по

інтерфейсу для розгортання та підключення ресурсів;

– надійність – з великим набором інфраструктури стає дуже просто неправильно налаштувати ресурс або надати послуги в неправильному порядку. З IaC ресурси будуть налаштовані точно так, як оголошено, і неявні / явні залежності можуть бути використані для забезпечення порядку створення;

– експериментація – завдяки легкості розгортання інфраструктури експериментальні зміни можна легко дослідити за допомогою зменшення ресурсів для мінімізації витрат. Після затвердження все може бути розширено для виробництва;

– найкращі практики – розробники завжди прагнуть використовувати відомі найкращі практики інженерії програмного забезпечення, де це можливо. Написання коду для розробки та розгортання інфраструктури полегшує це на арені надання ресурсів в хмарі, використовуючи встановлені методики, такі як написання модульного, настроюваного коду, призначеного для контролю версій. Це призводить нас до того, що ми розглядаємо нашу інфраструктуру як дещо програмне забезпечення саме по собі і спрямовує нас у напрямку культури DevOps.

1.8.2 Управління Terraform

Управління Terraform здійснюється через дуже простий у користуванні інтерфейс командного рядка (рис. 1.6). Terraform CLI – це лише додаток командного рядка. Terraform CLI як аргумент приймає підкоманди, такі як "apply" або "plan". Виконання усіх команд відбувається на стороні розробника.

```
~/TerraformScripts $terraform plan ~/TerraformScripts/
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but
will not be persisted to local or remote state storage.


The Terraform execution plan has been generated and is shown below.
Resources are shown in alphabetical order for quick scanning. Green resources
will be created (or destroyed and then created if an existing resource
exists), yellow resources are being changed in-place, and red resources
will be destroyed. Cyan entries are data sources to be read.

Note: You didn't specify an "-out" parameter to save this plan, so when
"apply" is called, Terraform can't guarantee this is what will execute.

+ azurerm_resource_group.LinuxTest01
    location: "westus"
    name:     "LinuxTest01"
    tags.%:   "<computed>"


Plan: 1 to add, 0 to change, 0 to destroy.
~/TerraformScripts $
```

Рисунок 1.7 – Приклад використовування команди "plan" Terraform CLI

Також керувати Terraform можна за допомогою Terraform Cloud (рис 1.7). Ця послуга розміщена на веб-сайті https://app.terraform.io. Предоставляються безкоштовні рахунки для невеликих команд та платні плани з додатковими наборами функцій для середнього бізнесу.

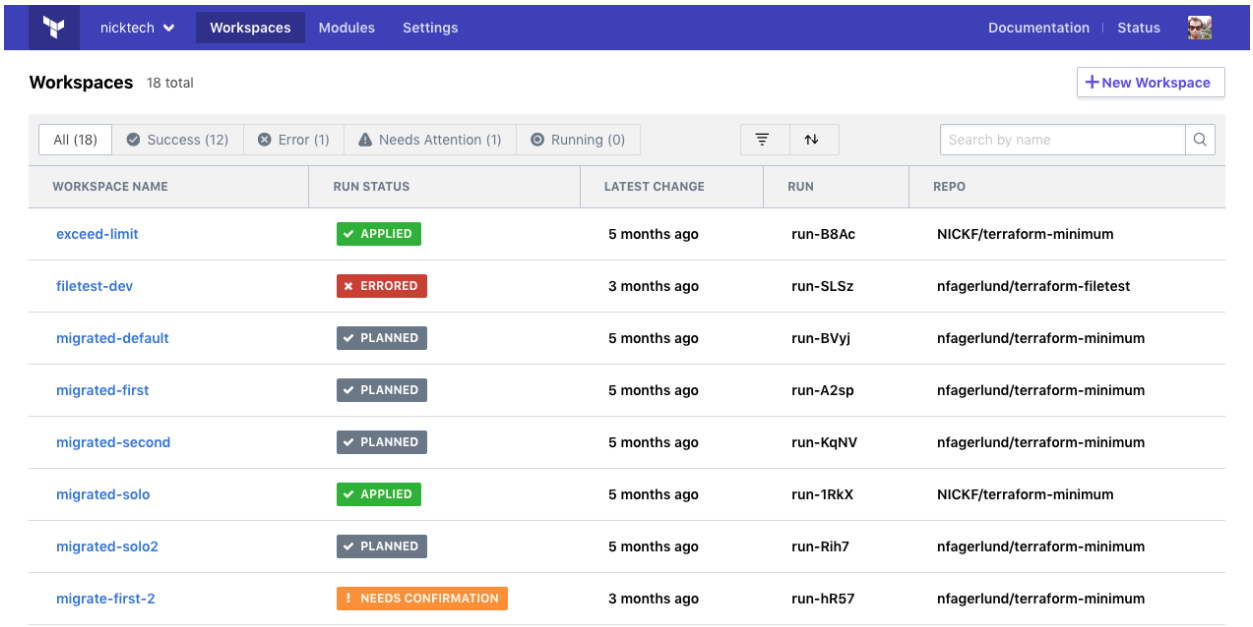

Рисунок 1.8 – Веб інтерфейс сервісу Terraform Cloud

1.8.3 Робота з Terraform

Модель даних Terraform дуже проста: Terraform керує ресурсами, а у ресурсів є атрибути. Кілька прикладів зі світу AWS:

- інстанси EC2;
- том EBS;
- ELB.

Terraform забезпечує зіставлення ресурсів, описаних в файлі конфігурації (.tf), з відповідними ресурсами хмарного провайдера. Таке зіставлення іменується станом (.tfstate), це гігантський файл у форматі JSON.

Також підтримуються вхідні змінні (variables.tf), вони служать параметрами для модуля Terraform, дозволяючи налаштувати параметри модуля без зміни власного вихідного коду модуля та дозволяють ділитися модулями між різними конфігураціями. Коли ви оголошуєте змінні в кореневому модулі конфігурації, ви можете встановити їх значення, використовуючи параметри CLI та змінні середовища. Коли ви оголошуєте їх у дочірніх модулях, викликовий модуль повинен передавати значення в блок модуля.

Інша ключова конструкція Terraform – це «джерело даних» (рис 1.9). Це дозволяє нам посилатися на ресурси, які вже повинні існувати в AWS, дозволяючи нам витягувати з них інформацію для подачі в нові ресурси тощо. Зрозуміти таку модель даних з ресурсами і атрибутами не настільки складно, однак, вона може не цілком збігатися з API хмарного провайдера [4].

Фактично, єдиний ресурс Terraform може відповідати як одному, так і декільком базовим об'єктам хмарного провайдера – або навіть не відповідати жодному. Ось кілька прикладів з AWS (рис 1.9):

- aws_ebs_volume в Terraform відповідає тому AWS EBS;
- aws_instance в Terraform з вбудованим блоком ebs_block_device як в попередньому прикладі відповідає двом ресурсів EC2;

```
resource "aws_instance" "example" {
  ami           = "ami-2757f631"
  instance_type = "t2.micro"
}

resource "aws_ebs_volume" "example-volume" {
  availability_zone = "${aws_instance.example.availability_zone}"
  type              = "gp2"
  size              = 100
}

resource "aws_volume_attachment" "example-volume-attachment" {
  device_name = "/dev/xvdb"
  instance_id = "${aws_instance.example.id}"
  volume_id   = "${aws_ebs_volume.example-volume.id}"
}
```

Рисунок 1.9 – Описання інстансу з томом EBS

При запуску команди terraform apply Terraform оновлює стан, направляючи відповідний запит хмарного постачальнику. Потім порівнює повернуті ресурси з тією інформацією, що записана у вашій конфігурації Terraform. Якщо виявиться яка-небудь різниця, то створюється план, по суті – перелік змін, які потрібно внести в ресурси хмарного провайдера, щоб фактична конфігурація відповідала тій, що вказана у вас в конфігурації. Нарешті, Terraform застосовує ці зміни, направляючи відповідні виклики до API хмарного провайдера. Якщо ви внесете зміни в свій код, повторний запуск плану і застосування команд дозволить Terraform використовувати свої знання про розгорнутих ресурсах (.tfstate), щоб розрахувати, які зміни необхідно внести, будь то створення або знищення. Нарешті, коли треба видалити свою інфраструктуру, просто введіть команду terraform destroy, і вона вийде з ладу.

2 ПЛАНУВАННЯ ТА РОЗРОБКА АРХІТЕКТУРИ

2.1 Визначення із хмарним провайдером

Використання Terraform не обмежується тільки AWS. Теоретично він може працювати з будь-яким провайдером, а також може використовуватись для створення власних центральних систем обробки даних. Для більшості проектів різних розмірів та потреб саме AWS підходить як найкраще. Даний сервіс орієнтован на середній та крупний бізнес, але він також підходить для окремих розробників. Переваги даної платфоми:

– сервіси Amazon є фінансово вигідними для стартапів;

– масштабування та еластичність сервісів;

– без зобов'язань та договорів (сервіси тарифікуються за часом);

– безпека – многофакторна аутентифікація, професійна підтримка 24/7, вбудований брандмауер, служби IAM для відслідковування дій користувачів;

– висока доступність – AWS будує ЦОД в різних географічних регіонах;

– надійність – гнучке керування реплікаціями та достпуність 99,999999999 %.

2.2 Керування інфраструктурою

Для керування інфраструктурою можна використовувати як Terraform, так і AWS CloudFormation. AWS CloudFormation – сервіс від Amazon, що також дозволяє керувати інфраструктурою за практиками IaC. Більш детально різницю між цими сервісами можно розглянути на таблиці 2.1.

Таблиця 2.1 – Порівняння Terraform та CloudFormation

Порівняння	Terraform	CloudFormation
Multiple cloud support	За рахунок використання різних провайдерів-плагінів може працювати з будь-яким великим хмарним провайдером.	Жорстко прив'язаний до Amazon.
Відстеження змін	Якщо зміни відбулися не в коді TF, а на ресурсі, який він відтворив, TF зможе це відстежити	Аналогічна функція з’явилась лише в листапді 2018 року
Умови	Немає підтримки умов (тільки у вигляді тернарних операторів).	Умови підтримуються
Зберігання станів	Дозволяє вибрати декілка видів back-end (локально, S3, shared host)	Зберігається тільки всередені AWS
Імпорт ресурсів	Дозволяє легко імпортувати ресурси які можна взяти під контроль	Не підтримується

2.3 Інфраструктура для веб-додатку

Для того, що б побудувати відмовостійку інфраструктуру для веб-додатку з високою доступністю на базі AWS необхідно спланувати основні сервіси зображені на рисинку 2.1, такі як: EC2, RDS, ELB, Auto Scaling Group, Security Group, Route 53.

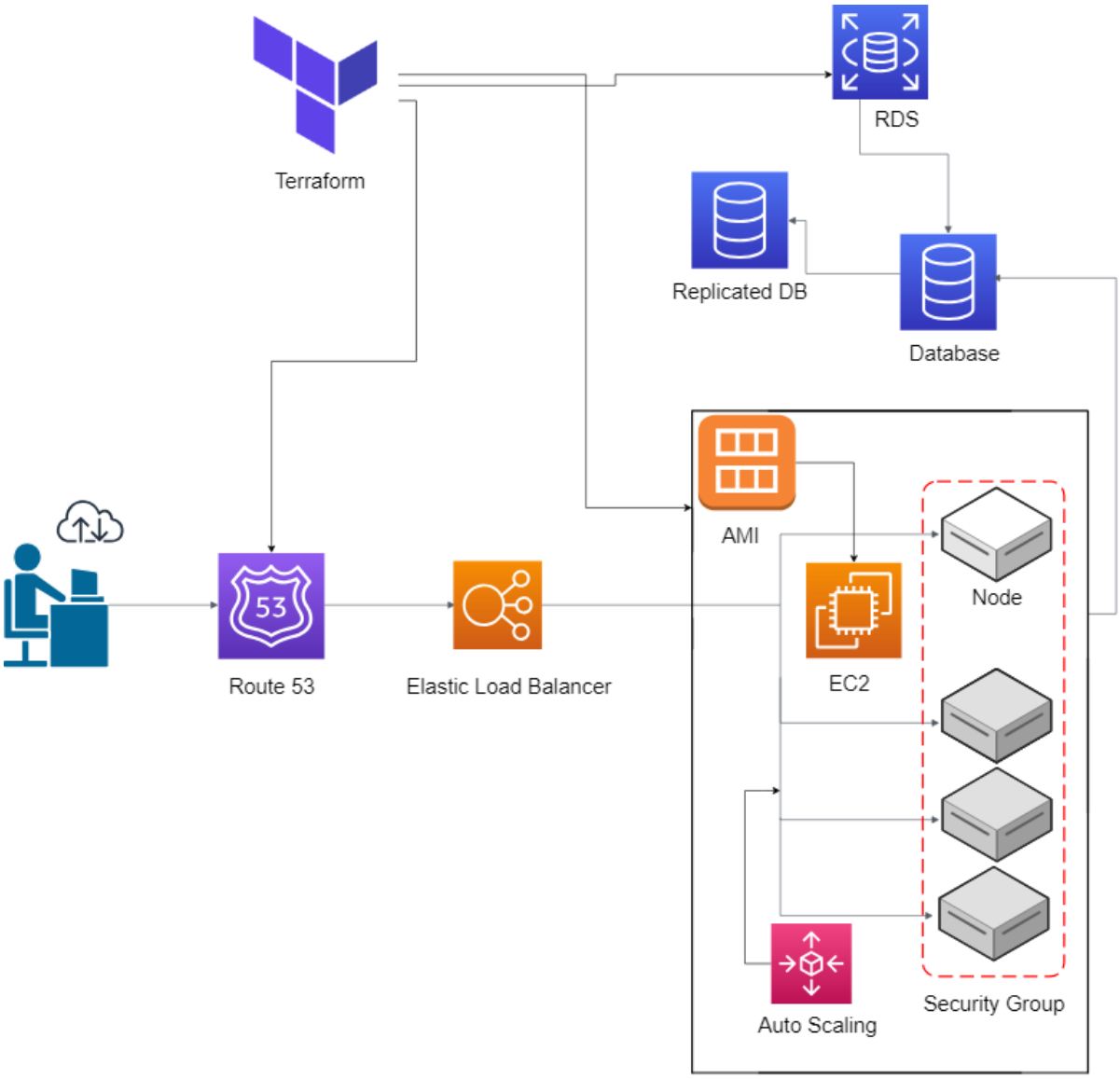

Рисунок 2.1 – Загальна схема проекту

2.3.1 Масштабування для високого навантаження

Масштабування є базовою потребою до будь-якої архітектури, та як відомо є два шляхи для його забезпечення:

– вертикальне – це збільшення продуктивності кожного компонента системи (процесор, оперативна пам'ять, інші компоненти);

– горизонтальне – коли сполучають декілька елементів воєдино, а система в цілому складається з безлічі обчислювальних вузлів, які вирішують загальну задачу, тим самим збільшуючи загальну надійність і доступність системи. Збільшення продуктивності досягається додаванням в систему додаткових вузлів.

Перший підхід непоганий, але є істотний мінус – обмеженість потужності одного обчислювального вузла. Неможливо нескінченно збільшувати частоту обчислювального ядра процесора і пропускну здатність шини. Тому горизонтальне масштабування значно виграє у вертикального, адже при нестачі продуктивності можна додати в систему вузол (або групу вузлів) [6].

2.3.2 База даних RDS

Сервіс Amazon RDS надає можливість використовувати одну з нижчеприведених ядер баз даних:

– Amazon Aurora;
– PostgreSQL;
– MySQL;
– MariaDB;
– Oracle;
– Microsoft SQL Server.

Кожна з наведених баз даних повинна використовуватися в окремих випадках. Одною з найпопулярніших на сьогодняшній день є MySQL, величезний досвід застосування її в різних завданнях, в тому числі і для високонавантажених систем, робить її хорошим вибором. Але для того, щоб

масштабувати систему обробки даних потрібно провести деякі маніпуляції. Дефолтні конфігураційні параметри в MySQL розраховані на мікроскопічні бази даних, що працюють під малими навантаженнями.

Процес оптимізації бази даних MySQL скадається з двох частин — первісне налаштування і коригування параметрів під час роботи. Коригування параметрів в робочому режим багато в чому залежить від специфіки Вашої системи і її моніторингу.

До переваг MySQL можна віднести такі параметри:

– простота – MySQL легко встановити не зважаючи на платформу, існує багато інструментів, включаючи візуальні;

– багато функій – MySQL підтримує більшу частину фунціонала SQL;

– безпека – вбудовано багато функцій безпеки;

– потужність і масштабованість – MySQL може працювати з дійсно великими обсягами даних, і непогано підходить для масштабованих додатків;

– швидкість – нехтування деякими стандартами дозволяє MySQL працювати продуктивніше.

З недоліків:

– відомі обмеження – по визначенню, MySQL не може зробити все, що завгодно, і в ній присутні певні обмеження функціональності;

– питання надійності – деякі операції реалізовані менш надійно, ніж в інших РСУБД;

– застій в розробці – хоча MySQL і є open-source продуктом, робота над нею сильно загальмована. Проте, існує кілька БД, повністю заснованих на MySQL (наприклад, MariaDB).

2.3.3 Балансування навантаження БД

Першим користувача «зустрічає» веб-сервер, на його плечі покладаються завдання віддачі статичних ресурсів і передачі запитів з додатком, після цього відбувається взаємодія із базою даних яка найчастіше

може викликати ефект «bottleneck» (рис. 2.2)

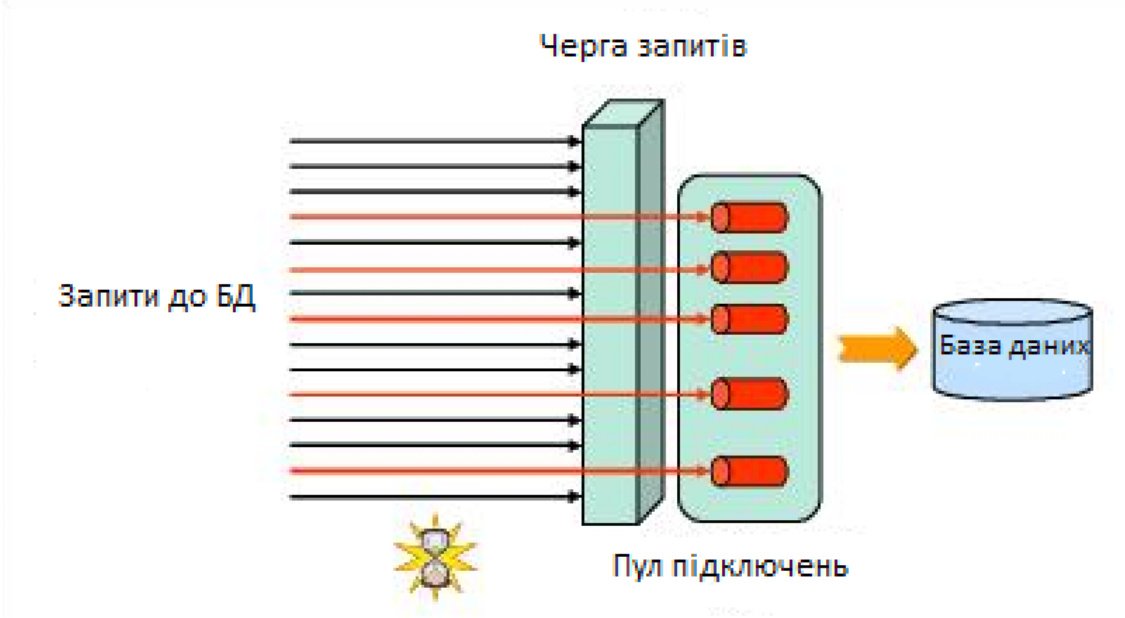

Рисунок 2.2 – Ефект «bottleneck» на прикладі бази даних

Для веб-додатка оптимальним варіантом є сервер з найбільшим числом процесорних ядер (для обслуговування великої кількості паралельно працюючих користувачів). Amazon надає набір Computer Optimized Instances які найкрашим чином підходять для обробки запитів.

Робота бази даних – це численні дискові операції введеня вивведення (запис і читання даних). Тут найкращим варіантом буде сервер з твердотілим накопичувачем SSD. Amazon може запропонувати Storage Optimized Instances, але також підійде і сервер з лінійки General Purpose. Забезпеченням високої доступності бази даних більшою мірою є читання даних ніж запис, тож потрібно кластерізувати базу даних на читання (рис. 2.3).

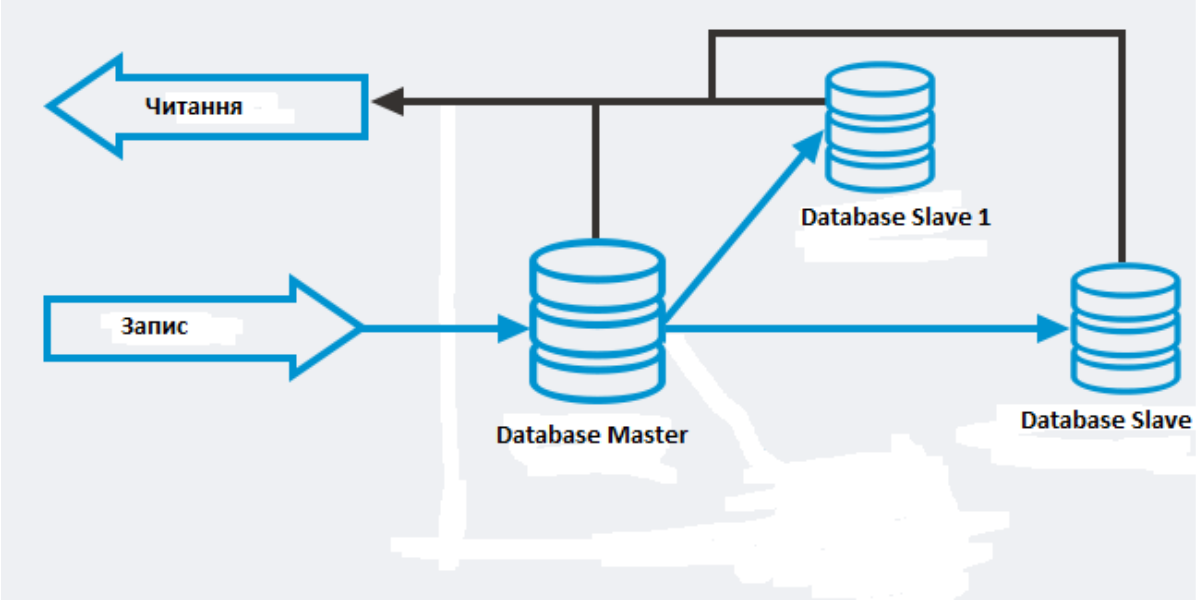

Рисунок 2.3 – Кластерізована БД для читання

Для статичних ресурсів не потрібен ні потужний процесор, ні великий обсяг оперативної пам'яті, найкращим рішенням буде внести їх під зберігання в Amazon Simple Storage Service.

В якості балансувальника навантаження буде використан сервіс Amazon Elastic Load Balancer, він автоматично здійснює контроль за станом доданих в розсилку машин, для цього буде реалізован найпростіший пінг ресурс, який буде відпповідати 200 кодом на запити, саме його вказують балансувальнику.

2.3.4 Балансування навантаження EC2 за допомогою Amazon ELB та Auto Scaling Group

Метою балансування навантаження є оптимізація використання ресурсів, максимізація пропускної здатності, зменшення часу відгуку і запобігання перевантаження будь-якого одного ресурсу. Використання декількох компонентів балансування навантаження замість одного може підвищити надійність і доступність за рахунок резервування. Балансування навантаження передбачає зазвичай наявність спеціального програмного забезпечення або апаратних засобів, таких як багаторівневий комутатор або система доменних імен, як серверний процес.

Балансувальних навантаження знаходится між клієнтами та усіма внутриішніми компонентами та виконує декілька важливих задач:

– виявлення служб – які бекенд-вузли доступні в системі? Які їх адреси (наприклад, як повинен працювати балансувальник навантаження);

– перевірка працездатності – які бекенд-вузли в даний час здорові і доступні для прийому запитів;

– балансування навантаження – який алгоритм слід використовувати для балансування окремих запитів до здорових бекенд-вузлів.

Application Load Balancer працює на рівні запитів (рівень 7), маршрутизуя трафік на цільові об'єкти – інстанси EC2, контейнери та IP-адреси – на основі вмісту запиту. Application Load Balancer ідеально підходить для розширеного балансування навантаження HTTP і HTTPS-трафіку. Він забезпечує розширену маршрутизацію запитів, орієнтовану на доставку додатків, побудованих на базі сучасних архітектур, включаючи мікросервіси і додатки на основі контейнерів. Application Load Balancer спрощує настройку і підвищує безпеку додатка, гарантуючи, що завжди використовуються найновіші версії шифрів і протоколів SSL / TLS.

Як уже зазначалося, класична задача хмарної системи - це динамічне масштабування під мінливе навантаження. Воно може бути вертикальним, коли "на льоту" модифікуються потужності обмеженого числа серверів, або горизонтальним, коли під чергову додаткову порцію запитів запускається новий сервер типової конфігурації. Горизонтальне масштабування в Amazon реалізується за допомогою веб-сервісу Auto Scaling, призначеного для автоматичного запуску або зупинки серверів EC2 в залежності від поточного навантаження (рис. 2.4), а також від певних користувачем умов і розкладу. Зокрема, Auto Scaling дозволяє масштабувати систему динамічно (підлаштовуючись під навантаження) або прогностично (коли добре відомі піки навантаження і можна ставити старт додаткових ресурсів в зазначений час дня, тижня чи місяця). При цьому можна досягти хорошої економії за

рахунок оперативного відключення невикористовуваних серверних екземплярів.

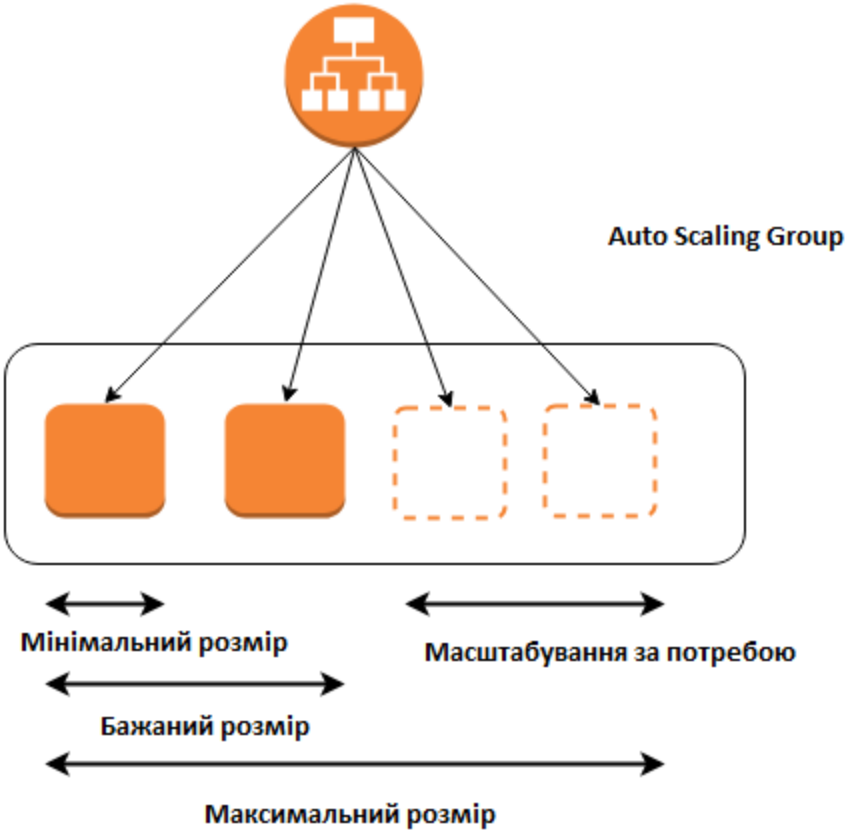

Рисунок 2.4 – Розташування і визначення меж Auto Scaling Group

Для запуску Auto Scaling треба підготувати групи для вирішення прикладних завдань, в яких буде визначено мінімальне, максимальне і рекомендоване число працюючих серверів. Кожна група отримує також свою конфігурацію запуску, визначальну специфіку використовуваних серверів. Крім того, необхідно створити план масштабування: саме він визначить, коли і як будуть працювати механізми горизонтального масштабування. Auto Scaling додатково простежить за станом кожного примірника і при необхідності перезапустить "завислі" сервери.

2.3.5 Route 53

Функції Route 53 допоможуть забезпечити постійну можливість підключення всіх користувачів до веб-додатку. Наприклад, функція відказоустойчивости DNS може виявити свій веб-сайт і перенаправити користувачів у інших місцях, де ваше додаток запущено і працює. Це збільшує доступність вашого веб-сайту і допомагає підтримувати доступність для ваших кінцевих користувачів. Також Route 53 інтелектуально спрямовує трафік на основі складних політик маршрутизації та – за допомогою автоматизованих перевірок здоров'я - подалі від серверів, які можуть бути непрацездатними.

Як і багато служб AWS, Route 53 є послугою з оплатою. З вас буде стягнуто кількість створених та підтримуваних розміщених зон, а також кількість відправлених запитів. Route 53 пропонує потужну політику для забезпечення ефективних запитів DNS. Щойно ви розпочали роботу свого домену, ви можете вибрати політику маршрутизації, яка найкраще відповідає вашим потребам. Щоб максимально використати послугу, потрібно буде правильно зрозуміти функції кожного типу політики.

1. Simple Routing Policy – найпростіший тип маршрутизації;
2. Weighted routing policy – призначаючи різні числові ваги (або "пріоритети") на декілька серверів, що надають веб-сервіс, ви можете направити більший або менший відсоток вашого вхідного трафіку на один конкретний сервер над іншим;
3. Latency-based routing policy – політика на основі затримки спрямовує запити на трафік до сервера, який зможе відповісти з найменшою можливою затримкою (затримкою);
4. Failover routing policy – політика відмови відправить увесь трафік на сервер, який ви встановили як основний, до тих пір, поки цей сервер залишається здоровим;

5. Geolocation routing policy – ця політика дозволяє визначати цільові ресурси на основі географічного розташування користувачів.

Також Route 53 надає можливість розгорнути приватний DNS. Приватні записи DNS дозволяють створювати приватні розміщені зони та маршрутний трафік, використовуючи легко керовані доменні імена у межах ваших VPC. Наприклад, це може дозволити вам швидко перемикатися між ресурсами на основі IP-адреси без необхідності оновлення декількох вбудованих посилань.

3 РОЗРОБКА МОДЕЛІ КЕРУВАННЯ ДЛЯ AMAZON WEB SERVICES

3.1 Веб-складова

В якості веб-додатку використовується CMS Wordpress.

Базововий конфігураційний файл (лістінг 3.1) необхідний для взаємодії бази даних для PHP та Wordpress (рис 3.1) , завдяки якому визначаються основні складові, такі як:

- назва бази даних (`'DB_NAME'`);
- логін (`'DB_USER'`);
- пароль (`'DB_PASSWORD'`);
- ip-адреса та порт (`'DB_HOST'`);
- кодування (`'DB_CHARSET'`);
- аутентіфікаційний ключ (`'AUTH_KEY'`).

Даний файл необхідний для корректної роботи MySQL та, якщо виникають помилки – допомгає в налагодженні.

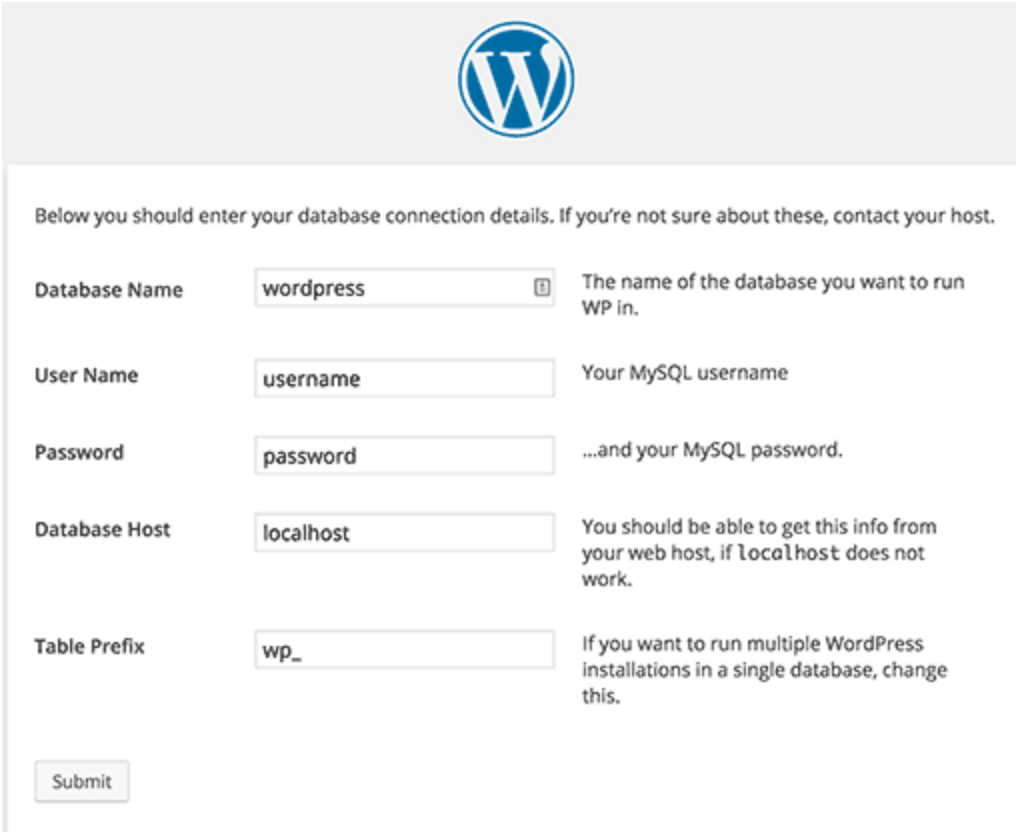

Рисунок 3.1 – Сторінка первинного налаштування Wordpress

Лістинг 3.1

```
<?php

// ** MySQL settings - You can get this info from your web host
** //

define( 'DB_NAME', '${db_name}' );

define( 'DB_USER', '${db_user}' );

define( 'DB_PASSWORD', '${db_pass}' );

define( 'DB_HOST', '${db_host}:${db_port}' );

define( 'DB_CHARSET', 'utf8' );

define( 'AUTH_KEY', 'auth_key' );

define( 'WP_DEBUG', false );

/** Absolute path to the WordPress directory. */
if ( ! defined( 'ABSPATH' ) ) {
     define( 'ABSPATH', dirname( __FILE__ ) . '/' );
}
/** Sets up WordPress vars and included files. */
require_once( ABSPATH . 'wp-settings.php' );
```

Лістінг 3.2 містить в собі дані які вносяться до бази даних в якості прикладу. Таблиця складається з тестових значень для ідентіфікації користувачів та дампу інформації з клієнтської таблиці.

Лістинг 3.2

```
     /*!40101 SET
@OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
     /*!40101 SET
@OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
     /*!40101 SET
@OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
     /*!40101 SET NAMES utf8 */;
     /*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
     /*!40103 SET TIME_ZONE='+00:00' */;
     /*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS,
UNIQUE_CHECKS=0 */;
     /*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
```

```
FOREIGN_KEY_CHECKS=0 */;
     /*!40101 SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
     /*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

     --
     -- Table structure for table `customer`
     --
     use mydb;
     DROP TABLE IF EXISTS `customer`;
     /*!40101 SET @saved_cs_client     = @@character_set_client
*/;
     /*!40101 SET character_set_client = utf8 */;
     CREATE TABLE `customer` (
       `id` int(11) NOT NULL AUTO_INCREMENT,
       `first_name` varchar(30) COLLATE utf8_unicode_ci NOT NULL,
       `last_name` varchar(50) COLLATE utf8_unicode_ci NOT NULL,
       `phone_number` varchar(15) COLLATE utf8_unicode_ci NOT
NULL,
       `email_address` varchar(255) COLLATE utf8_unicode_ci NOT
NULL,
       `city` varchar(80) COLLATE utf8_unicode_ci NOT NULL,
       `state` varchar(2) COLLATE utf8_unicode_ci NOT NULL,
       `date_registered` date NOT NULL,
       PRIMARY KEY (`id`)
     ) ENGINE=MySQL AUTO_INCREMENT=11 DEFAULT CHARSET=utf8
COLLATE=utf8_unicode_ci;
     /*!40101 SET character_set_client = @saved_cs_client */;
```

Файл, наведений в лістингу 3.3, являє собою скриптовий файл, який використовується при створенні образу AMI як User Data, він дозволяє проінсталювати усі компоненти залежностей веб-додатку. Буде встановлено клієнт бази даних MySQL, веб-сервер Apache, фреймворк PHP5, та CMS Wordpress. Після інсталляції файли веб-сторінки для Wordpress будуть надані веб-серверу та визначені права на директорію.

Лістинг 3.3

```
#!/bin/bash
sudo echo "127.0.0.1 `hostname`" >> /etc/hosts
sudo apt-get update -y
sudo apt-get install mysql-client -y
sudo apt-get install apache2 apache2-utils -y
sudo apt-get install php5 -y
sudo apt-get install php5 libapache2-mod-php5 php5-mcrypt
```

```
php5-curl php5-gd php5-xmlrp -y
     sudo apt-get install php5-mysqlnd-ms -y
     sudo service apache2 restart
     sudo wget -c http://wordpress.org/wordpress-5.1.1.tar.gz
     sudo tar -xzvf wordpress-5.1.1.tar.gz
     sleep 20
     sudo mkdir -p /var/www/html/
     sudo rsync -av wordpress/* /var/www/html/
     sudo chown -R www-data:www-data /var/www/html/
     sudo chmod -R 755 /var/www/html/
     sudo cp /var/www/html/wp-config-sample.php /var/www/html/wp-
config.php
     sudo service apache2 restart
     sleep 20
```

3.2 Описання архітектури AWS

Описання архітектури AWS в Terraform зроблено модульно для спрощенного подальшого адміністрування, основні компоненти винесені в файли з відповідними назвами:

- ec2.tf;
- alb.tf;
- security_group.tf;
- vpc.tf;
- vars.tf.

Параметри що визначені в лістінгу 3.4:

- параметр для визначення регіону розгортання (`"us-east-1"`);
- ресурс для зберігання публічних SSH ключів для підключення до інстансів пілся розгортання (`"aws_key_pair"`);
- дані для підключення конфігураційного файлу PHP (`"phpconfig"`) разом із змінними для функційовання бази даних (`"mysql"`);
- модуль резервування для балансувальника навантаження (`"alb"`) із визначеним типом балансування (`"application"`);
- ресурс для визначення типу інстансу EC2 та залежність від бази даних (`"aws_instance"`).

Лістинг 3.4

```
resource "aws_key_pair" "keypair1" {
  key_name   = "${var.stack}-keypairs"
  public_key = "${file("${var.ssh_key}")}"
}

data "template_file" "phpconfig" {
  template = "${file("conf.wp-config.php")}"

  vars {
    db_port = "${aws_db_instance.mysql.port}"
    db_host = "${aws_db_instance.mysql.address}"
    db_user = "${var.username}"
    db_pass = "${var.password}"
    db_name = "${var.dbname}"
  }
}

resource "aws_db_instance" "mysql" {
  allocated_storage      = 20
  storage_type           = "gp2"
  engine                 = "mysql"
  engine_version         = "5.7"
  instance_class         = "db.t2.micro"
  name                   = "${var.dbname}"
  username               = "${var.username}"
  password               = "${var.password}"
  parameter_group_name   = "default.mysql5.7"
  vpc_security_group_ids =
["${aws_security_group.mysql.id}"]
  db_subnet_group_name   =
"${aws_db_subnet_group.mysql.name}"
  skip_final_snapshot    = true
}

resource "aws_instance" "ec2" {
  ami           = "${data.aws_ami.ubuntu.id}"
  instance_type = "t2.micro"

  depends_on = [
    "aws_db_instance.mysql",
  ]

  key_name                    =
"${aws_key_pair.keypair1.key_name}"
  vpc_security_group_ids      =
["${aws_security_group.web.id}"]
  subnet_id                   = "${aws_subnet.public1.id}"
  associate_public_ip_address = true
```

```
    user_data = "${file("userdata.sh")}"

    tags {
      Name = "EC2 Instance"
    }

    provisioner "file" {
      source      = "userdata.sh"
      destination = "/tmp/userdata.sh"

      connection {
        type        = "ssh"
        user        = "ubuntu"
        private_key = "${file("~/.ssh/id_rsa")}"
      }
    }

    provisioner "file" {
      content     = "${data.template_file.phpconfig.rendered}"
      destination = "/tmp/wp-config.php"

      connection {
        type        = "ssh"
        user        = "ubuntu"
        private_key = "${file("~/.ssh/id_rsa")}"
      }
    }

    timeouts {
      create = "20m"
    }
  }

  data "aws_ami" "ubuntu" {
    most_recent = true

    filter {
      name   = "name"
      values = ["ubuntu/images/hvm-ssd/ubuntu-trusty-14.04-
amd64-server-*"]
    }

    filter {
      name   = "virtualization-type"
      values = ["hvm"]
    }

    owners = ["099720109477"] # Canonical
  }

  module "alb" {
      name                    = "App-Load-Balancer"
```

```
        region                  = "us-east-1"
        environment             = "PROD"

        load_balancer_type          = "application"
        security_groups             =
["${module.vpc.security_group_id}",
"${module.vpc.default_security_group_id}"]
        subnets                     = ["${module.vpc.vpc-
privatesubnet-ids}"]
        vpc_id                      = "${module.vpc.vpc_id}"
        enable_deletion_protection  = false

        backend_protocol    = "HTTP"
        alb_protocols       = "HTTP"

        target_ids          = ["${module.ec2.instance_ids}"]
    }
```

Файл конфігурації для балансувальника навантаження наведено в лістингу 3.5, який містить в собі:

– ресурс визначення змінних для балансувальника навантаження ("aws_lb") які складаються з підмереж, захисту від видалення, IP адрес кінцевих точок, тайм-аутів на відповіді сервера, та логів;

– ресурс ("aws_lb_target_group") містить в собі змінні для перевірки стану кінцевих точок, порти підключення, ідентіфакатор для підмережі.

– ресурс ("aws_lb_listener") містить в собі змінні для протоколу передачі даних веб-додатку (frontend), політику безпеки для групп підключення.

Лістинг 3.5

```
    resource "aws_lb" "alb" {
        name_prefix        = "${var.name_prefix}-"
        name                = "${lower(var.name)}-alb-
${lower(var.environment)}"
        security_groups     = ["${var.security_groups}"]
        subnets             = ["${var.subnets}"]
        internal            = "${var.lb_internal}"

        enable_deletion_protection  =
```

```
"${var.enable_deletion_protection}"
        load_balancer_type          =
"${var.load_balancer_type}"
        idle_timeout                = "${var.idle_timeout}"
        ip_address_type             = "${var.ip_address_type}"

        access_logs         = ["${var.access_logs}"]

        timeouts {
            create  = "${var.timeouts_create}"
            update  = "${var.timeouts_update}"
            delete  = "${var.timeouts_delete}"
        }

        lifecycle {
            create_before_destroy = true
        }

        tags {
            Name            = "${lower(var.name)}-alb-
${lower(var.environment)}"
            Environment     = "${var.environment}"
            Orchestration   = "${var.orchestration}"
            Createdby       = "${var.createdby}"
        }
    }

    resource "aws_lb_target_group" "alb_target_group" {
        name                 = "${lower(var.name)}-alb-tg-
${lower(var.environment)}"
        port                 = "${var.backend_port}"
        protocol             = "${upper(var.backend_protocol)}"
        vpc_id               = "${var.vpc_id}"
        target_type          = "${var.target_type}"
        deregistration_delay = "${var.deregistration_delay}"

        tags {
            Name            = "${lower(var.name)}-alb-tg-
${lower(var.environment)}"
            Environment     = "${var.environment}"
            Orchestration   = "${var.orchestration}"
            Createdby       = "${var.createdby}"
        }

        health_check {
            interval            = "${var.health_check_interval}"
            path                = "${var.health_check_path}"
            port                = "${var.health_check_port}"
            healthy_threshold   =
"${var.health_check_healthy_threshold}"
            unhealthy_threshold =
"${var.health_check_unhealthy_threshold}"
            timeout             = "${var.health_check_timeout}"
```

```
            protocol            = "${var.backend_protocol}"
            matcher             = "${var.health_check_matcher}"
        }
        stickiness {
            type            = "lb_cookie"
            cookie_duration = "${var.cookie_duration}"
            enabled         = "${var.cookie_duration != 1 ? true
: false}"
        }
    }

    resource "aws_lb_listener" "frontend_http" {
        count               = "${trimspace(element(split(",",
var.alb_protocols), 1)) == "HTTP" ||
trimspace(element(split(",", var.alb_protocols), 2)) == "HTTP" ?
1 : 0}"

        load_balancer_arn   = "${aws_lb.alb.arn}"
        port                = "80"
        protocol            = "HTTP"

        "default_action" {
            target_group_arn    =
"${aws_lb_target_group.alb_target_group.arn}"
            type                = "forward"
        }

        depends_on =
["aws_lb.alb","aws_lb_target_group.alb_target_group"]
    }

    resource "aws_lb_listener" "frontend_https" {
        count               = "${trimspace(element(split(",",
var.alb_protocols), 1)) == "HTTPS" ||
trimspace(element(split(",", var.alb_protocols), 2)) == "HTTPS"
? 1 : 0}"

        load_balancer_arn   = "${aws_lb.alb.arn}"
        port                = 443
        protocol            = "HTTPS"
        certificate_arn     = "${var.certificate_arn}"
        ssl_policy          = "ELBSecurityPolicy-2016-08"
        default_action {
            target_group_arn    =
"${aws_lb_target_group.alb_target_group.arn}"
            type                = "forward"
        }

        depends_on =
["aws_lb.alb","aws_lb_target_group.alb_target_group"]
    }
```

```
     resource "aws_lb_target_group_attachment"
"alb_target_group_attachment"
```

Для забезпечення розгортання веб-додатку в AMI та виведення усіх необхідних для подальшого використання змінних (ідентіфікатор AMI, логін для SSH, пароль, зона доступності, адреса та порт бази даних, публічний URL для доступу на веб-додаток) використовується файл наведений в лістінгу 3.6.

Лістинг 3.6

```
     output "ami_id" {
       value = "${data.aws_ami.ubuntu.id}"
     }

     output "Login" {
       value = "ssh -i ${aws_key_pair.keypair1.key_name}
ubuntu@${aws_instance.ec2.public_ip}"
     }

     output "azs" {
       value = "${data.aws_availability_zones.azs.*.names}"
     }

     output "db_access_from_ec2" {
       value = "mysql -h ${aws_db_instance.mysql.address} -P
${aws_db_instance.mysql.port} -u ${var.username} -
p${var.password}"
     }

     output "access" {
       value = "http://${aws_instance.ec2.public_ip}/index.php"
     }
```

Далі визначено змінну для підключення до API Amazon Web Services, зазначено використовувану версію Terraform.

Лістинг 3.7

```
    provider "aws" {
      region = "${var.aws_reg}"
      version = "2.12.0"
    }
```

```
    provider "template" {
        version = "~> 2.1.2"
    }

    terraform {
    required_version = "v0.11.11"
}
```

Для опису груп безпеки та правил firewall для веб-додатку використовується файл наведений в лістингу 3.8, він містить в собі ресурси які визначають діапазони портів для підключення як до бази даних (MySQL 3306, HTTP 80, HTTPS 443), так і до веб-додатку, також вхідний трафік відкрито на 22 порти, що дозволяє підключатись напряму скрізь Secure Shell.

Лістинг 3.8

```
    resource aws_security_group "mysql" {
      name        = "${var.stack}-DBSG"
      description = "managed by terrafrom for db servers"
      vpc_id      = "${aws_vpc.vpc.id}"

      tags {
        Name = "${var.stack}-DBSG"
      }

      ingress {
        protocol        = "tcp"
        from_port       = 3306
        to_port         = 3306
        security_groups = ["${aws_security_group.web.id}"]
      }

      egress {
        protocol    = -1
        from_port   = 0
        to_port     = 0
        cidr_blocks = ["0.0.0.0/0"]
      }
    }

    resource aws_security_group "web" {
      name        = "${var.stack}-webSG"
      description = "This is for ${var.stack}s web servers
security group"
      vpc_id      = "${aws_vpc.vpc.id}"
```

```
  tags {
    Name = "${var.stack}-webSG"
  }

  ingress {
    protocol    = "tcp"
    from_port   = 22
    to_port     = 22
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    protocol    = "icmp"
    from_port   = -1
    to_port     = -1
    cidr_blocks = ["${aws_vpc.vpc.cidr_block}"]
  }

  ingress {
    protocol    = "tcp"
    from_port   = 80
    to_port     = 80
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    protocol    = -1
    from_port   = 0
    to_port     = 0
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

Для опису зон доступності, роутерів, шлюзів, NAT, та підмереж VPC використовується файл, наведений в лістингу 3.9, схему можна побачити на рис. 3.2. Основними ресурсами є:

- шлюз для трафіку з публічних підмереж (`"aws_internet_gateway"`);
- ассоціації таблиці маршрутизації для публічних та приватних підмереж (`"aws_route_table_association"`);
- визначення публічної та приватної підмережі VPC з діапазоном CIDR (`"aws_subnet"`);

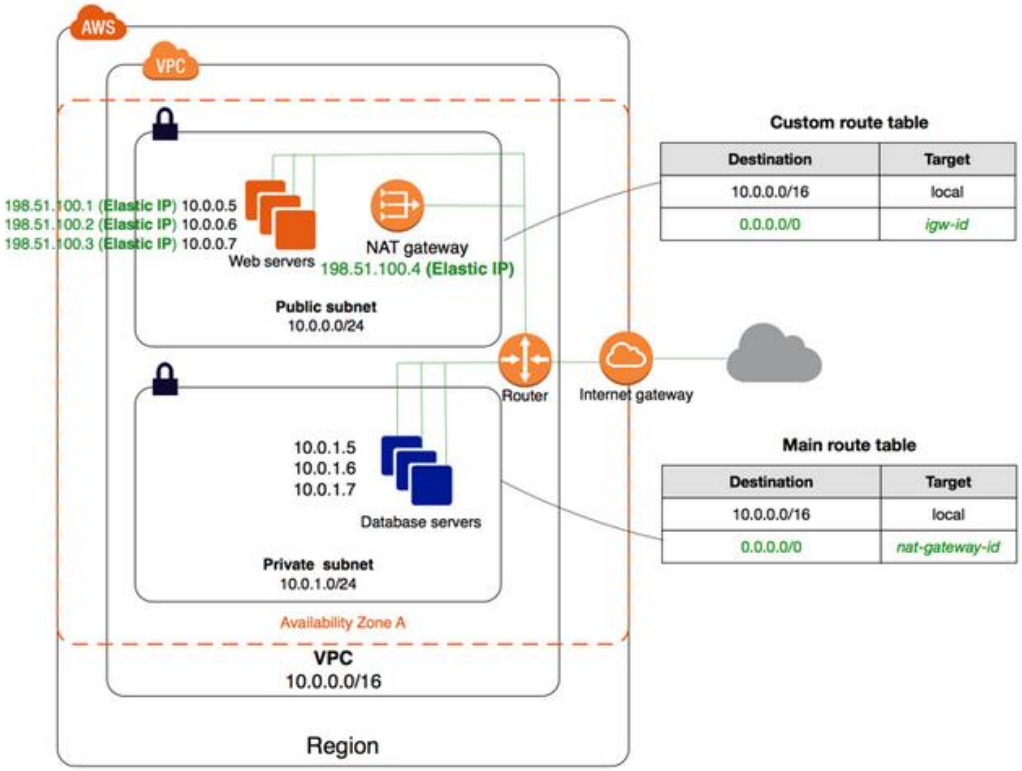

Рисунок 3.2 – Схема VPC

Лістинг 3.9

```
data "aws_availability_zones" "azs" {}

resource "aws_vpc" "vpc" {
  cidr_block = "10.0.0.0/16"

  tags {
    Name = "${var.stack}-vpc"
  }
}

resource "aws_internet_gateway" "igw" {
  vpc_id = "${aws_vpc.vpc.id}"

  tags {
    Name = "${var.stack}-igw"
  }
}

resource "aws_nat_gateway" "nat" {
  subnet_id     = "${aws_subnet.public1.id}"
  allocation_id = "${aws_eip.eip.id}"

  tags {
```

```
    Name = "${var.stack}-nat"
  }
}

resource "aws_eip" "eip" {

  vpc = true

  tags {
    Name = "${var.stack}-nat-ip"
  }
}

resource "aws_route_table" "private" {
  vpc_id = "${aws_vpc.vpc.id}"

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = "${aws_nat_gateway.nat.id}"
  }

  tags {
    Name = "${var.stack}-private"
  }
}

resource "aws_route_table" "public" {
  vpc_id = "${aws_vpc.vpc.id}"

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = "${aws_internet_gateway.igw.id}"
  }

  tags {
    Name = "${var.stack}-public"
  }
}

resource "aws_route_table_association" "private1" {
  route_table_id = "${aws_route_table.private.id}"

  subnet_id = "${aws_subnet.private1.id}"
}

resource "aws_route_table_association" "private2" {
  route_table_id = "${aws_route_table.private.id}"
  subnet_id      = "${aws_subnet.private2.id}"
}

resource "aws_route_table_association" "private3" {
  route_table_id = "${aws_route_table.private.id}"
  subnet_id      = "${aws_subnet.private3.id}"
```

```
}

resource "aws_route_table_association" "public1" {
  route_table_id = "${aws_route_table.public.id}"

  subnet_id = "${aws_subnet.public1.id}"
}

resource "aws_route_table_association" "public2" {
  route_table_id = "${aws_route_table.public.id}"
  subnet_id      = "${aws_subnet.public2.id}"
}

resource "aws_subnet" "public1" {
  vpc_id            = "${aws_vpc.vpc.id}"
  cidr_block        = "10.0.1.0/16"
  availability_zone =
"${data.aws_availability_zones.azs.names[0]}"

  tags {
    Name = "${var.stack}-public-1"
  }
}

resource "aws_subnet" "public2" {
  vpc_id     = "${aws_vpc.vpc.id}"
  cidr_block = "10.0.2.0/24"

  availability_zone =
"${data.aws_availability_zones.azs.names[1]}"

  tags {
    Name = "${var.stack}-public-2"
  }
}

resource "aws_subnet" "private1" {
  vpc_id     = "${aws_vpc.vpc.id}"
  cidr_block = "10.0.3.0/24"

  availability_zone =
"${data.aws_availability_zones.azs.names[0]}"

  tags {
    Name = "${var.stack}-private-1"
  }
}

resource "aws_subnet" "private2" {
  vpc_id     = "${aws_vpc.vpc.id}"
  cidr_block = "10.0.4.0/24"

  availability_zone =
```

```
"${data.aws_availability_zones.azs.names[1]}"

       tags {
         Name = "${var.stack}-private-2"
       }
     }

     resource "aws_subnet" "private3" {
       vpc_id            = "${aws_vpc.vpc.id}"
       cidr_block        = "10.0.5.0/24"
       availability_zone =
"${data.aws_availability_zones.azs.names[2]}"

       tags {
         Name = "${var.stack}-private-3"
       }
     }

     resource "aws_db_subnet_group" "mysql" {
       name       = "${var.stack}-subngroup"
       subnet_ids = ["${aws_subnet.private1.id}",
"${aws_subnet.private2.id}", "${aws_subnet.private3.id}"]

       tags {
         Name = "${var.stack}-subnetGroup"
       }
     }
```

Для зручного управління основними складовими ресурсів Terraform та міграції між хмарними провайдерами всі змінні, що використовуються в проекті, відокремлені файлом outputs.tf, що наведений в лістингу 3.10. Виправлення змінних дозволяє швидко змінювати регіон розгортання, ідентіфакаційні дані (включаючи SSH), типи інстансів, тип балансувальника навантаження, тривалість дії cookies.

Лістинг 3.10

```
     variable "name" {
       description = "Name to be used on all resources as prefix"
       default     = "TEST-ALB"
     }

     variable "region" {
       description = "The region where to deploy this code (e.g.
us-east-1)."
       default     = "us-east-1"
```

```
}

variable "environment" {
    description = "Environment for service"
    default     = "STAGE"
}

variable "orchestration" {
    description = "Type of orchestration"
    default     = "Terraform"
}

variable "createdby" {
    description = "Created by"
    default     = "Anatolii Bohatyrenko"
}

variable stack {
  description = "this is name for tags"
  default     = "terraform"
}

variable username {
  description = "DB username"
}

variable password {
  description = "DB password"
}

variable dbname {
  description = "db name"
}

variable ssh_key {
  default     = "~/.ssh/id_rsa.pub"
  description = "Default pub key"
}

variable "security_groups" {
    description = "A list of security group IDs to assign to
the ELB. Only valid if creating an ELB within a VPC"
    type        = "list"
}

variable "subnets" {
    description = "A list of subnet IDs to attach to the
ELB"
    type        = "list"
    default     = []
}

variable "lb_internal" {
```

```
        description = "If true, ALB will be an internal ALB"
        default     = false
    }

    variable "name_prefix" {
        description = "Creates a unique name beginning with the
specified prefix. Conflicts with name"
        default     = "alb"
    }

    variable "enable_deletion_protection" {
        description = "If true, deletion of the load balancer
will be disabled via the AWS API. This will prevent Terraform
from deleting the load balancer. Defaults to false."
        default     = false
    }

    # Access logs
    variable "access_logs" {
        description = "An access logs block. Uploads access logs
to S3 bucket. Can be used just for network LB!"
        type        = "list"
        default     = []
    }

    variable "load_balancer_type" {
        description = "The type of load balancer to create.
Possible values are application or network. The default value is
application."
        default     = "application"
    }

    variable "idle_timeout" {
        description = "The time in seconds that the connection
is allowed to be idle. Default: 60."
        default     = "60"
    }

    variable "ip_address_type" {
        description = "The type of IP addresses used by the
subnets for your load balancer. The possible values are ipv4 and
dualstack"
        default     = "ipv4"
    }

    variable "vpc_id" {
        description = "Set VPC ID for ?LB"
    }

    variable "alb_protocols" {
        description = "A comma delimited list of the protocols
the ALB accepts. e.g.: HTTPS"
        default     = "HTTP,HTTPS"
```

```
    }

    variable "backend_port" {
        description = "The port the service on the EC2 instances
listen on."
        default     = 80
    }

    variable "backend_protocol" {
        description = "The protocol the backend service speaks.
Options: HTTP, HTTPS, TCP, SSL (secure tcp)."
        default     = "HTTP"
    }

    variable "target_ids" {
        description = "The ID of the target. This is the
Instance ID for an instance, or the container ID for an ECS
container. If the target type is ip, specify an IP address."
        type        = "list"
        #default     = []
    }

    variable "certificate_arn" {
        description = "The ARN of the SSL Certificate. e.g.
'arn:aws:iam::XXXXXXXXXXX:server-certificate/ProdServerCert'"
        default     = ""
    }

    }

    variable "cookie_duration" {
        description = "If load balancer connection stickiness is
desired, set this to the duration in seconds that cookie should
be valid (e.g. 300). Otherwise, if no stickiness is desired,
leave the default."
        default     = 1
    }
```

ВИСНОВКИ

У роботі представлений метод керування хмарною інфраструктурою за допомогою парадигми Infrastructure as Code та зокрема Terraform. Як приклад для проектування була вибрана відмовостійка інфраструктура з підтримкою високого навантаження на базі Amazon Web Services.

Не має значення де саме розгорнута інфраструктура, Terraform надає можливість мігрувати до будь-якого хмарного провайдера, або дата-центра. Наведений приклад є лише одним з багатьох методів побудування інфраструктури для веб-додатків. Головною метою та задачею Terraform є створення інфраструктури, після цього не має значення що буде виконуватися на розгорнутих потужностях. На мою думку Terraform справляється зі своїм головним завданням на відмінно, не зважаючи на те, що на даний момент навіть не анонсована версія 1.0, а поточна версія на момент написання роботи – 0.11.11.

Terraform продовжує активно розроблятись та розширюватись, про це свідчить поява різноманітних open-source інструментів пов’язаних як із зпрощенням роботи з Terraform так і з ефективним викростиуванням на практиці. Завдяки зрозумілому синтаксису, початкова перешкода є низькою. Розробники можуть швидко звикнути до декларування інфраструктури в коді. В результаті Terraform – це інструмент, що демонструє культуру DevOps завдяки тому, що вона значно спрощує як експерименти, так і продуктивне використання хмарних рішень.

Подальші напрями досліджень можуть бути пов'язані з автоматизацією тестування побудованих архітектур. Terraform не зважаючи на усі переваги має деякі недоліки, наприклад – неможливість відслідковування стану інфраструктури при совмістному використуванні Terraform не застосовуючи Continious Integration, використування із кластерами dedicated серверів, відустність умовних операторів, вирішення залежностей між модулями.

Однак я впевнений в тому, що впродовж декількох років усі недоліки будуть усунуті.

Дана робота демонструє деякі найкращі практики в побудуванні хмарної архітектури на базі Amazon Web Services для високонавантажених веб-застосунків за допомогою Terraform. Основна увага була зосереджена на тому, як використовувати хмарну та одноразову інфраструктуру, щоб швидко відбудовуватися, докладаючи невеликих зусиль у процесі надання, налаштування, розгортання та знищення хмарної інфраструктури.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Andreas W. Amazon Web Services in Action / Andreas W., Michael W. – Manning Publications Co., 2016. – P. 53–56.
2. Marcus Y. Implementing Cloud Design Patterns for AWS / Packt Publishing Ltd. 2015 – 212 p.
3. Yevgeniy B. Terraform: Up & Running, 2nd Edition / O`Reilly, 2019. – 188 p.
4. Kirill S. Getting Started with Terraform / Packt Publishsing Ltd. 2017. – 160 p.
5. Джэннифер Д. Философия DevOps. Искусство управления IT / Дженнифер Д., Кэтрин Д. // Питер.– 2017– С. 50-57.
6. Джордж С. Проект «Феникс». Роман о том, как DevOps меняет бизнес к лучшему / Джордж С., Кевин Б., Джин К. - Эксмо, 2015 – 147 с.