БАКАЛАВРСЬКА РОБОТА

БР. ІП - 12.00.00.000 ПЗ

Група ІП-21-1

Додяк Владислав

2025

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Додяк Владислав Валерійович

(прізвище, ім'я, по батькові)

УДК 004.942
(індекс)

БАКАЛАВРСЬКА РОБОТА

Побудова масштабованих хмарних додатків
(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня _____Додяк Владислав Валерійович_____
(підпис, ініціали та прізвище здобувача)

Науковий керівник _____Гобир Ліда Мирославівна асистент_____
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

_доц._____Бандура В.В.__
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доцент. В.В. Бандура
“ ____ ” _____2024 р.

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ БАКАЛАВРА СТУДЕНТОВІ

Додяк Владислав Валерійович

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) "Побудова масштабованих хмарних додатків"

керівник проекту (роботи) к.т.н., доцент Гобир Лідія Мирославівна

затверджені наказом вищого навчального закладу від **“** 28 ” квітня 2025 р. № 264/7

2. Строк подання студентом проекту (роботи) 10 червня 2025 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1 Основи хмарних технологій та сервісів

2.Розгортання та масштабування хмарної інфраструктури

3.Оптимізація і розповсюдження

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Інформаційна панель Linux (рис.2.1, ст.20)

2. Екран керування резервним копіюванням Linode (рис. 2.3, ст.26)

3. Дворівнева архітектура з глобальним кешуванням (рис 2.5, ст 34)

4. Вивід ApacheBench для кешованої конфігурації (рис. 3.2, ст 38)

5. Створення розповсюдження CloudFront (рис. 3.7, ст 54)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

2. Дата видачі завдання **2025 р.**

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Визначення та обґрунтування теми роботи	15.02.2025	виконано
2	Огляд існуючих концепцій, рішень та сервісів в даній області	25.02.2025	виконано
3	Побудова моделі або алгоритму власного рішення	15.03.2025	виконано
4	Документування реалізації власного оригінального рішення вибраними засобами	25.04.2025	виконано
5	Оформлення пояснювальної записки кваліфікаційної роботи	10.06.2025	виконано

Студент _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Бакалаврська робота містить 60 сторінок, 15 рисунків, список використаних джерел із 20 найменування,

Метою роботи дослідити принципи побудови масштабованих хмарних додатків та розробити практичні підходи до їх створення, оптимізації і розгортання з використанням сучасних хмарних технологій.

Об'єкт дослідження: Хмарні обчислення та сервіси, що забезпечують розробку та масштабування хмарних додатків.

Предмет дослідження: Методи, інструменти та технології побудови масштабованих хмарних додатків, зокрема засоби розгортання, управління інфраструктурою, оптимізації продуктивності й розповсюдження контенту.

Результати дослідження: Розроблено практичні рекомендації щодо побудови масштабованої хмарної архітектури з урахуванням продуктивності, безпеки та витрат.

В першому розглядаються основи хмарних технологій і сервісів, включаючи визначення хмарних технологій, типи сервісів (SaaS, PaaS, IaaS), принципи роботи Docker і особливості технології IaaS

Другий розділ присвячений розгортанню та масштабуванню хмарної інфраструктури, охоплюючи створення віртуальних серверів, налаштування служб баз даних, формування хмарних кластерів і використання кешування для підвищення масштабованостіВ третьому розділі

Третій розділ зосереджується на оптимізації та розповсюдженні хмарних рішень, аналізуючи методи кешування в програмах, реплікацію баз даних, використання мереж доставки контенту (CDN) і розгортання додатків на Google Cloud Platform.

Висновок:.узагальнено ключові результати та окреслено перспективи розвитку хмарних технологій для побудови масштабованих додатків.

КЛЮЧОВІ СЛОВА: ХМАРНІ ТЕХНОЛОГІЇ, МАСШТАБОВАНІ ДОДАТКИ, КОНТЕЙНЕРИЗАЦІЯ, DOCKER, GOOGLE CLOUD PLATFORM, IAAS, CDN, КЕШУВАННЯ, РЕПЛІКАЦІЯ БАЗ ДАНИХ, ОПТИМІЗАЦІЯ.

ANNOTATION

The bachelor's thesis contains 60 pages, 15 figures, a list of used sources with 20 names,

The purpose of the work is to investigate the principles of building scalable cloud applications and develop practical approaches to their creation, optimization and deployment using modern cloud technologies.

Object of research: Cloud computing and services that provide development and scaling of cloud applications.

Subject of research: Methods, tools and technologies for building scalable cloud applications, in particular means of deployment, infrastructure management, performance optimization and content distribution.

Research results: Practical recommendations have been developed for building a scalable cloud architecture taking into account performance, security and costs.

The first section covers the basics of cloud technologies and services, including the definition of cloud technologies, types of services (SaaS, PaaS, IaaS), Docker principles and features of IaaS technology

The second section is devoted to the deployment and scaling of cloud infrastructure, covering the creation of virtual servers, configuring database services, forming cloud clusters and using caching to increase scalability. In the third section

The third section focuses on the optimization and distribution of cloud solutions, analyzing caching methods in applications, database replication, the use of content delivery networks (CDN) and deploying applications on Google Cloud Platform.

KEYWORDS: CLOUD TECHNOLOGIES, SCALABLE APPLICATIONS, CONTAINERIZATION, DOCKER, GOOGLE CLOUD PLATFORM, IAAS, CDN, CACHING, DATABASE REPLICATION, PERFORMANCE OPTIMIZATION..

ЗМІСТ

ВСТУП

Сучасна цифрова економіка вимагає від компаній швидкої адаптації до зростаючих потреб користувачів, що зумовлює необхідність створення високопродуктивних, надійних і масштабованих хмарних додатків. Хмарні технології стали основою для розробки програмного забезпечення, яке здатне обробляти великі обсяги даних, забезпечувати безперебійну роботу та динамічно адаптуватися до змін у навантаженні. Побудова масштабованих хмарних додатків є комплексним завданням, яке поєднує інженерні принципи, архітектурні підходи та сучасні інструменти, такі як контейнеризація, інфраструктура як сервіс (IaaS) і мережі доставки контенту (CDN). Ці технології дозволяють створювати системи, які не лише відповідають поточним вимогам, але й можуть масштабуватися для підтримки майбутнього зростання.

Актуальність теми зумовлена стрімким зростанням попиту на хмарні сервіси, які забезпечують гнучкість, економію ресурсів і високу доступність. Згідно з прогнозами аналітичних агентств, таких як Gartner, ринок хмарних технологій продовжуватиме зростати, а компанії, які ефективно використовують масштабовані хмарні рішення, отримають конкурентні переваги. Однак створення таких додатків пов’язане з низкою викликів, включаючи вибір відповідних технологій, оптимізацію продуктивності, забезпечення безпеки та ефективне управління ресурсами. Особливого значення набувають інструменти, такі як Docker для контейнеризації, Google Cloud Platform для розгортання інфраструктури, а також стратегії кешування та реплікації баз даних для підвищення масштабованості.

Метою цього дослідження є аналіз ключових аспектів побудови масштабованих хмарних додатків, включаючи теоретичні основи хмарних технологій, практичні підходи до розгортання інфраструктури та методи оптимізації продуктивності. Робота спрямована на систематизацію знань про сучасні інструменти та технології, а також на надання практичних рекомендацій для розробників і архітекторів хмарних систем. Дослідження охоплює як базові

концепції, так і передові методи, що дозволяють створювати ефективні та масштабовані додатки.

Цей документ має на меті не лише надати теоретичну базу, але й запропонувати практичні рекомендації, які можуть бути використані розробниками, інженерами та дослідниками для створення сучасних хмарних рішень, здатних відповідати викликам динамічного цифрового середовища.

Завданнями дослідження є проаналізувати поняття хмарних технологій, їх типи та основні сервіси (IaaS, PaaS, SaaS). Ознайомитися з інструментами контейнеризації, зокрема Docker, та їхньою роллю у побудові хмарних додатків. Дослідити архітектуру та етапи створення віртуальних серверів і налаштування базової хмарної інфраструктури. Розглянути методи масштабування хмарних рішень, зокрема через кешування, розподіл навантаження та реплікацію даних. Оцінити можливості використання CDN (мереж доправлення контенту) для прискорення роботи додатків. Реалізувати оптимізаційні рішення для хмарного застосунку: кешування, розподіл запитів, масштабування бази даних. Проаналізувати переваги та особливості використання хмарних платформ, зокрема Google Cloud Platform. Зробити висновки щодо ефективності використаних технологій для побудови масштабованих хмарних додатків.

Результати дослідження: реалізовано реплікацію бази даних, що забезпечує високу доступність і відмовостійкість хмарної системи. Впроваджено CDN (мережу доставки контенту), вивчено можливості Google Cloud Platform як прикладу масштабованої хмарної екосистеми, що підтримує гнучке розгортання, моніторинг і управління додатками. Розроблено практичні рекомендації щодо побудови масштабованої хмарної архітектури з урахуванням продуктивності, безпеки та витрат.

Об'єкт дослідження: Хмарні обчислення та сервіси, що забезпечують розробку та масштабування хмарних додатків.

Предметом дослідження: Методи, інструменти та технології побудови масштабованих хмарних додатків, зокрема засоби розгортання, управління

інфраструктурою, оптимізації продуктивності й розповсюдження контенту

Методи дослідження: Аналіз літературних джерел Метод систематизації та класифікації, емпіричне моделювання, експериментальний метод, порівняльний, візуалізація результатів.

Бакалаврська робота містить 60 сторінок, 15 рисунків, список використаних джерел із 29 найменуванням.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

GCP - Google Cloud Platform

AWS - Amazon Web Services

CDN – мережа доставки контенту

РОЗДІЛ 1 ОСНОВИ ХМАРНИХ ТЕХНОЛОГІЙ ТА СЕРВІСІВ

1.1 Визначення хмарних технологій і типи сервісів

Існує невелика плутанина щодо того, що таке «перебувати в хмарі». Деякі люди навіть помилково вважають, що просто перебування в Інтернеті означає використання хмарних технологій. Суттєва відмінність між хмарною технологією та іншими типами Інтернет-хостингу полягає в тому, що хмарна служба пропонує принаймні можливість швидко масштабувати вашу програму. Історично розробники розгортали свої веб-додатки на стаціонарних серверах, які вони купували або орендували на певному об'єкті. Можна було отримати більше обладнання, але це завжди вимагало чимало часу та зусиль. Часто розробнику доводиться вкладати великі капітальні витрати на сервери, купувати їх, налаштовувати та розгортати. Цей процес може тривати тижні або навіть місяці. Навіть якщо купувати безпосередньо в хостинг-компанії, процес складання цінової пропозиції та налагодження роботи може тривати більше тижня.

Обіцянка хмари полягає в тому, що замість того, щоб проходити через процес фізичного налаштування для нових серверів, додаткову потужність можна надати або миттєво, або принаймні протягом хвилин або годин, а не днів, тижнів або місяців. З деякими рішеннями вам навіть не доведеться турбуватися про машини - хмарне рішення автоматично масштабує вашу програму на стільки машин, скільки вам потрібно. З іншими достатньо лише кілька клацань, щоб запитати, створити образ і завантажити нову машину, яка є копією існуючої машини, а потім додати її до веб-програми. У будь-якому випадку, основною ідеєю хмари є миттєва, автоматизована масштабованість і гнучкість. © Джонатан Бартлетт, 2019 7 Дж. Бартлетт, Створення масштабованих веб-додатків PHP за допомогою хмари, https://doi.org/10.1007/978-1-4842-5212-3_2 Багато хмарних провайдерів навіть надають доступ до кількох центрів обробки даних. Вам потрібен набір серверів в Америці та набір серверів у Європі? немає проблем Лише кількома клацаннями миші ви зможете це зробити.

Хмарні обчислення часто плутають, оскільки існує кілька різних типів

хмарних обчислень, кожен із яких має свої переваги та недоліки. Різниця між типами хмарних сервісів здебільшого залежить від рівня абстракції, який пропонується. Найпростіший тип хмарних служб, на яких буде зосереджено увагу в цій книзі, називається інфраструктура як послуга, скорочено IaaS. IaaS означає, що ви можете придбати та розгорнути частини інфраструктури (сервери, балансувальники навантаження, брандмауери тощо) одним натисканням кнопки. Це робиться за допомогою технології віртуалізації серверів. Віртуалізація серверів дозволяє компанії, яка надає послуги IaaS, розгорнути один дуже великий сервер і розділити його на кілька менших серверів. На кожному сервері працює програма гіпервізора , яка дозволяє компанії швидко й автоматично розділити сервер.

Компанія може взяти комп'ютер із 16 процесорами та 64 гігабайтами оперативної пам'яті та розділити його на 4 віртуальні машини, кожна з яких має 4 процесори та 16 гігабайтів оперативної пам'яті. Це має три переваги. Перший – космос. Купуючи найбільшу машину, компанія IaaS забезпечує собі найбільшу обчислювальну потужність на одиницю стійки, яка є цінним товаром у серверних кімнатах. Тому, купивши одну велику машину та розділивши її на чотири менші машини, вони використали лише четверту частину простору, який вони могли б використати за інших умов. По-друге, це вартість ядра ЦП. Упаковуючи стільки ядер ЦП і стільки пам'яті в один сервер, їхня вартість за ядро ЦП і вартість за гігабайт пам'яті знижуються. Тому, купуючи більші комп'ютери, вони можуть забезпечувати нижчу вартість ядра ЦП для користувача, який купує менші віртуальні сервери. Третя перевага, однак, найважливіша - керованість.

Для підтримки віртуалізації невелика частина кожної машини призначена для гіпервізора - невеликої операційної системи, яка керує іншими віртуальними машинами на сервері . Оскільки машини є віртуальними машинами, а не реальними апаратними пристроями, ними неймовірно легко керувати. Оператор може надіслати команду до гіпервізора, і він зможе миттєво налаштувати нову віртуальну машину, клонувати новий завантажувальний диск і запустити новий віртуальний сервер за лічені хвилини. Історично склалося так, що якщо я хотів

би новий сервер, мені довелося б купити сервер, потім, перебуваючи фізично біля консолі, потрібно було б встановити відповідну операційну систему та системне програмне забезпечення, і, нарешті, перенести сервер у стійку, де я підключаю його та вмикаю. Мені потрібно переконатися, що мережу підключено та налаштовано. Можливо, мені доведеться налаштувати BIOS, щоб дозволити роботу без підключення клавіатури. Якщо виникла проблема з машиною, мені потрібно зробити резервну копію машини, знайти нову фізичну машину, скопіювати резервну копію на нову машину, встановити нову машину та фізично замінити її на стару машину. З віртуалізованими машинами я можу дозволити гіпервізору подбати про все це. Все, що мені потрібно, це достатня кількість додаткових серверів із гіпервізорами, щоб, коли мені знадобиться нова машина, я міг просто сказати гіпервізору створити для мене нову віртуальну машину та де знаходиться копія диска, яку я хочу використовувати.

Що ще краще, так це те, що з більшістю платформ IaaS вам навіть не потрібно турбуватися про гіпервізори та ємність. Постачальник IaaS робить усе це за вас. Постачальник IaaS надає інтерфейс «вкажи та клацни», який дозволяє завантажувати віртуальні сервери, просто ввійшовши у веб-консоль керування. Ви говорите йому, яку велику машину ви бажаєте (тобто кількість ядер процесора та розмір пам’яті), і він виділить вам сервер. Ви говорите йому, що ви хочете від нього (або базову операційну систему, або клон існуючої машини), і він скопіює це на свій завантажувальний диск і завантажить його для вас. Вуаля! У вас є новий сервер, готовий до роботи. Крім того, у більшості служб IaaS, якщо виникають проблеми з фізичним обладнанням, служба подбає про це за вас. Якщо буде виявлено серйозну проблему з апаратним забезпеченням, вони просто вимкнуть ваш сервер, перенесуть його на новий, запустять резервне копіювання та надішлють вам електронний лист із повідомленням про те, що сталося. Якщо це менш серйозна проблема, деякі постачальники надішлють вам сповіщення з проханням натиснути кнопку, щоб виконати міграцію у найбільш зручний для вас час. Ще кращою є модель ціноутворення. Більшість постачальників IaaS мають опцію погодинної ціни.

Інший варіант масштабування додатків відомий як платформа як послуга, що скорочено називається PaaS. З PaaS замість того, щоб надавати вам голі машини та дозволяти вам запускати все, що забажаєте, PaaS визначає платформу, на якій ви можете запускати свою програму. Постачальник PaaS керує всією платформою, і вам залишається лише турбуватися про свою програму. Наприклад, Heroku є популярним постачальником PaaS. Heroku займається всім адмініструванням і обслуговуванням сервера. Ви навіть не можете увійти на їхні машини! Ви просто надсилаєте їм код, і вони розгортають його на своїх машинах замість вас. Інші поширені постачальники PaaS включають Google App Engine, Windows Azure, Amazon Elastic Beanstalk і OpenShift. За допомогою PaaS ви обираєте, скільки «робочих процесів» використовувати, і система розподілить цю роботу між необхідними машинами (воркер - це лише активний процес, і кожен постачальник PaaS має власний термін для нього). Таким чином, постачальник PaaS піклується про платформу (апаратне забезпечення, операційну систему, встановлені програми), а ви лише керуєте кодом програми. Теоретично це звучить чудово - вам більше не потрібно взагалі керувати серверами! Все це робиться для вас автоматично за лаштунками. Однак реальність така, що платформи PaaS не такі прозорі, як здається, і постачальники PaaS, як правило, мають досить високу ціну.

1.2 Що таке Докер

Нова технологія в галузі, яка викликала багато шуму, - Docker. Програми Docker є чимось середнім між IaaS і PaaS. Програма Docker - це образ, який, по суті, містить повну інсталяцію всіх програм, необхідних для запуску програми. Ними можна керувати, розгортати та масштабувати їх подібно до PaaS, де ви можете просто вказати службі, скільки ви хочете запустити, і вона подбає про розгортання вашого образу в потрібних місцях, але у вас є набагато більший ступінь контролю, подібно до IaaS. Цікавою частиною Docker є можливість «зв’язувати» різні сервіси разом. По суті, ви надаєте імена різним типам служб, які надає або потребує контейнер Docker, а потім можете використовувати ці

імена, щоб вказувати своїм службам, як знаходити одна одну. Однак це здебільшого лише проблема під час початкового налаштування вашої програми, після чого вже не має значення, чи ви пов’язали компоненти програми вручну чи

за допомогою модного інструменту. Docker сам по собі є технологією, а не постачальником. Існує багато постачальників, які дозволяють розгортати програми Docker, у тому числі Docker Inc., компанія, що відповідає за технологію Docker.

1.3 Технодлгія Iaas

Ця робота зосереджена на хмарних моделях IaaS з кількох причин. По-перше, IaaS є дуже гнучким. Незалежно від того, який тип робочого навантаження ви хочете запустити, незалежно від того, яку платформу ви хочете запустити, IaaS просто надає вам голі сервери. Що ви з ними робите, залежить від вас. Це також означає, що ви не прив’язані до систем конкретного постачальника. По-друге, IaaS працює досить передбачувано. Хоча є відмінності, більшість постачальників IaaS працюють досить схожими. Ви обираєте розмір коробки, говорите, що хочете, і натискаєте кнопку. Якість, вартість і гнучкість цих послуг дуже різноманітні, але всі вони досить схожі. З PaaS ваш вибір значно обмежені. Перш за все, ви обмежені тим, які платформи пропонує ваш постачальник.

Якщо ви програмуєте на Ruby on Rails, ви обмежені лише Ruby on Rails PaaS. Хоча більшість постачальників PaaS відкрили свої системи для досить великої кількості різних серверів додатків, все ще існують обмеження. По-друге, ви повинні написати свій код відповідно до того, як вони налаштували свою платформу. Зазвичай це включало невелике або повне відсутність локального сховища файлів, певні конкретні типи баз даних для підключення, лише певні дозволені параметри платформи чи розширення та невелику або повну відсутність гнучкості щодо конфігурації сервера. Іноді це добре, але іноді це надто обмежує. Крім того, PaaS зазвичай занадто непрозорий. Доступ до

журналів сервера часто ускладнений, а проблеми з налагодженням, пов'язані з сервером, майже неможливі. Іноді постачальники PaaS мають інструменти, які можуть допомогти, але вони не що інше, як можливість налагодити безпосередньо на машині, яка має проблему. Ця непрозорість іноді може призвести до жахливих результатів.

Наприклад, якщо ви купуєте систему баз даних PaaS і чомусь ваша база даних пошкоджується, ваші можливості будуть надзвичайно обмеженими. В основному вам потрібно подзвонити в компанію і благати про допомогу. Деякі компанії дуже чуйно реагують на це, але це мене дуже нервує. Також майте на увазі, що для того, щоб система PaaS працювала, вони повинні постійно оновлювати свої системи. Насправді це те, за що ви їм платите. Однак немає жодної гарантії, що оновлення, яке вони зроблять завтра, випадково не призведе до сміття вашої програми. Можливо, оновлення було необхідним, але це не те рішення, яке ви маєте приймати. Подібним чином використання постачальника PaaS означає, що ви повинні оновлюватись за їхнім графіком. Якщо вони вирішать застаріти частину своєї інфраструктури, вам доведеться переписати свій код, щоб вирішити цю проблему. Якщо вони вирішать, що версія програмного забезпечення, яке ви використовуєте, застаріла, ви повинні переписати свій код, щоб використовувати нову версію. Коротше кажучи, PaaS означає, що ви втрачаєте контроль над своєю технологією, тоді як IaaS означає, що ви маєте повний контроль.

Хоча певні труднощі усуваються за допомогою PaaS, вони зазвичай компенсуються проблемами, які вони створюють. Нарешті, PaaS зазвичай дуже дорогий. Наприклад, чотири ядра ЦП на Heroku (звичайний постачальник Ruby on Rails PaaS) коштують 100 доларів на місяць. Вартість 4-ядерної машини на Linode становить лише 40 доларів на місяць, а машини Linode працюють швидше. Я виявив, що більшість постачальників PaaS стягують приблизно втричі більше, ніж постачальники go od IaaS за еквівалентну продуктивність. Постачальники PaaS роблять для вас більше, але це варто, лише якщо ви не маєте жодного досвіду керування системами у своїй лінійці розробників, і вам часто

доведеться сплачувати ці витрати, щоб не відставати від платформи PaaS. Якщо у вашій організації є ноу-хау з налаштування та обслуговування серверів (і ця робота є хорошим початком для вивчення того, як це зробити), IaaS наразі є найпростішим, найгнучкішим, найшвидшим і найменш дорогим способом скористатися перевагами хмарних технологій, і він не прив'язує вас до певного постачальника. Якщо ви вирішите, що справді хочете використовувати систему, схожу на PaaS, просто знайте, що існує багато систем PaaS з відкритим кодом, які можна запускати на ваших серверах IaaS.

Є багато міркувань, які стосуються вибору служби або постачальника IaaS. Найважливішим фактором є надійність послуги. Не допоможе мати чудову веб-програму, якщо служба не працює. Це не принесе прибутку вашій компанії в довгостроковій перспективі, якщо ви не можете отримати доступ до своїх даних. Таким чином, надійність послуги має суттєво впливати на ваші рішення. Це також поширюється на їх здатність вирішувати квитки. Кожна служба колись матиме проблеми. Якщо компанія не може вчасно відповісти на запити або вирішити їх, то ви не повинні мати на них виробничі системи. Наступним основним фактором є співвідношення ціна/якість.

Багато перших компаній, що розробляли хмарну інфраструктуру, надавали надзвичайно високу цінність гнучкості хмарних обчислень, у результаті чого майже кожне хмарне рішення коштувало жахливо високої вартості. Amazon Web Services (AWS) має службу хмарних обчислень, відому як EC2, яка є чудовим прикладом цього. Як згадувалося раніше, 4-ядерна машина Linode з 8 ГБ оперативної пам'яті коштує 40 доларів на місяць. Подібна специфікована машина на EC2 (машина c5.xlarge) коштує 122 долари на місяць. Історично склалося так, що EC2, як правило, був набагато повільнішим, ніж Linode, навіть із тими самими характеристиками. На початку хмарних обчислень EC2 був одним із небагатьох великих гравців у цій галузі, і щоб отримати гнучкість, вони змусили вас заплатити високу ціну. Особисто я б вважав за краще вийти за межі хмари та зробити традиційне налаштування хостингу, де я орендую фізичні сервери.

Для мене гнучкість не варта тієї ціни, яку вимагає AWS. На іншій чаші ваг є хмарні сервіси, ціни на які неймовірно високі. Тобто за ціною ви можете зрозуміти, що вони не можуть забезпечити надійне довгострокове обслуговування за такою ціною. Або компанія припинить свою діяльність, їй доведеться підвищити ціни, або послуга в кінцевому підсумку перевищить передплату та погіршиться до стану непридатності. Чудовим прикладом є CloudAtCost (www.cloudatcost.com). З CloudAtCost ви сплачуєте одноразову комісію та отримуєте можливість зберігати сервер назавжди. Наприклад, за 70 доларів ви отримуєте 2 ядра ЦП і 1 ГБ оперативної пам'яті. Але це не щомісячні витрати. Це одноразова вартість! Як я вже сказав, це неймовірно хороша ціна. Вони стягують щорічну ціну за обслуговування облікового запису в розмірі 9 доларів США, але це однакова вартість, незалежно від того, скільки у вас серверів (ймовірно, це було запроваджено, щоб вони могли вимкнути сервери, які більше не обслуговувалися). Є речі, для яких така послуга може бути корисною. Якщо вам, наприклад, потрібен сервер для розробки, не має значення, чи час від часу вимикається мережа чи що технічні спеціалісти не швидко відповідають. І якщо одного разу вони зачинять свої двері, ви не будете на вулиці. Але я б точно не поставив свій бізнес на таку послугу. Якщо ви хочете побачити інші послуги за неймовірно низькими цінами, відвідайте www.lowendbox.com .

Останнім фактором є гнучкість. Одна справа - мати можливість навести курсор і клацнути, щоб налаштувати нову машину, але що, якщо ви не можете зберігати образи дисків і вам доводиться перебудовувати машину кожного разу, коли ви її запускаєте? Це буде копіткий процес, і ви втратите одну з головних переваг хмарних обчислень. У цій сфері AWS блищить. AWS складається не лише з служб хмарних обчислень (EC2), AWS також забезпечує конфігурацію, контроль і автоматизацію майже кожного аспекту хмарних інфраструктур. AWS надає масштабовані рішення для зберігання, послуги перекодування відео, масштабовані бази даних, служби черги повідомлень і служби пошуку. Це надає вашому кластеру зовнішню службу моніторингу, яка автоматично створює нові

сервери для обробки навантаження, коли воно зростає, і видаляє сервери з вашого кластера та вимикає їх, коли навантаження на мережу зменшується. Іншими словами, додаткові послуги, які надає AWS, майже безмежні. Однак, зрештою, гнучкість AWS переважує жахливе співвідношення ціна/продуктивність EC2. Однак, на щастя, кращі служби AWS (такі як S3 і CloudFront) можна використовувати окремо, навіть якщо ви використовуєте іншого постачальника як основного постачальника IaaS. Ми обговоримо, як це зробити в наступних розділах. Якщо ви ще не здогадалися, я віддаю перевагу постачальнику хмарних технологій Linode (www.linode.com). Співвідношення ціна/продуктивність є неперевершеним. Причина цього тричі. По-перше, обладнання новіше. По-друге, вони використовують лише SSD-накопичувачі (тобто твердотільні диски без обертових дисків), які, як правило, на порядок швидші за звичайні диски. Це, мабуть, один із найважливіших аспектів продуктивності Linode. По-третє, вони запровадили засоби контролю, щоб запобігти «галасливим сусідам».

Шумний сусід - це віртуальна машина, яка знаходиться на тому ж фізичному сервері, що й ваша машина, але використовує всі ресурси вводу-виводу комп’ютера. На платформах IaaS ви не можете вибирати (або навіть знати), хто ще користується тим же фізичним обладнанням. Тому дуже важливо використовувати службу, яка запобігає виснаженню ресурсів. Linode реалізував низку елементів керування, щоб запобігти надмірному використанню ресурсів окремою віртуальною машиною. Це добре не тільки для продуктивності вашого власного віртуального сервера, але й тому, що це означає, що вам не доведеться турбуватися про те, що ви будете поганим сусідом для когось іншого! Хоча Linode не має такої гнучкості, як AWS, цю гнучкість навряд чи можна упустити. Більшість важливих речей, які ви хочете зробити з AWS, надзвичайно прості з Linode, а додаткові функції AWS роблять криву навчання AWS крутою та заплутаною. До того часу, коли хтось витратить додаткові гроші, щоб отримати гнучкість AWS (наприклад, створить систему для автоматичного завантаження нових машин у відповідь на завантаження), вони могли б витратити ті самі гроші

на Linode, надавши своєму кластеру достатню ємність, щоб це не мало значення.

Існують інші послуги, подібні до Linode - DigitalOcean часто вважають подібною службою за порівнянною ціною. Однак мій досвід роботи з Linode був достатньо позитивним, тому я не відчував потреби досліджувати їх усі. Linode надає те, що

мені потрібно, має простий у використанні інтерфейс і робить це за чудову ціну. Тому ця робота буде зосереджена на розробці хмарних програм на Linode. Вибираючи постачальника, також важливо уважно прочитати його умови обслуговування, щоб переконатися, що використання за призначенням сумісне. Мало того, що деякі служби заборонені для деяких служб (наприклад, деякі служби забороняють масовий маркетинг електронною поштою), багато служб мають обмеження щодо швидкості масштабування мережі. Це здебільшого для запобігання зловживанням, але важливо дізнатися про це заздалегідь. У будь-якому випадку, ви можете звернутися до потенційних постачальників хмарних технологій і переконатися, що ваше заплановане використання відповідає їхнім інструкціям щодо надання послуг, перш ніж зупинитися на одному.

1.4 Висновок до розділу

У розділі розглянуто основні принципи хмарних технологій та їхнє відмінне місце серед інших форм інтернет-хостингу. Основною перевагою хмари є здатність до швидкого, автоматизованого масштабування інфраструктури без необхідності втручання у фізичні налаштування серверів. Розглянуто типи хмарних сервісів залежно від рівня абстракції:

Проаналізовано переваги та недоліки використання хмарної інфраструктури IaaS у порівнянні з PaaS. Основна теза полягає в тому, що IaaS забезпечує набагато більшу гнучкість, контроль, передбачуваність і економічну ефективність, ніж PaaS. Хоча PaaS може бути корисним для команд без досвіду адміністрування серверів, він накладає жорсткі обмеження на налаштування, доступ до логів, оновлення системи та інші аспекти роботи додатків, що може призвести до втрати контролю над власною технологією.

РОЗДІЛ 2 . РОЗГОРТАННЯ ТА МАСШТАБУВАННЯ ХМАРНОЇ ІНФРАСТРУКТУРИ

2.1 Створення віртуального сервера

Сподіваємось, якщо ви веб-розробник або менеджер, ви зможете зареєструватися в службі Linode без зайвих проблем. Для цього потрібна кредитна картка, але ви побачите, що витрати на роботу всього в цій книзі, швидше за все, менші, ніж на саму книгу, якщо ви вимкнете свої послуги, коли закінчите з ними. Перш ніж рухатися вперед, створіть обліковий запис у Linode зараз. Після того, як ви зареєструєтесь і ввійдете, Linode перенесе вас на вашу інформаційну панель, яка має виглядати приблизно так, як показано на рисунку 2.1 .

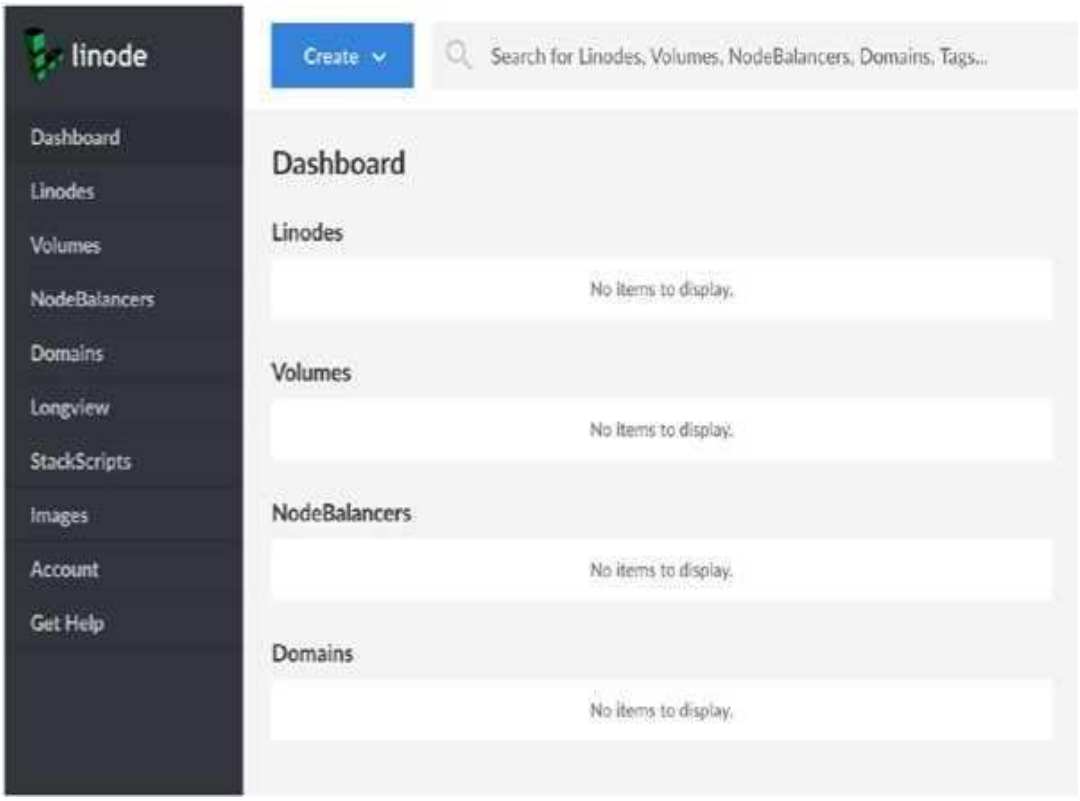

Рисунок 2.1. Інформаційна панель Linux

Зараз у вас нічого не налаштовано, тому ваша інформаційна панель досить порожня. Щоб почати, натисніть кнопку «Створити». Linode називає свої віртуальні сервери «вузлами» або «Linodes», тому виберіть «Linode», щоб

створити нову машину. Після цього Linode поставить вам кілька запитань, які допоможуть вам налаштувати вузол для використання. Хоча існує безліч хороших варіантів, скористайтеся наведеними тут, щоб мати змогу слідувати разом із роботою:

1. У розділі «Виберіть дистрибутив» виберіть «CentOS 7».
2. У розділі «Регіон» не має значення, який ви виберете, але ви повинні щоразу вибирати той самий, щоб ваші сервери спілкувалися один з одним. У цій книзі використовуватиметься засіб «Даллас, Техас».
3. У «Linode Plan» найдешевшим, достатнім для наших цілей, є план «Nanode 1GB». На момент написання цієї статті вартість цього плану становить менше 1 цента за годину.
4. У розділі «Мітка Linode» ми назвемо цю машину template_node .
5. Ви можете проігнорувати розділ «Додати теги». Теги корисні для групування машин разом, якщо їх багато.
6. У розділі "Пароль адміністратора" додайте пароль для цієї машини. Переконайтеся, що пароль безпечний, оскільки є багато хакерів, які просто намагаються ввести різні паролі для облікових записів адміністратора. Незвичайним є 50 000 таких спроб злому щомісяця.
7. На даний момент ви можете залишити "Додаткові додатки" недоторканими. Ми розглянемо резервні копії та приватні IP-адреси пізніше в книзі.

Після встановлення всіх цих параметрів натисніть кнопку «Створити», і Linode почне створювати вашу машину. Linode переведе вас на інформаційну панель для вашої машини, яка , серед іншого, матиме індикатор прогресу та «Стрічку активності» (див. рисунок 2.2). Коли індикатор виконання завершиться, ви тепер щасливий власник нового сервера в хмарі!

2.2 Налаштування служби бази даних

Будь-який хороший веб-додаток має базу даних. Моїм вибором завжди

була база даних PostgreSQL (www.postgresql .org). Існує міф, що PostgreSQL працює повільно. У цьому була частка правди - у 1990-х роках. Однак, починаючи з PostgreSQL 7, PostgreSQL був найефективнішим, і з кожним випуском він стає лише кращим. Крім того, PostgreSQL завжди чудово справлявся зі складними запитами, і залишається таким і сьогодні. PostgreSQL націлений на програмування без обмежень. Наприклад, у текстовому стовпці PostgreSQL ви можете зберігати до 4 гігабайт в одному стовпці одного рядка - і все одно сортувати за ним. У багатьох базах даних більша частина вашого часу витрачається на те, щоб дані відповідали бажаній архітектурі бази даних. Я виявив, що з PostgreSQL база даних набагато частіше вже готова до вашої власної архітектури даних.

Хоча це не робота про PostgreSQL, ми обговоримо деякі її функції, пов’язані з кластерами вузлів. Щоб інсталювати PostgreSQL, просто виконайте наступні дії від імені користувача root (весь цей розділ слід виконувати від імені користувача root): yum install -y postgresql-сервер Це встановить усі необхідні пакети для PostgreSQL. Щоб налаштувати початкову базу даних, введіть наступне: postgresql-setup initdb Це створює всі необхідні каталоги та файли для запуску PostgreSQL. Далі нам потрібно налаштувати метод автентифікації для підключення до наших баз даних PostgreSQL. PostgreSQL зберігає як свої дані, так і конфігурацію в каталозі /var/lib/pgsql/data . Файл, який керує доступом до бази даних, це pg_hba.conf. Відредагуйте цей файл (введіть nano /var/lib/pgsql/data/pg_hba.conf), щоб додати наступні два рядки вгору: локальні всі всі довірені хости всі всі всі md5 Перший рядок каже, що довіряти всім локальним з’єднанням (тобто не через мережу). Тому нам не знадобиться пароль під час роботи з базою даних безпосередньо в командному рядку. Другий рядок говорить, що будь-хто може підключитися до бази даних через мережу, використовуючи відповідний пароль. Це було б дещо небезпечно (ми не хочемо, щоб будь-хто міг підключитися до нашої бази даних), за винятком того, що за замовчуванням база даних прослуховує лише локальну адресу, 127.0.0.1, тому зараз ви все одно не можете підключитися до неї ззовні. Обов’язково збережіть

файл, а потім вийдіть із редактора.

Хоча це не робота про PostgreSQL, ми обговоримо деякі її функції, пов’язані з кластерами вузлів. Щоб інсталювати PostgreSQL, просто виконайте наступні дії від імені користувача root (весь цей розділ слід виконувати від імені користувача root): yum install -y postgresql-сервер Це встановить усі необхідні пакети для PostgreSQL. Щоб налаштувати початкову базу даних, введіть наступне: postgresql-setup initdb Це створює всі необхідні каталоги та файли для запуску PostgreSQL. Далі нам потрібно налаштувати метод автентифікації для підключення до наших баз даних PostgreSQL. PostgreSQL зберігає як свої дані, так і конфігурацію в каталозі /var/lib/pgsql/data . Файл, який керує доступом до бази даних, це pg_hba.conf. Відредагуйте цей файл (введіть nano /var/lib/pgsql/data/pg_hba.conf), щоб додати наступні два рядки вгору: локальні всі всі довірені хости всі всі всі md5 Перший рядок каже, що довіряти всім локальним з’єднанням (тобто не через мережу). Тому нам не знадобиться пароль під час роботи з базою даних безпосередньо в командному рядку. Другий рядок говорить, що будь-хто може підключитися до бази даних через мережу, використовуючи відповідний пароль. Це було б дещо небезпечно (ми не хочемо, щоб будь-хто міг підключитися до нашої бази даних), за винятком того, що за замовчуванням база даних прослуховує лише локальну адресу, 127.0.0.1, тому зараз ви все одно не можете підключитися до неї ззовні. Обов’язково збережіть файл, а потім вийдіть із редактора.

Команда psql надасть вам інтерактивний сеанс SQL до вашої бази даних. Щоб використовувати його, просто введіть: psql -U програма гостьової книги gbuser Командний рядок зміниться на щось на зразок guestbookapp=> , що означає, що ви перебуваєте в базі даних. Щоб вийти будь-коли, ви можете ввести \q . Як і багатьом командам системного адміністрування, PostgreSQL насправді не хвилює, де у файловій системі ви знаходитесь, коли виконуєте його команди. Він спілкується зі службою бази даних, яка працює у власному каталозі. Тепер, коли ми підключилися до бази даних, ми створимо одну таблицю за допомогою цієї команди: створити таблицю gb_entries(ідентифікатор серійний первинний

ключ, текст назви, текст електронної пошти, текст повідомлення, створена мітка часу, has_img bool за замовчуванням false); Поле id було створено з типом serial , який є механізмом автонумерації PostgreSQL. Щоб переглянути таблицю, яку ви щойно створили, введіть \d gb_entries . Коли закінчите, вийдіть із бази даних, ввівши \q .

2.3 Налаштування базового хмарного кластера

У цьому розділі ми розглянемо просту дворівневу архітектуру. Ця архітектура буде складатися з

- Сервер бази даних
- Набір веб-серверів
- Балансувальник навантаження, який керує трафіком між веб-серверами

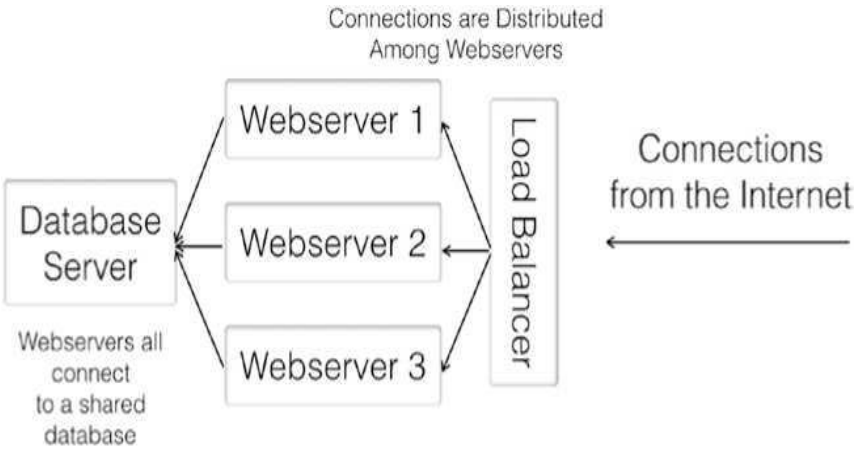

Рисунок 2.2. Діаграма простої дворівневої архітектури

Усі з’єднання надходять до єдиного балансувальника навантаження, завданням якого є перенаправлення з’єднань на один із кількох веб-серверів. Балансувальник навантаження не лише пересилає з’єднання, але й відстежує працездатність окремих веб-серверів і припинить надсилати з’єднання з веб-сервером, якщо він перестане відповідати. Тоді кожен веб-сервер спільно використовує один сервер бази даних. При розробці хмарних додатків

програміст повинен знати не тільки як налаштувати кластер, але й як його аналізувати. Якщо ви подивитеся на діаграму на рисунку 2.3 , то побачите, що всі веб-сервери залежать від одного сервера бази даних. Це робить сервер бази даних обмежуючим фактором кластера. Майже кожен кластер, незалежно від того, наскільки добре він розроблений, має деякі обмежувальні фактори. Мета полягає в тому, щоб мінімізувати їхній вплив на вашу архітектуру. Оскільки ця архітектура програми обмежена одним вузлом бази даних, її найкраще використовувати для розгортань малого чи середнього розміру, де більша частина обробки виконується на веб-серверах, а не в базі даних. Приклад програми, через свою простоту, фактично виконує дуже мало обробки на веб-сервері. Незважаючи на це, цей розділ покаже вам, як налаштувати сервери для розгортання в цій конфігурації.

Основна дворівнева програма вказує на те, що нам знадобиться кілька серверних вузлів. Ми могли б досягти цього, просто завантаживши нові машини з чистою копією CentOS і налаштувавши кожну окремо. Однак, оскільки ми вже витратили час на налаштування поточного сервера, щоб він працював так, як ми хочемо, нам слід скористатися часом, який ми витратили на налаштування всього. Linode пропонує кілька служб, які можуть виконати це завдання, кожна зі своїми перевагами та недоліками. Linode має можливість створювати збережені зображення, які можна використовувати для безпосереднього створення нових вузлів (замість вибору операційної системи ви можете вибрати збережене зображення). У Linode також є служба клонування, яка дозволяє клонувати існуючі машини, якщо і вихідна, і кінцева машини вимкнені (це не жорстке правило, але у вас можуть виникнути проблеми з узгодженістю, якщо ви спробуєте клонувати працюючу машину). Нарешті, ви можете створювати клони з резервних копій машини.

Я віддаю перевагу створенню клонів із резервних копій, тому що (а) ви все одно повинні регулярно створювати резервні копії своїх серверів, (б) вам не потрібно вимикати машину та нічого не робити, поки ви створюєте нові сервери, і (в) це змушує вас працювати із системою резервного копіювання та відчувати

себе комфортно з нею, поки ви також навчаєтеся клонувати сервери (колись вам потрібно буде відновити з резервної копії, тому добре освоїтися з процесом, перш ніж він вам знадобиться). Служба зображень Linode може працювати для цього, але вона має занадто багато обмежень для реального використання. Щоб запобігти використанню користувачами служби зображень як служби резервного копіювання, вони обмежили розмір і кількість зображень, але більшість моїх машин зазвичай більші за мінімальний розмір, дозволений для зображень Linode. Зверніть увагу, що інші хмарні служби (наприклад, DigitalOcean) пропонують подібні послуги, але з іншими наборами обмежень.

Якщо ми хочемо завантажити новий сервер, який є ідентичним клоном існуючого сервера, то все, що нам потрібно зробити, це створити резервну копію поточного сервера. Що нам потрібно зробити, це спочатку ввімкнути резервне копіювання на нашому вузлі сервера. Для цього просто увійдіть до Linode, клацніть свій вузол у списку та натисніть вкладку «Резервні копії» на інформаційній панелі вашого вузла. Це дасть вам кнопку з написом «Увімкнути резервне копіювання для цього Linode». Це додає невелику щомісячну плату, але воно того варте. Натисніть кнопку, і ваша машина автоматично матиме доступні щотижневі, щоденні та спеціальні знімки. Тепер екран має виглядати так, як на рисунку 2.3

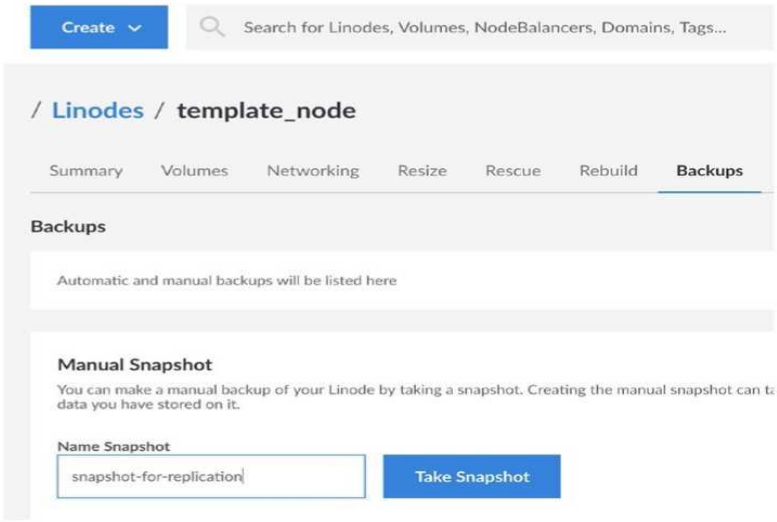

Рисунок 2.3. Екран керування резервним копіюванням Linode

Після того, як резервне копіювання почалося, ми можемо переходити до наступного завдання, а саме створення нового сервера. Новий сервер буде нашим сервером бази даних. Щоб створити машину, перейдіть до «Створити», а потім до «Linode». Однак замість вибору розповсюдження під заголовком «Зображення» виберіть «Мої зображення», а потім «Резервні копії». Це покаже список ваших вузлів, для яких доступні резервні копії. Клацніть свій вузол, і тоді він покаже доступні резервні копії, серед яких буде та, яку ми щойно створили. Дивіться рисунок 2.4 , щоб побачити, як це має виглядати. Ви можете створити стільки, скільки завгодно велика машина, але лише для практики ви можете використовувати їх найменший розмір машини. Цей вузол має бути в тому самому центрі обробки даних, що й інший (інакше вони не зможуть приватно, дешево та швидко спілкуватися один з одним), але Linode автоматично помістить новий вузол, створений із резервної копії, у той самий центр обробки даних.

Оскільки у нас дві машини, вони потрібні для спілкування. Вони могли спілкуватися через свою публічну IP-адресу, але це призводить до кількох проблем. Перш за все, якщо у вас є служби, які ви не хочете, щоб вони були загальнодоступними (наприклад, ваша база даних), важче запобігти доступу громадськості до них, якщо у вас є лише публічна IP-адреса. Крім того, Linode стягує гроші за трафік на вашій публічній IP-адресі, тому, якщо ви спілкувалися через цю IP-адресу, Linode стягуватиме з вас плату за внутрішній трафік. Тому мати приватну IP-адресу важливо, оскільки вона дозволяє комп’ютерам спілкуватися один з одним через швидку, безкоштовну та безпечнішу внутрішню мережу. Щоб уникнути цих проблем, Linode дозволяє налаштувати внутрішню мережу для ваших серверів. Усі сервери в обліковому записі спільно використовують внутрішню мережу, якщо вони для цього налаштовані. Щоб додати сервер до внутрішньої мережі, вам просто потрібно перейти на вкладку «Мережа» вашого вузла та натиснути «Додати приватний IPv4». Це дасть вам додаткову інформацію про приватні адреси, і ви можете просто натиснути «Виділити», щоб продовжити. Це призначить комп’ютеру приватну IP-адресу. Оскільки всі веб-сервери спілкуватимуться з цим сервером, ви захочете записати

згенеровану приватну IP-адресу. З цього моменту ми називатимемо цю адресу DB.MASTER.PRIVATE.IP , тому завжди замінюйте її приватною IP-адресою сервера, яку ви щойно записали. Зверніть увагу, що на деяких хмарних платформах (включаючи Linode) приватні IP-адреси не є повністю приватними. Тобто інші хмарні клієнти в тому самому центрі обробки даних можуть бути в цій мережі. Таким чином, хоча в приватній мережі безпечніше, ніж у загальнодоступній, приватні мережі в хмарах не гарантують підключення лише наших комп'ютерів. Таким чином, у робочій системі вам все одно потрібно буде вжити заходів, щоб запобігти небажаному доступу навіть у внутрішній мережі. Однак Linode фільтрує трафік до кожного вузла, тож вам не потрібно турбуватися про те, що хтось стежить за трафіком даних у внутрішній мережі. Вам потрібно буде перезавантажити вузол, щоб завершити процес. Після завершення завантаження ви все одно зможете побачити програму за адресою http:// NEW.NODE.PUBLIC.IP/list.php, але ви не зможете побачити його на приватній IP-адресі, оскільки, як зазначалося, вона приватна.

Тепер на цій машині є повна копія програми та бази даних. Однак він все ще налаштований як односерверна система. Нам потрібно налаштувати її як головну базу даних для кластера машин. У цьому розділі, який слід виконувати як користувач root, буде показано, що вам потрібно зробити, щоб це сталося. Оскільки ця машина є клоном template_node , це означає, що всі користувачі, програми та конфігурації були скопійовані на цей вузол. Таким чином, ви можете підключитися до машини як root, використовуючи пароль, який ви встановили раніше. Щоб використовувати його як сервер бази даних для інших вузлів, вам потрібно буде ввімкнути базу даних для прослуховування з'єднань з цих вузлів. За замовчуванням PostgreSQL лише прослуховує підключення через інтерфейс локального хосту. Ми не хочемо, щоб PostgreSQL прослуховував підключення в загальнодоступному Інтернеті. Тому ми хочемо налаштувати PostgreSQL так, щоб він прослуховував підключення як на локальному хості, так і на своїй приватній IP-адресі.

Для цього, як користувач root, відкрийте файл

/var/lib/pgsql/data/postgresql.conf за допомогою nano та змініть рядок із зазначенням listen_ addresses . Змініть цей рядок на: listen_addresses = 'localhost,DB.MASTER.PRIVATE.IP'

Обов’язково замініть DB.MASTER.PRIVATE.IP фактичною приватною IP - адресою вашого вузла. Якщо на початку рядка є позначка коментаря (#), обов’язково видаліть її; інакше команда не буде активною. Збережіть файл за допомогою Ctrl-o (просто натисніть клавішу повернення, щоб перевірити назву файлу, якщо вона запитає). Потім, щоб вийти, використовуйте Control-x. Тепер перезапустіть PostgreSQL командою: systemctl перезапустіть postgresql Крім того, ви захочете відкрити брандмауер, щоб він міг приймати віддалені з’єднання для PostgreSQL. Ви можете зробити це за допомогою: firewall-cmd --add-service postgresql firewall-cmd --add-service postgresql --permanent Зазвичай я підтримую роботу веб-сервера на сервері бази даних, щоб я міг це перевірити. Однак, якщо ви хочете, ви можете вимкнути веб-сервер за допомогою команд: systemctl зупинити httpd systemctl вимкнути httpd

Тепер, коли ми налаштували базу даних, настав час налаштувати веб-сервери. Зауважте, що насправді ми не будемо використовувати template_node як сервер. Мені подобається тримати поруч маленьку машину, яка просто використовується як шаблон для майбутніх коробок, особливо для веб-вузлів. Таким чином я можу мати одну невелику машину, яка є актуальною, з резервною копією тощо, і коли я буду готовий створити новий "образ", я просто створю резервну копію з іменем для використання. Зауважте, що це буде перезаписувати попередній знімок зображення, але для моїх цілей це зазвичай нормально. Якщо вам потрібно зберігати старі версії, просто зберігайте вузол шаблону для кожної конфігурації, яку ви хочете підтримувати. Тепер ми налаштуємо template_node як шаблон веб-сервера. Нам потрібно внести лише три зміни:

1. Вимкніть базу даних на цій машині.
2. Змініть код веб-додатку, щоб він вказував на нашу нову базу даних.
3. Увімкніть приватну IP-адресу на машині, щоб вона могла використовувати приватну мережу.

Щоб виконати перше завдання, все, що нам потрібно зробити, це увійти на машину template_node як root і запустити:

systemctl зупинити

postgresql systemctl

вимкнути postgresql

Щоб виконати друге завдання, нам потрібно лише змінити файл common.php. Я б запропонував змінити його на вашій локальній машині та використати SFTP для передачі нового файлу. Однак ви також можете використовувати nano на сервері, щоб змінити його безпосередньо. Все, що вам потрібно зробити, це змінити рядок підключення. Там, де зараз написано host=localhost , змініть його на host=DB. MASTER.PRIVATE.IP, де DB.MASTER.PRIVATE.IP - приватна IP-адреса вашого вузла dbmaster . Вам потрібно буде внести цю зміну двічі - один раз у функції getReadOnlyConnection() і один раз у функції getReadWriteConnection() .

Коли ви закінчите, завантажте код назад на сервер за допомогою SFTP. На цьому етапі код не працюватиме, оскільки він не може отримати доступ до машини dbmaster . Це тому, що PostgreSQL прослуховує лише свою приватну IP-адресу, а template_node ще не має приватної IP-адреси для спілкування. Тому вам потрібно додати приватну IP-адресу до машини, щоб вона могла підключитися до dbmaster за своєю приватною IP-адресою. Використовуйте процес щоб створити приватну IP-адресу для машини (не забудьте перезавантажити після цього!). Після виконання цих кроків ваша машина template_node зможе підключитися до dbmaster , тому перевірте її. Перейдіть за IP-адресою вашого сервера template_node і подивіться, чи він все ще може працювати. Якщо так, то вітаємо, адже ви щойно запровадили маленьку дворівневу систему! Тепер, як я вже говорив раніше, ми насправді не будемо використовувати template_node для фактичного обслуговування запитів. Мета полягає в тому, щоб використовувати його так, щоб ми могли легко завантажувати нові вузли веб-сервера, щоб розширити потужність, коли нам це потрібно. Отже, тепер, коли template_node повністю налаштовано як веб-сервер, зробіть новий резервний знімок.

Цей крок критично важливий. Кожного разу, коли ви вносите зміни в template_node , ви повинні зробити новий резервний знімок, щоб нові вузли, які ви створюєте з нього, мали ваші нові зміни (хоча це взагалі не впливає на існуючі вузли). Тепер ми створимо три (або скільки завгодно) вузли веб-сервера для нашого кластера. Ось кроки для кожного нового вузла веб-сервера:

1. Створіть новий вузол, виконавши кроки. Переконайтеся, що (a) він створюється в тому самому центрі обробки даних, що й dbmaster , і (b) встановіть назву вузла на webnode! (або 2, або 3).

2. Додайте приватну IP-адресу до вузла, щоб він міг підключитися до dbmaster у приватній мережі, виконавши дії.

3. Після завершення завантаження комп'ютера переконайтеся, що він повністю функціональний, переглянувши свою веб-програму на загальнодоступній IP-адресі вузла (тобто, http:// WEB.NODE.PUBLIC.IP/list.php).

Наприкінці цього у вас має бути три машини, webnode1 , webnode2 і webnode3 , кожна з яких може діяти як інтерфейс вашого веб-додатку.

2.4 Покращення масштабованості за допомогою кешування

У нашій поточній програмі база даних є головним вузьким місцем. Це означає, що додавання додаткових веб-серверів суттєво не збільшить навантаження, яке може витримати хмара. Коли ви виявите вузьке місце, корисно приділити час і подумати про те, як цього вузького місця можна уникнути. У нашому випадку дані змінюються не так часто. Навіть якщо це так, отримання актуальних значень із гостьової книги не є критично важливим. Якби комусь довелося чекати кілька секунд або навіть хвилин, щоб побачити останній запис у гостьовій книзі, це не був би кінець світу. Якщо у вас є вміст, до якого часто звертаються, але ви можете дозволити собі бути лише трохи застарілим (або дуже застарілим), ви можете реалізувати кешування , щоб пришвидшити роботу. Кешування просто означає наявність тимчасового сховища результатів,

до якого ми можемо швидко отримати доступ. Бази даних працюють повільно, оскільки бази даних в першу чергу стурбовані цілісністю даних. Ви хочете знати, що ваші дані в безпеці, зберігаються на диску, не зникають і доступ до них можна отримати за допомогою довільних запитів. З іншого боку, кеш-пам’ять є ефемерною - зазвичай вона просто зберігається в пам'ять. Їх мета - швидкий пошук даних будь-якою ціною. Багато кеш-пам'яті, наприклад, просто почнуть викидати дані, якщо вони заповнять свою пам'ять. Це добре, тому що, якщо щось не існує в кеші, ми можемо перейти до бази даних і повторно отримати дані.

Коротше кажучи, бази даних - це постійність і надійність, а кеші - щоб отримати те, що я хочу, якомога швидше. Кеші зазвичай реалізуються як прості пари ключ/значення, зазвичай із доданою датою закінчення терміну дії. Тобто кожна частина даних, що зберігається в кеші, має призначений «ключ». Наприклад, оскільки існує лише один список усіх записів у гостьовій книзі, ми можемо мати єдиний «ключ» кешу для цього. У програмі це називатиметься listentlist . Проте кожен окремий запис у гостьовій книзі може отримати власний ключ кешу (тобто кеш міг знати, що він унікальний), який включатиме ідентифікатор бази даних запису. Ключі кешу зазвичай є звичайними рядками, і в більшості систем кешування вміст може бути будь-яким. Кеш-пам’ять не виконує жодної спеціальної обробки, вони просто зберігають все, що ви запропонуєте, у певному ключі, тому ви повинні бути обережними, щоб не використовувати той самий ключ для двох різних речей!

Кеш використовується таким чином, що програма спочатку визначає, який ключ використовувати для даних. Мета полягає в тому, щоб мати можливість легко вивести ключ кешу з параметрів URL-адреси. Програма починає роботу з перевірки, чи має кеш значення для ключа кешу. Якщо кеш містить значення, воно використовується. Якщо цього не відбувається, програма отримує значення «звичайним» способом (тобто, зазвичай звертаючись до бази даних за значенням і, можливо, виконуючи деякі додаткові обчислення чи маніпуляції), а потім зберігає його в кеші з датою закінчення терміну дії. Потім, незалежно від того, надійшло значення з кешу чи «звичайним» способом,

значення використовується в програмі. Після закінчення терміну дії (або кеш переповнюється значеннями), кеш-значення просто зникає для наступного запиту, змушуючи програму отримувати нове значення «звичайним» способом.

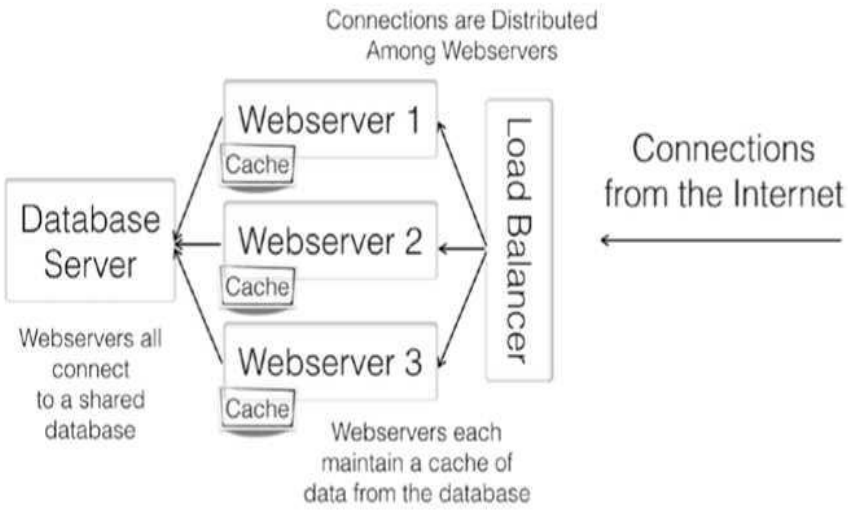

Рисунок 2.4. Дворівнева архітектура з локальним кешуванням

Як ви бачите, ця методологія доступу до кешу порівняно зі звичайним доступом гарантує, що хоча механізм кешування є кращим, механізм без кешу також доступний, і цей механізм заповнює кеш. Як зазначалося, база даних зазвичай вважається повільною частиною будь-якої програми, оскільки вона має бути дуже обережною, щоб правильно обробляти ваші дані та зберігати їх назавжди. Кеші, з іншого боку, вважаються ефемерними. Якщо ми просто хочемо раптово очистити кеш, це не вплине на роботу сторінки, тому що вона просто повернеться до бази даних і знову отримає значення. Якщо сайт сильно завантажений і постійно запитує той самий вміст, навіть кешування на кілька секунд може значно збільшити швидкість і зменшити використання ресурсів. Попередньо ми протестували цей кластер, оскільки він обслуговує приблизно 250 запитів на секунду на сторінці списку. Якби ми кешували результати цієї сторінки протягом 10 секунд, це було б на 2500 запитів до бази даних менше за цей період! На рисунку 2.5 показано, як ми можемо змінити нашу стандартну дворівневу архітектуру, щоб додати рівень кешування. У цій архітектурі кожен окремий веб-сервер підтримує власний кеш результатів. Це може призвести до

невеликих невідповідностей між веб-серверами, оскільки вони могли оновитися їх результати в різний час. Однак, якщо термін дії не встановлено занадто далеко в майбутньому, ці проблеми мінімальні. Крім того, якщо є проблеми, пам'ятайте, що «липкість» балансувальника навантаження. Це прив'язує конкретного користувача до певного сервера, що означає, що користувач завжди буде на тому самому сервері, що означає, що він завжди використовуватиме той самий кеш сервера. Крім того, це робить кешування більш ефективним, оскільки кожен сервер має кешувати дані лише для підмножини користувачів.

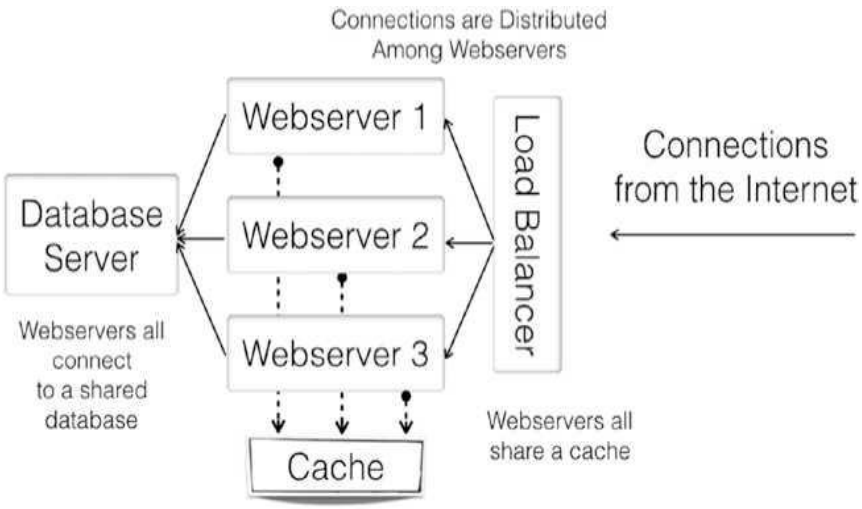

Рисунок 2.5. Дворівнева архітектура з глобальним кешуванням

Що кешувати та як довго це кешувати, дуже залежить від програми. Для багатьох архітектур існують фрагменти, які можна кешувати протягом хвилин або навіть днів, а також інші фрагменти, які можна кешувати лише протягом секунд або не можуть бути кешовані взагалі. Кешування часто може призводити до несподіваних результатів, тому під час розробки варто включити функцію, за допомогою якої можна повністю вимкнути кешування, щоб перевірити, чи є проблема у вашому кеші. Якщо важливо підтримувати узгодженість кешу на всіх ваших серверах, варто розглянути іншу архітектуру, яка використовує глобальний кеш, як на рисунку 2.5. У цій архітектурі замість того, щоб кожен веб-сервер підтримував власний кеш, існує спільний кеш, до якого всі мають

доступ. Це додає трохи більше затримки мережі, оскільки всі виклики кешування мають виконуватися через мережу, але це додає більшої послідовності ваших результатів.

Залежно від того, як він реалізований, цей вид зовнішнього кешу також може стати незручним. Однак існує багато серверів кешування, які можуть охоплювати кілька серверів і балансувати запити між серверами. Тим не менш, це також додає рівень складності управління до головоломки. Майже в усіх випадках я вважаю, що мати єдину зовнішню службу кешування важко налаштувати та підтримувати, і це приносить невелику користь, якщо вона взагалі є. У більшості ситуацій розміщення кешу на кожному веб-сервері дає вам найбільш масштабовану ефективність, навіть якщо це коштує невеликої узгодженості, яку, як згадувалося, можна полегшити за допомогою «липкості» балансувальника навантаження.

2.4 Висновки до розділу

Цей розділ демонструє встановлення та первинне налаштування PostgreSQL, включно з ініціалізацією бази даних, налаштуванням автентифікації через pg_hba.conf та створенням таблиці у базі даних. Надано приклади використання команд для взаємодії з базою через psql, що забезпечує основу для зберігання даних веб-додатків. Побудова дворівневої архітектуриБуло реалізовано базову хмарну інфраструктуру. Розглянуто питання кешування.

Кешування є потужним інструментом для покращення масштабованості веб-додатків, особливо у випадках, коли бази даних виступають вузьким місцем у системі. Воно дозволяє зменшити навантаження на базу даних, забезпечуючи швидкий доступ до часто використовуваних, хоча й потенційно не зовсім актуальних, даних. Локальне кешування на рівні веб-серверів забезпечує найкращу масштабованість, навіть попри можливі незначні невідповідності в даних між серверами. Отже, грамотне впровадження кешування - це ефективний спосіб підвищити масштабованість і швидкодію системи без значного ускладнення її архітектури.

РОЗДІЛ 3. ОПТИМІЗАЦІЯ І РОЗПОВСЮДЖЕННЯ ХМАРНИХ РІШЕНЬ

3.1 Реалізація кешування в програмі

Архітектура кешування, яку ми збираємося запровадити тут, показана на рисунку 3.1 , тому що її легше реалізувати, а також тому, що нею легше керувати в довгостроковій перспективі. Що ми збираємося зробити, так це виконати зміну конфігурації на машині template_node , а потім просто вимкнути наші існуючі

блоки webnodeX і замінити їх новими блоками, реплікованими з template_node . Це набагато простіше (і менш схильне до помилок), ніж проходити кожен сервер і вносити зміни. Перше, що нам потрібно зробити, це встановити службу кешування. Ми будемо використовувати memcached для нашої служби кешування, оскільки її легко запускати, отримувати доступ і керувати нею. Щоб установити memcached і ввімкнути його, просто введіть наступні команди від імені root:

yum install -y memcached

systemctl увімкнути

memcached systemctl

запустити memcached

Вам також знадобляться розширення PHP для memcached . Їх потрібно скомпілювати, тому нам потрібно буде встановити ще кілька розширень (знову ж таки, як root).

```
function getCache() {
  $conn = new Memcached();
  $conn->addServer("localhost", 11211);
  return $conn;
}
```

Рисунок 3.1. Функція підключення Memcache

yum install -y libmemcached

yum install -y php74-php-pecl-memcached

Не забувайте про «d» наприкінці memcached , тому що є інше розширення під назвою memcache , яке не робить того, що ми хочемо. Зауважте, що це також увімкне розширення в PHP. Він робить це за нас, але якщо вам потрібно налаштувати PHP-сторінку конфігурації, файл знаходиться в каталозі /etc/opt/remi/php74/php.d . Тепер нам потрібно перезапустити процес PHP-FPM, щоб використовувати нові розширення PHP: systemctl перезапустіть php74-php-fpm Далі нам потрібно змінити нашу програму, щоб створити з’єднання з нашою локальною службою memcached . Тому додайте код з рис 6-3 у common.php .

Номер 11211, згаданий у коді, є портом, який memcached прослуховує за замовчуванням. Ваш код має використовувати кеш таким чином:

1. Створіть ключ кешу, щоб унікально ідентифікувати інформацію в кеші.
2. Перевірте, чи інформація вже існує в кеші. Якщо так, використовуйте його.
3. Якщо інформація ще не існує в кеші, знайдіть її повільним способом (тобто виконайте запит до бази даних).
4. Візьміть інформацію та збережіть її в кеш-пам’яті за допомогою цього ключа з майбутнім часом закінчення (у цьому випадку ми встановимо термін дії 10 секунд після встановлення).

Далі ми оновимо функцію list.php для використання кешу. На рисунку 3.2 показано, як переписати list.php для використання кешу. Це лише верхня частина сценарію, яка зберігає результат запиту в $result . Фактичний вихідний код HTML залишається незмінним.

Після того, як ваш новий хмарний кластер буде запущено, настав час випробувати нову архітектуру та перевірити, чи досягли ми якогось приросту продуктивності. Для цього налаштування я запустив ApacheBench на окремих серверах і на збалансованому кластері. Окремі сервери, оскільки вони більше не покладалися на базу даних як вузьке місце, змогли обслуговувати понад 800 запитів на секунду! Крім того, оскільки в базі даних не було вузьких місць, продуктивність могла масштабуватися майже лінійно. Лінійне масштабування означає, що кожна коробка, яку ви додаєте, однаково підвищує продуктивність. У цьому випадку з одним сервером наша продуктивність становила 800 запитів на секунду, два сервери дали близько 1500 запитів на секунду, а на трьох серверах ми змогли постійно обслуговувати понад 2300 запитів на секунду! Зауважте, що якщо ви не бачите подібних приростів продуктивності, перевірте конфігурацію балансира, щоб переконатися, що ви не ввімкнули липкість сеансу в будь-який момент, оскільки це обмежить ваші сеанси ApacheBench одним сервером. Крім того, переконайтеся, що на вкладці «Параметри» також не

ввімкнуто «Результат підключення клієнта». рисунок 3.2 показує результат ApacheBench на повному кластері.

Отже, те, що ми дізналися, це те, що кешування не лише підвищило швидкість програми, але й покращило масштабованість програми. Оскільки нам потрібно звернутись до бази даних лише після закінчення терміну дії кешу, швидкість бази даних тепер відносно неважлива. Фактично, навіть якщо ми знову пришвидшимо базу даних, це мало б відносно невеликий вплив на нашу загальну ефективність просто тому, що воно рідко використовується. З іншого боку, якщо ви дійсно використовуєте програму, ви побачите, що після публікації запису в гостьовій книзі вона не відразу відображатиметься на сайті. Насправді, якщо ви швидко перезавантажите сторінку, ви можете виявити, що вона то з'являється, то зникає залежно від того, коли сервер, на якому ви перебуваєте, оновлює свій кеш. Це можна пом'якшити різними способами. По-перше, ви можете зменшити кількість часу, протягом якого дані кешуються. У цьому тесті дуже мало різниці в тому, кешуєте ви 1 секунду чи 10.

Отже, просто змінивши рядок закінчення терміну дії кешу на такий (тобто встановивши час закінчення терміну дії на 1 секунду замість 10), програма дуже швидко оновиться без істотного впливу на продуктивність цих великих навантажень, які ми тестуємо: $expiration_seconds = 1;

```
Server Software:        Apache/2.4.6
Server Hostname:        BALANCER.IP.ADDRESS.HERE
Server Port:            80

Document Path:          /list.php
Document Length:        383 bytes

Concurrency Level:      50
Time taken for tests:   0.425 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      560000 bytes
HTML transferred:       383000 bytes
Requests per second:    2353.76 [#/sec] (mean)
Time per request:       21.243 [ms] (mean)
Time per request:       0.425 [ms] (mean,
   across all concurrent requests)
Transfer rate:          1287.21 [Kbytes/sec]
   received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.4      0       4
Processing:     2   19  13.4     14     108
Waiting:        2   19  13.4     14     108
Total:          2   20  13.4     14     109
```

Рисунок 3.2. Вивід ApacheBench для кешованої конфігурації

Навіть більше, якщо ви зберігаєте сеанси, прив’язані до певного сервера, ви можете наказати серверу очистити окремі ключі або навіть увесь кеш під час певних подій. Тому в кінці create.php ми могли б додати такі рядки, щоб очистити кеш списку на цьому сервері:

```
$cache = getCache();
$cache->delete("entrylist");
```

Або, якщо програма була настільки складною, що потрібно було видалити багато ключів, код міг просто очистити весь кеш за допомогою $cache->flush(); . У будь-якому випадку, як бачите, архітектура кешування може зробити вашу програму дещо складнішою та складнішою в управлінні, але зазвичай вони того варті через часто різке підвищення продуктивності та масштабованості.

Кеш-пам’ять, хоч і надзвичайно корисна, приносить свої власні проблеми. Для кращого налагодження веб-сторінок найкраще завжди мати набір параметрів, які можна передати програмі, щоб вона вимкнула кешування. Наприклад, у багатьох моїх власних програмах передача no_cache=1 в URL-адресі вимкне кешування. зазвичай це перше місце, куди я звертаюся, коли повідомляють про проблеми. НЄГЄ є деякі ознаки того, що у вас можуть виникнути проблеми з кешуванням, і що з ними робити:

• Проблема: ваша програма видає старий вміст, хоча в базі даних є новіший вміст.

Діагноз: кеш-пам'ять зберігає застарілий вміст.

Рішення: термін дії вашого вмісту мине швидше або надайте додаткову інформацію про ключ кешу, щоб повідомити кеш, коли вам потрібні нові дані.

Проблема: ваша програма видає невідповідні дані з урахуванням параметрів.

Діагностика: це часто трапляється, коли ваш ключ кешу недостатньо конкретний. Наприклад, якщо вміст виходив не тією мовою, це могло означати, що вам потрібно додати поточну мову як частину ключа кешу.

Рішення: додайте більше параметрів до свого ключа кешу, щоб переконатися, що ви дійсно ідентифікуєте кожну частину унікального вмісту за допомогою унікального ключа, інакше ви можете вирішити, що цей вміст занадто специфічний для кешування.

Проблема: одна й та сама сторінка видає різний вміст під час кожного перезавантаження.

Діагноз: вміст кешу на кожному сервері залежить від того, коли до нього було доступно.

Рішення: є кілька способів виправити це. ВИ можете (а) зменшити проміжок часу до завершення терміну дії, (б) збільшити «липкість» балансувальника навантаження, щоб переконатися, що одна й та сама особа завжди звертається до того самого сервера (і, отже, того самого кешу), (в) використовувати глобальний (або синхронізований) кеш замість локального

кешу та (г) додати додаткові ключові параметри кешу, щоб краще координувати дані, які користувач отримує з кешу.

Проблема: Кеш не прискорює вашу програму настільки, як ви думали.

Діагноз: або ви ніколи не використовуєте свій кеш, або ви кешуєте неправильні речі.

Рішення: ОСКІЛЬКИ існує багато способів, як це може піти не так, є багато способів це виправити. часто це відбувається, коли кожен користувач отримує доступ до різного набору даних, і тому нічого не витягується з кешу. Це можна виправити, збільшивши розмір кешу та/або інтелектуально попередньо завантаживши в кеш дані, до яких, імовірно, буде доступ. ВАМ також може знадобитися збільшити термін дії ваших даних. однак може статися, що вам доведеться виконувати багато постобробки даних, і це займає більше часу, ніж фактичний запит.

3.2 Реплікація бази даних

Деякі речі просто не можна кешувати. Спеціальні звіти, оперативні зміни та сайти, де шаблони доступу розподілені по великій кількості непов’язаних сторінок, важко оптимізувати за допомогою кешування. Для подібних робочих навантажень ви можете розгорнути більший сервер бази даних, але зрештою навіть вони зіткнуться з обмеженнями. Таким чином, багато архітектур додатків включають реплікацію бази даних, де існує більше ніж один сервер бази даних, що обслуговує запити.

Існує багато типів реплікації бази даних залежно від ваших потреб. Основні типи реплікації включають

Реплікація після відмови : у цій конфігурації репліковані сервери не допомагають із навантаженням, але вони гарантують, що якщо основний сервер бази даних вийде з ладу, існує база даних з актуальною інформацією, готова взяти на себе роботу.

Реплікація головної/репліки : у цій конфігурації головна база даних є

єдиною, що має доступ для читання/запису. Сервери-копії отримують дані, коли вони записуються на головному (або незабаром після цього), але є копіями основної бази даних лише для читання. Усі оновлення надходять до головної бази даних, але запити можуть надходити до основного або будь-якого сервера-репліки. Це також відомо як реплікація "головний/підлеглий", де репліка бази даних вважається "підлеглою базою даних".

Мультиголовна реплікація: у цій конфігурації всі бази даних вважаються однаково «головними» базами даних, і записи можуть виконуватися в будь-якій з них. Записи в будь-яку базу даних потім синхронізуються з рештою кластера.

У цій главі буде зосереджено увагу на реплікації головний/репліка, оскільки її найпростіше реалізувати, вона має найменшу кількість практичних проблем і дає найбільші результати за найменших зусиль. Мультимайстерна реплікація використовується рідко, тому що її важко налаштувати, підтримувати та підтримувати ефективну роботу, і дуже мало баз даних її підтримують. Навіть якщо підтримується, реплікація з кількома головними користувачами часто створює нові проблеми, які важко вирішити, такі як конфлікти даних (тобто, коли конфліктні дані фіксуються на двох різних серверах). Тому, щоб зберегти простоту, ця робота буде зосереджена на конфігураціях майстер/репліка. рисунок 3.3 показує концептуальний вигляд типової архітектури майстер/репліка.

Система реплікації PostgreSQL значно просунулася за ці роки як у функціях, так і в простоті використання. Хоча це не складно використовувати, для розуміння потрібні деякі пояснення. Вбудована система реплікації PostgreSQL використовує для реплікації техніку, відому як потокове передавання журналів. PostgreSQL, щоб гарантувати узгодженість даних, створює так званий журнал попереднього запису або WAL. По суті, PostgreSQL записує в WAL зміни, які він збирається зробити, а потім насправді вносить зміни. Це означає, що якщо сервер бази даних вимикається під час оновлення, він має запис про те, що він робив, і може просто завершити операцію, коли знову ввімкнеться.

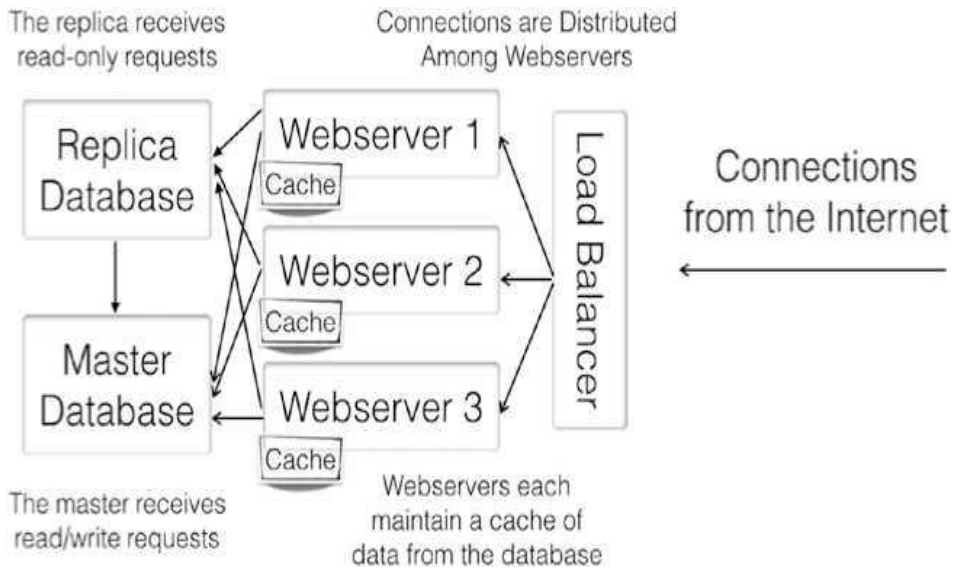

Рисунок 3.3 Діаграма архітектури бази даних Master/Replica

Цікаво, що це саме та інформація, яку також повинен знати сервер реплікації. Тому, щоб реалізувати реплікацію бази даних, PostgreSQL просто надсилає файли WAL на сервери реплікації, які так само впроваджують зміни. Цей тип реплікації відомий як потокова передача WAL. Щоб досягти цього в нашому кластері, нам потрібно налаштувати нашу основну базу даних, щоб отримувати підключення до реплікації. Увійдіть до dbmaster як користувач root, відредагуйте файл /var/lib/pgsql/data/postgresql.conf і встановіть такі параметри:
wal_level = hot_standby
wal_keep_segments = 32
max_wal_senders = 4
hot_standby = on

Якщо ви використовуєте пізнішу версію (9.4 або новішу) PostgreSQL, вам також потрібно буде встановити: max_replication_slots = 4

Однак цей параметр порушить роботу версії PostgreSQL, яка поставляється з CentOS 7.2, з якою ми працюємо в цій книзі. Ці зміни конфігурації досягають кількох речей:

- wal_level налаштовує «журнал попереднього запису» PostgreSQL (тобто WAL), щоб він зберігав достатньо деталей для надсилання всього, що знадобиться серверу-репліці.

• wal_keep_segments зберігає достатньо даних із WAL, які висять навколо після його використання, щоб сервер-репліка все ще міг отримати доступ до даних, якщо він відстає. Ми встановили значення 32, що є досить консервативним параметром. Це дозволяє новій репліці мати багато часу для повної синхронізації та запобігає незначним мережевим збоям і уповільненням роботи серверів.

• параметра max_wal_senders має бути встановлено принаймні кількість серверів-реплік плюс два (і, якщо ви вирішите інсталювати новішу версію PostgreSQL, можливо, вам знадобиться встановити те саме значення для параметра max_ replication_slots). •hot_standby = on дозволяє репліці сервера відповідати на запити. Далі нам потрібно додати користувача-реплікатора до бази даних PostgreSQL. Введіть psql -U postgres, щоб отримати доступ до бази даних, а потім введіть наступне, щоб створити реплікатор користувача для реплікації (все в одному рядку):

СТВОРИТИ реплікатор ролей З ПАРОЛЕМ РЕПЛІКАЦІЇ 'mypassword' LOGIN; Потім введіть \q , щоб вийти. Далі нам потрібно надати віддаленим серверам дозвіл на відкриття підключень реплікації до цього сервера. Додайте такий рядок до /var/lib/pgsql/data/pg_hba.conf : Реплікатор реплікації хосту всі md5 Тепер нам потрібно перезапустити базу даних, щоб нові параметри вступили в силу: systemctl перезапустіть postgresql Тепер наш сервер повністю готовий приймати запити на реплікацію. Тепер ми можемо налаштувати репліку сервера. Щоб налаштувати сервер-репліку PostgreSQL, перше, що нам потрібно зробити, це створити новий вузол Linode, на якому він буде працювати, скопіювавши наш шаблонний вузол за допомогою стандартної процедури. Для цієї вправи назвіть новий вузол dbreplica . Вам також потрібно буде додати приватну IP-адресу до цієї машини та записати її (у решті цієї книги ми називатимемо її DB.REPLICA.PRIVATE.IP). Тепер завантажте dbreplica та увійдіть як root . Якщо ви дотримувались інструкцій, на цьому комп’ютері не має бути запущено PostgreSQL. Якщо він запущений, ви можете вимкнути його за допомогою systemctl stop postgresql . Після вимкнення PostgreSQL нам потрібно очистити

наявну інсталяцію PostgreSQL. Зробіть це за допомогою такої команди: rm -rf /var/lib/pgsql/data/ * Тепер нам потрібно запросити початкову бінарну резервну копію від головної системи як відправну точку для реплікації. Це досягається за допомогою команди pg_basebackup . Щоб створити вихідну початкову точку резервного копіювання, спочатку перейдіть до користувача postgres так: su - postgres Далі введіть таку команду (всі в одному рядку): pg_basebackup -x -U реплікатор -h DB.MASTER.PRIVATE.IP -D /var/lib/pgsql/data Він запитає у вас пароль, а потім скопіює весь екземпляр PostgreSQL з головної бази даних, включаючи файли конфігурації. Далі вам потрібно буде налаштувати файли конфігурації після завершення цього кроку. Файл postgresql.conf , який ми налаштували в розділі 5 має команду listen_addresses , яка містить приватну IP-адресу сервера. На жаль, оскільки це було скопійовано з головної бази даних, наразі вона має приватну IP-адресу головної бази даних. Щоб виправити це, просто відкрийте /var/lib/pgsql/data/postgresql.conf і змініть конфігурацію listen_ addresses на читання: listen_addresses = 'localhost,DB.REPLICA.PRIVATE.IP' Після того, як це завершиться успішно, вам потрібно повідомити PostgreSQL, що цей сервер використовуватиметься як гаряче резервне. Це робиться шляхом вказівки серверу переходити в «режим безперервного відновлення» під час запуску. Для цього ми створюємо файл /var/lib/pgsql/data/recovery.conf із таким вмістом (останні два рядки мають бути введені в одному рядку): standby_mode = 'увімкнено' primary_conninfo = 'хост=DB.MASTER.PRIVATE.IP порт=5432 користувач=пароль реплікатора=мій пароль' Коли все буде готово, вам потрібно повернутися до користувача root таким чином: вихід Тепер, коли ви знову є користувачем root, нам потрібно знову ввімкнути базу даних PostgreSQL і переконатися, що вона ввімкнеться автоматично після перезавантаження: systemctl запустити postgresql systemctl увімкнути postgresql Нам також потрібно переконатися, що re є дірою в брандмауері для отримання підключень до бази даних: firewall-cmd --add-service postgresql firewall-cmd --add-service postgresql --permanent На даний момент ваша система запущена та працює як репліка сервера! Щоб переконатися в цьому, виконайте наступну команду, яка перерахує

всі запущені процеси PostgreSQL: ps afxw|grep postgres Один із процесів у списку повинен містити слово recovering або startup у вихідних даних. Це означає, що база даних запущена, активна та живиться з журналів WAL головного. Ви також можете перевірити файли журналів, які будуть у каталозі /var/ lib/pgsql/data/pg_log . У кінці файлу журналу має бути написано щось на кшталт «система бази даних готова приймати підключення лише для читання» та «потокова реплікація успішно підключена до основного». Реалізації PostgreSQL називаються «примірниками» і можуть містити будь-яку кількість баз даних. Якщо ви дотримуєтесь інструкцій у цій книзі, ваш екземпляр postgresQl містить лише одну базу даних (насправді три, оскільки postgresQl завжди постачається з парою встановлених баз даних шаблонів, template0 і template1). створити нову базу даних досить легко за допомогою програми командного рядка createdb або вказівки створити базу даних під час роботи psql. у будь-якому випадку пам’ятайте, що оскільки файл WAL postgresQl є функцією системного рівня (тобто файли WAL є спільними для всього екземпляра бази даних), потокова реплікація postgresQl реплікує весь екземпляр postgresQl, а не лише одну базу даних.

Якщо відокремлення одного сервера-репліки не дає вам необхідного підвищення продуктивності, ви можете мати стільки серверів-реплік, скільки вам потрібно. Ви можете зробити це шляхом реплікації вузла шаблону та повторення процесу, описаного в Розділі 7.2, або шляхом безпосередньої реплікації сервера-репліки. Реплікація сервера-репліки не така автоматична, як реплікація веб-вузлів, але вона може заощадити кілька кроків. Щоб зробити це, вам потрібно спочатку ввімкнути резервне копіювання на вашому поточному вузлі dbreplica , а потім створити нові екземпляри репліки з резервних копій dbreplica . Після створення кожного екземпляра репліки вам потрібно підключитися до нового сервера репліки за протоколом ssh і встановити значення listen_addresses у /var/lib/pgsql/data/postgresql.conf на власну приватну IP-адресу, оскільки за замовчуванням буде встановлено на адресу dbreplica . Після цього вам потрібно буде перезапустити PostgreSQL за допомогою systemctl

restart postgresql . Після того, як ви створите кілька копій своєї бази даних, вам потрібно змінити код, щоб використовувати ваші бази даних. Було б чудово, якби ми могли створити балансувальник навантаження для наших реплік баз даних, а потім просто вказати весь код нашої програми на балансувальник навантаження. На жаль, Linode наразі не має можливості створювати внутрішні балансувальники навантаження (тобто балансувальники навантаження, які приймали лише запити в приватній мережі), тому нам доведеться надати власний механізм розподілу навантаження. Натомість ми будемо емулювати цю функцію в коді програми, вибираючи сервер випадковим чином під час підключення до бази даних. Оскільки у нас немає балансувальника навантаження, нам доведеться жорстко закодувати список серверів у вашому коді програми, а для додавання нового сервера-репліки також знадобиться змінити код програми та повторно надіслати цей код на всі веб-сервери. Щоб зрозуміти, як код програми буде змінено, припустімо, що тепер у нас є три сервери-репліки з приватними IP-адресами DB.REPLICA.PRIVATE.IP, DB.REPLICA2.PRIVATE.IP і DB.REPLICA3. PRIVATE.IP . Щоб змусити наше підключення лише для читання циклічно перемикатися між ними, ми перепишемо нашу функцію getReadOnlyConnection() згідно з рисунком 3.4 . Щойно цей код буде розміщено в нашому кластері, усі наші запити лише для читання будуть збалансовані навантаженням між декількома серверами-репліками, як показано на рисунку 3.5. Це означатиме, що нашим єдиним вузьким місцем є запити на читання та запис бази даних. У більшості програм у будь-якому випадку переважають запити лише для читання, тому наявність вузьких місць у запитах на читання та запис зазвичай не викликає проблем.

```
function getReadOnlyConnection() {
  $serverlist = [
    "DB.REPLICA.PRIVATE.IP",
    "DB.REPLICA2.PRIVATE.IP",
    "DB.REPLICA3.PRIVATE.IP"
  ];

  $idx = array_rand($serverlist);

  $host = $serverlist[$idx];

  return new PDO("pgsql:host=" . $host . ";" .
    "port=5432;dbname=guestbookapp;" .
    "user=gbuser;password=mypassword"
  );
}
```

Рисунок 3.4. Підключення до групи серверів-реплік

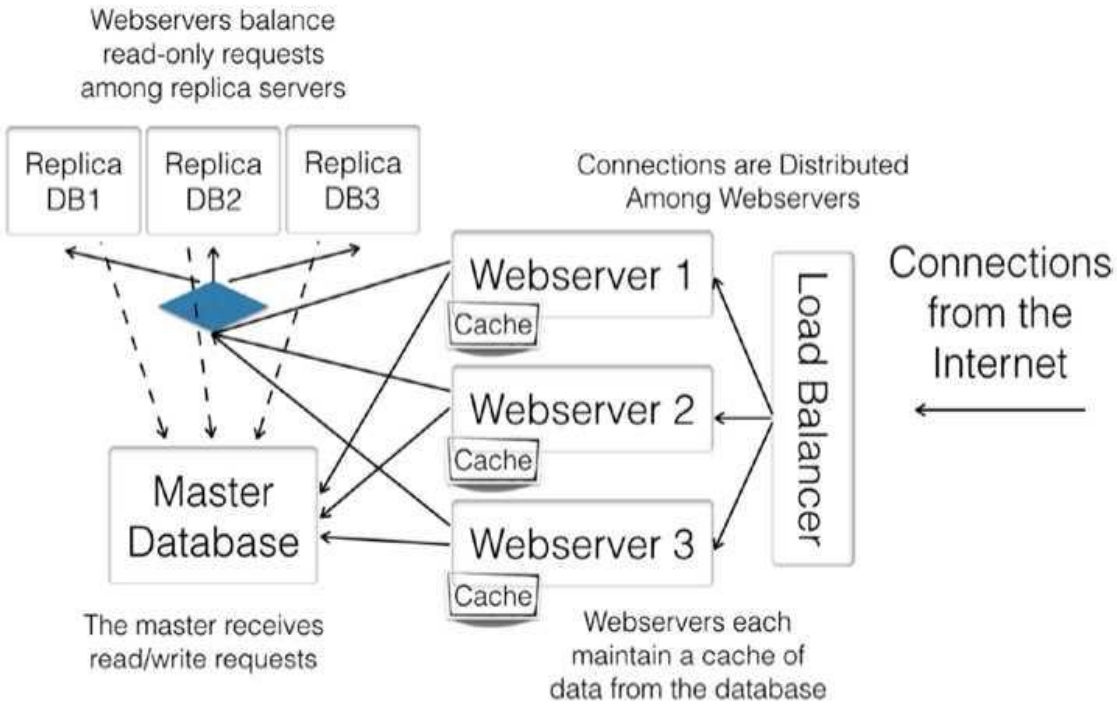

Рисунок 3.5. Діаграма конфігурації бази даних із кількома репліками

Іншим варіантом масштабованості бази даних є шардинг даних. Шардинг бази даних відносно простий за концепцією - це просто означає, що ви розділяєте дані таким чином, щоб не всі дані залежали від однієї системи баз даних. Наприклад, ви можете розділити свої дані так, щоб усі клієнти, чиї імена починаються на «А» і «В», були в одній базі даних, усі клієнти, чиї імена

починаються на «C» і «D», були в іншій і так далі. Ви можете розділити дані будь-яким способом, який вам подобається, за умови, що система може легко визначити, до якої бази даних їй потрібно зробити запит. Основний момент полягає в тому, що замість того, щоб усе керувалося однією системою баз даних, самі дані розподіляються між різними системами баз даних. У таких системах зазвичай або програма піклується про сегментування, або існує проміжний рівень, який обробляє передачу підключень до відповідних баз даних.

Було розроблено деякі нові інструменти, такі як pg_shard , який має на меті бездоганну роботу як плагін бази даних. У будь-якому випадку шардинг несе з собою цілий набір проблем із керуванням даними та їх цілісністю. Одним із пунктів баз даних було забезпечення узгодженості даних. Шардинг, по суті, усуває багато засобів захисту, які пропонували бази даних для досягнення масштабу. Тому важливо бути впевненим, що ви знаєте, чому ви хочете шардувати і як ви хочете шардувати, щоб мінімізувати ризик. Наприклад, якщо ви розділили клієнтів на різні бази даних, ви також повинні розділити пов’язані записи, щоб клієнт обслуговувався єдиною базою даних і щоб їхні записи також керувалися разом. Уявіть, що вам потрібно було відновити записи з резервної копії, але деякі пов’язані записи були в іншій системі баз даних! Шардинг вимагає багато роботи, вимагає багато планування та сильно залежить від особливостей вашого використання та робочого навантаження, тому важко говорити про це в загальних рисах. Тим не менш, якщо ви шукаєте більше способів масштабувати свою базу даних, шардинг, безумовно, є варіантом.

Шардинг також може працювати в поєднанні з іншими методами, такими як реплікація головної репліки або реплікація з кількома головними, але, знову ж таки, це вимагає багато додаткової любові, турботи та керування, щоб забезпечити належну роботу. Шардинг легший, якщо ваші групи входу не мають спільних даних. Наприклад, припустімо, що ви створили систему електронного маркетингу. Різні клієнти рідко діляться будь-якими даними в такій системі. Тому немає проблем зберігати їхні записи повністю окремо. Ви потенційно можете запустити кілька повністю незалежних хмар, кожна з яких має різний

набір клієнтів. Логіни від клієнтів A, B, C і D будуть перемішані в Cloud 1, логіни від клієнтів E, F, G і H будуть перемішані в Cloud 2 і так далі. Якщо всіма цими хмарами керувати окремо, таке налаштування також забезпечить певний рівень пом'якшення аварій, оскільки навряд чи виникне повне відключення всіх центрів обробки даних одночасно.

3.3 CDN Мережа доправлення (і розповсюдження) контенту

Поки що наші зусилля щодо масштабування нашого хмарного додатку були зосереджені на обслуговуванні динамічного вмісту (тобто вмісту, отриманого із запитів до бази даних). Однак для багатьох сайтів динамічний вміст насправді є найменшою частиною їхньої системи. Насправді на будь-якому веб-сайті більшість запитів стосуються навіть не динамічного вмісту, а статичного вмісту - ваших зображень, таблиць стилів і JavaScript. Тому, розглядаючи способи масштабування вашої програми в хмарі, важливо не забувати також масштабувати свої статичні ресурси. Ви можете масштабувати свої статичні ресурси, створивши більше інтерфейсних веб-серверів, оскільки зазвичай вони обслуговують статичні ресурси. Однак це може стати дорожчим (ви повинні підтримувати більше серверів онлайн) і важчим для керування (більше вузлів означає більше керування). Масштабувати статичні активи насправді набагато легше, ніж масштабувати динамічні, оскільки вам не потрібно так сильно турбуватися про те, що відбувається, коли вони змінюються. Тому існують сервіси, створені спеціально для масштабування статичних ресурсів. Служба, яка масштабує доставку статичних ресурсів, відома як «Мережа доставки вмісту», також відома як CDN.

Більшість CDN працюють так, що вони мають глобально розподілену мережу серверів вмісту. Скажімо, у вас є зображення розміром 10 Мб, яке ви хочете обслуговувати CDN. Зазвичай, якби цей користувач попросив ваш сервер надати зображення, тоді час обробки та пропускна здатність вашого сервера застрягли б під час надсилання зображення розміром 10 МБ. Якщо користувач жив на іншому березі океану, це додаткова проблема, оскільки ваш сервер

витрачатиме багато ресурсів на керування повільним і шумним з'єднанням. CDN дозволяє перенаправляти користувачів на URL-адресу зображення в CDN. CDN, коли вперше бачить URL-адресу, зазвичай нічого не знає про зображення, але має правила, які вказують йому, як знайти оригінальне зображення у вашій службі. З цього моменту, після того, як CDN захопить зображення вперше, щоразу, коли хтось запитає це зображення, CDN буде впоратися з доставкою зображення користувачам, не проходячи через ваші сервери. Крім того, більшість CDN мають сервери, розташовані в різних фізичних місцях. Ці місця відомі як «точки присутності» (Points of Presence, PoP), а сервери там часто називають «граничними серверами» (через це CDN часто називають «граничними кешами»). Використання периферійних серверів у різноманітних точках доступу дозволяє CDN не лише забезпечувати масштабованість через велику кількість серверів, крайові сервери дозволяють CDN надавати сервери, які знаходяться в безпосередній фізичній близькості від користувача.

Це значно пришвидшує роботу вашого користувача, якщо він може отримати багато даних із найближчих серверів замість того, щоб перетинати океани, щоб отримати дані. CDN надають фактично нескінченний масштаб для обслуговування статичних активів - для будь-якої відомої CDN вам не потрібно хвилюватися про переповнення їхньої мережі. Поки актив не змінюється, CDN зможе обробляти будь-яку кількість запитів для цього активу. Зазвичай основною ціною для CDN є пропускна здатність. Однак багато CDN мають менші витрати на пропускну здатність, ніж хмарні сервери. Тепер, з Linode, вам знадобиться надзвичайно активний сайт, щоб випередити включену пропускну здатність. Тим не менш, якби ви це зробили, ви б заплатили трохи менше за пропускну здатність, якби вона надавалася CDN. У будь-якому випадку, навіть якщо у вас є достатня пропускна здатність вашого сервера, переваги покращеної масштабованості за допомогою CDN зазвичай варті своєї невеликої ціни, оскільки вам не потрібно постійно працювати з серверами, щоб мати справу з трафіком, який може або не з'являтися. Вам потрібно платити лише за ту пропускну здатність, яку ви фактично використовуєте.

Налаштування простого CDN

На щастя, більшість CDN надзвичайно легко налаштувати. У цій книзі ми будемо використовувати Amazon CloudFront як CDN, хоча існує багато інших варіантів (CloudFlare, StackPath, CDN77 і Fastly, це лише деякі). Хороша річ у CDN полягає в тому, що оскільки послуги, які вони надають, досить прозорі, їх легко комбінувати та підбирати постачальників послуг для CDN. Принцип роботи CloudFront дуже простий:

1. Ви розміщуєте весь статичний вміст на своєму основному сайті. Це «офіційне» сховище вашого вмісту.

2. Ви створюєте хост на CloudFront для обслуговування вашого вмісту (зазвичай це матиме назву, як-от xyzabc.cloudfront.net).

3. Ви повідомляєте серверу CloudFront URL-адресу свого основного сайту.

4. Кожного разу, коли ви посилаєтеся на статичний вміст свого сайту, ви посилаєтеся на нього за допомогою URL-адреси CloudFront, а не посилаєтеся на вміст на вашому власному сервері. Наприклад, якщо ваше зображення було на http://mysite. example.com/mydirectory/myimage.png, коли ви посилаєтеся на нього, ви використовуватимете URL-адресу http://xyzabc.cloudfront.net/mydirectory/myimage. png. CloudFront знатиме з вашої конфігурації, як знайти myimage.png , кешуватиме його та надасть вашим користувачам.

Коли CloudFront вперше отримає запит на зображення, він перейде на ваш сайт, щоб отримати його. У подальшому будь-які майбутні запити обслуговуватимуться безпосередньо CloudFront за допомогою сервера поруч із користувачем, який запитує зображення. Давайте розглянемо, як насправді це зробити за допомогою Amazon AWS і CloudFront. Перше, що вам потрібно зробити, це зареєструватися в AWS на http://aws.amazon.com . Я припускаю, що ви можете виконати процес реєстрації без мене. AWS має величезну кількість доступних послуг, тому на інформаційній панелі замість того, щоб перелічувати їх усі, ви можете шукати одну. Введіть «CloudFront» у рядок пошуку, і ви зможете перейти до інформаційної панелі CloudFront. Оскільки ви вперше

використовуєте CloudFront, ви просто надасте кнопку з написом «Створити розповсюдження». З точки зору CloudFront, дистрибутив - це просто служба реплікатора CDN. Після того, як ви натиснете «Створити розповсюдження», CloudFront запитає вас про спосіб доставки та запропонує вам вибрати «Веб» або «RTMP». RTMP - це протокол, який використовується для потокової передачі великої кількості відеовмісту. Однак, оскільки ми лише розповсюджуємо базові ресурси, такі як зображення та таблиці стилів, ми вирішимо розпочати роботу з базової веб-доставки.

Коли CloudFront вперше отримає запит на зображення, він перейде на ваш сайт, щоб отримати його. У подальшому будь-які майбутні запити обслуговуватимуться безпосередньо CloudFront за допомогою сервера поруч із користувачем, який запитує зображення. Давайте розглянемо, як насправді це зробити за допомогою Amazon AWS і CloudFront. Перше, що вам потрібно зробити, це зареєструватися в AWS на http://aws.amazon.com . Я припускаю, що ви можете виконати процес реєстрації без мене. AWS має величезну кількість доступних послуг, тому на інформаційній панелі замість того, щоб перелічувати їх усі, ви можете шукати одну. Введіть «CloudFront» у рядок пошуку, і ви зможете перейти до інформаційної панелі CloudFront. Оскільки ви вперше використовуєте CloudFront, ви просто надасте кнопку з написом «Створити розповсюдження». З точки зору CloudFront, дистрибутив - це просто служба реплікатора CDN. Після того, як ви натиснете «Створити розповсюдження», CloudFront запитає вас про спосіб доставки та запропонує вам вибрати «Веб» або «RTMP». RTMP - це протокол, який використовується для потокової передачі великої кількості відеовмісту. Однак, оскільки ми лише розповсюджуємо базові ресурси, такі як зображення та таблиці стилів, ми вирішимо розпочати роботу з базової веб-доставки.

Рисунок 3.6. Список дистрибутивів CloudFront

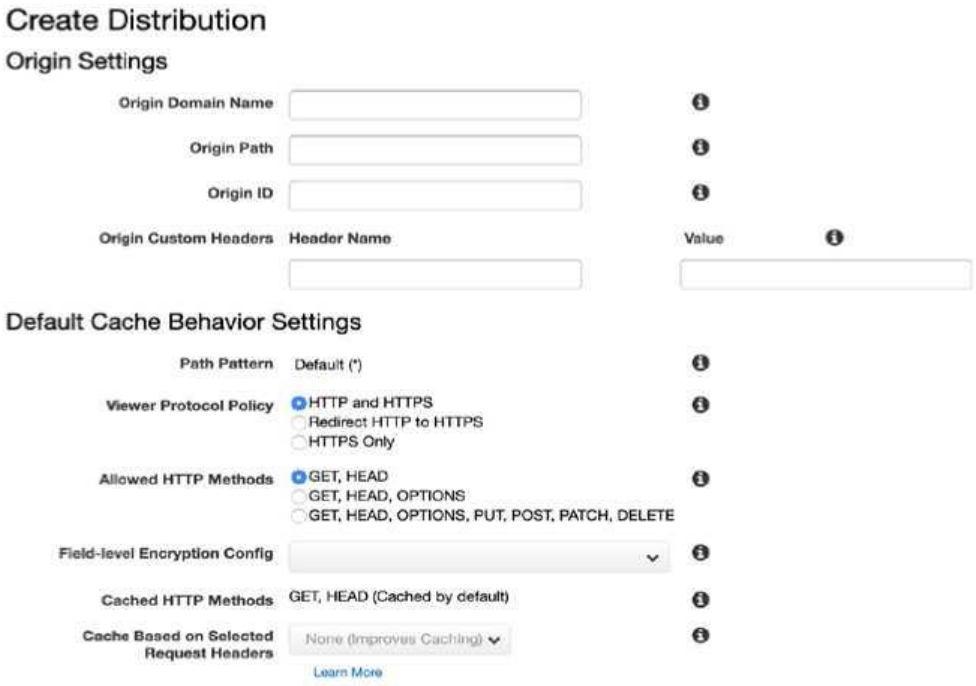

Рисунок 3.7. Створення розповсюдження CloudFront

Унизу сторінки є кнопка «Створити розповсюдження». Після того, як ви натиснете цю кнопку, ви потрапите у список дистрибутивів, де буде показано ваш новий дистрибутив, подібно до рисунку 3.7 . Найважливішою частиною цього екрана є «Доменне ім’я», яке показує, як отримати доступ до щойно створеного дистрибутива (може знадобитися налаштувати розмір поля, щоб побачити повне ім’я). Він також надасть вам статус, який займе від 5 хвилин до години, щоб перейти від «Виконується» до «Розгорнуто». Після розгортання CDN запрацює.

Після зміни статусу на «Розгорнуто» це доменне ім’я тепер повністю відображатиме ваш вихідний сайт. Однак це буде лише статична версія вашого

сайту. Якщо ваш сайт зміниться, CloudFront не оновлюватиме свої активи, якщо ви не покажете це. Припустімо, доменне ім’я, яке вам надав CloudFront, було xyzabc.cloudfront.net. Це означає, що якщо ви підете до http://xyzabc. cloudfront.net/list.php , він покаже вам список вашої гостьової книги. Однак якщо ви потім підете і змініте свій список гостьової книги, ці зміни не відобразяться на вашому CDN - він розглядає все як статичний актив. Ось чому здебільшого CDN обслуговують лише статичні ресурси - зображення, таблиці стилів, JavaScript тощо. Тому замість того, щоб отримувати доступ до всього сайту через CDN, давайте модифікуємо додаток, щоб обслуговувати лише нашу таблицю стилів через CDN. Все, що нам потрібно зробити, це змінити один рядок common.php . У функції getHeader() нам просто потрібно змінити тег <link> на читання: <link rel="stylesheet" href=" http://xyzabc.cloudfront.net/ guestbook.css" /> Обов’язково замініть xyzabc.cloudfront.net на доменне ім’я вашого дистрибутива! Щойно це буде розгорнуто на всіх серверах, ваша таблиця стилів тепер обслуговуватиметься з CDN.

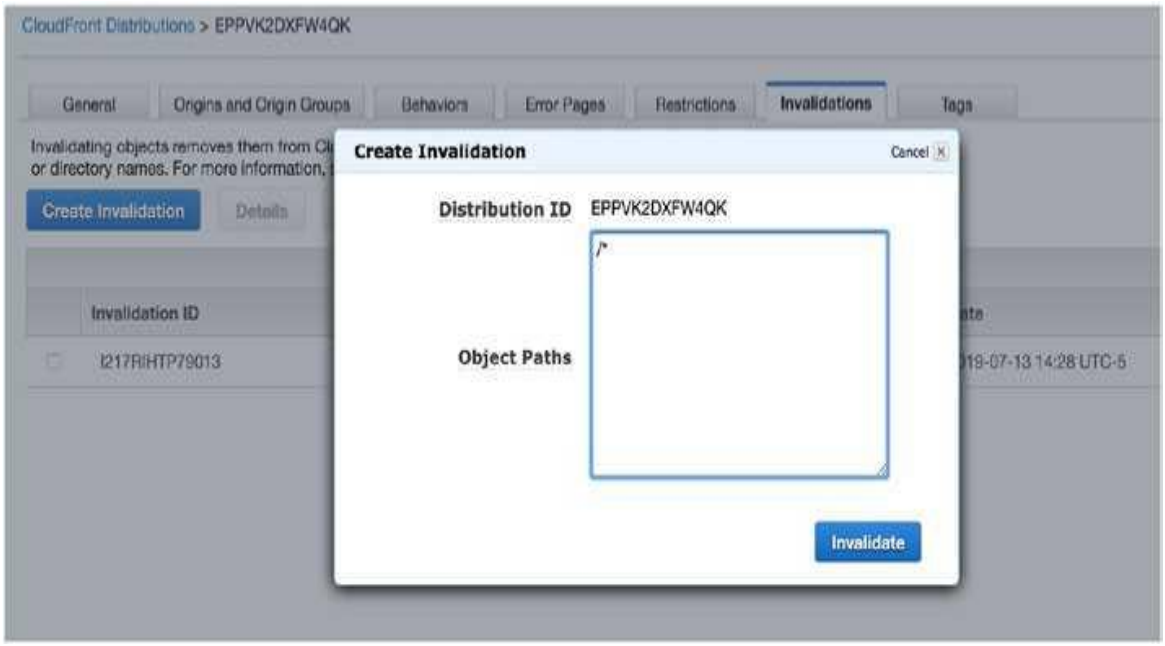

Рисунок 3.8. Видалення вмісту з CDN

Це означає, що ваш сервер майже ніколи більше не обслуговуватиме таблицю стилів. Час від часу термін дії частини кешованого вмісту CDN може

закінчитися, але це залежить від CDN. CDN оптимізує для себе, скільки даних він зберігає протягом якого часу, і як часто він повторно запитує ваші оригінальні файли. Для більш розширених програм ви можете налаштувати Apache на надсилання заголовка Expires: HTTP або заголовка Cache-Control: HTTP, щоб указати максимальний час, протягом якого CDN має зберігати ваші дані. Однак припустімо, що ви розгорнули нову версію програми, яка насправді мала оновлену таблицю стилів. Це означає, що версія, яку CDN обслуговує для вас, зараз застаріла - ймовірно, у кеші досі є стара версія. Це зовсім не проблема, це просто означає, що вам потрібно вручну сказати CDN «недійсним» ваш вміст, щоб він запитував його знову. Щоб зробити недійсним вміст у CloudFront CDN, спочатку клацніть ідентифікатор свого дистрибутива CloudFront. Це приведе вас до інформаційної сторінки з описом усіх параметрів, встановлених у вашому дистрибутиві. Праворуч є вкладка під назвою «Анвалідації». Перейдіть на цю вкладку, а потім натисніть «Створити недійсність». Це приведе вас до екрана, схожого на рисунок 3.8 . У полі «Шляхи до об’єктів» просто очистіть усе, що там є, і введіть лише /* , щоб зробити все недійсним. У той час як CloudFront дає вам детальний доступ для визнання недійсними та видалення певних елементів із CDN, я вважаю, що в більшості ситуацій просто зробити недійсним усе це простіше та чистіше. Щоб зробити це, натисніть кнопку «Визнати недійсним». Може знадобитися кілька хвилин, щоб анулювання поширилося на всі сервери CloudFront, але незабаром усі сервери обслуговуватимуть ваш новий вміст. Ви дізнаєтесь, коли це зроблено, коли «Статус» анулювання зміниться на «Завершено».

3.4 Використання Google Cloud Platform

Google Cloud Platform (GCP). GCP - нова дитина в блоці, створена в 2008 році (Linode розпочався в 2003 році, а AWS з’явився в мережі в 2006 році). GCP дуже схожий на AWS, хоча його налаштування трохи громіздкіше. Ви можете розпочати роботу з GCP, зареєструвавшись на сторінці https://cloud.google.com .

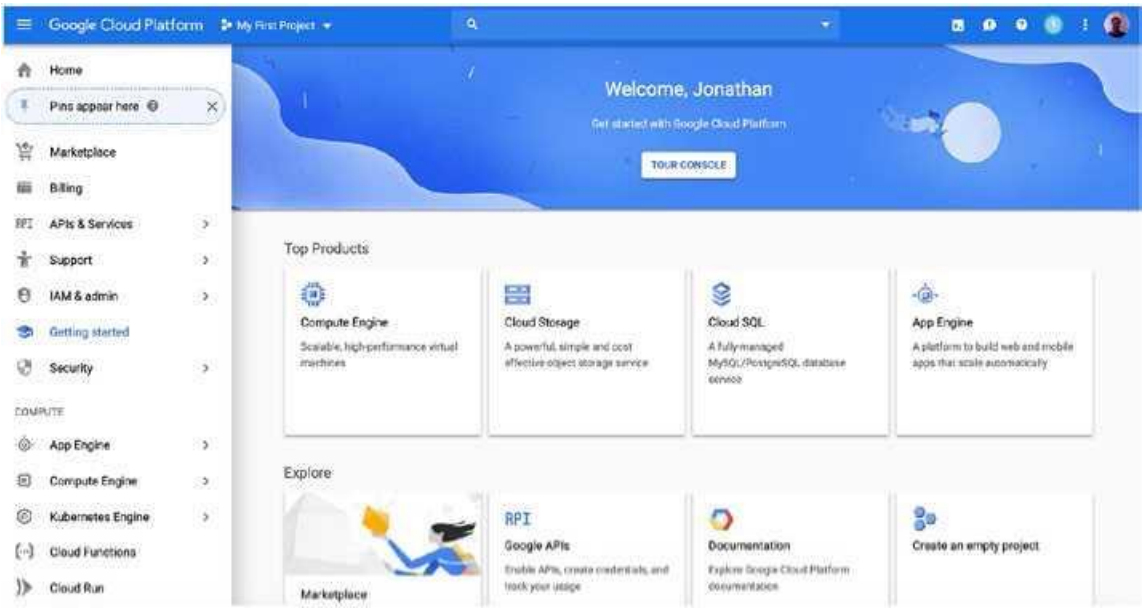

Рисунок 3.9 Екран привітання GCP

Перше, що потрібно знати про GCP, це те, що все організовано в «Проекти». Кожен проект - це щось на кшталт власного облікового запису, кожен із власними ресурсами, послугами тощо, але всі доступні за допомогою одного входу. Коли ви входите в Google Cloud Platform, це має виглядати приблизно так, як показано на рисунку 3.9 . Зауважте, що на верхній панелі поруч із текстом "Google Cloud Platform" зазначено назву вашого поточного проекту. Ви можете клацнути поточний проект, щоб змінити проекти або створити новий. Для цього прикладу я збираюся створити новий проект під назвою Book Examples Project. У лівій частині екрана наведено список служб GCP. Щоб створити нову машину, перейдіть до розділу «Compute» служб, натисніть «Compute Engine», а потім натисніть «VM Instances». GCP називає свої машини (або вузли, як їх називає Linode) «примірниками віртуальної машини». Щоб створити нову машину, натисніть кнопку «Створити екземпляр». Це призведе до появи екрана, схожого на рисунок 3.10 РОЗДІЛ 11 ВИКОРИСТАННЯ ХМАРНОЇ ПЛАТФОРМИ GOOGLE Ми викличемо шаблон-вузол нашої машини та виберемо найменший тип машини (у цьому випадку "f1-micro"). Для завантажувального диска ми виберемо CentOS 7. Коли ви прокрутите вниз, ви побачите розділ «Брандмауер». Переконайтеся, що ввімкнено «Дозволити HTTP-

трафік» і «Дозволити HTTPS-трафік». Решта параметрів за замовчуванням ви можете залишити. Коли все налаштовано, натисніть «Створити», і GCP створить для вас нову машину та поверне вас до екрана списку віртуальної машини.

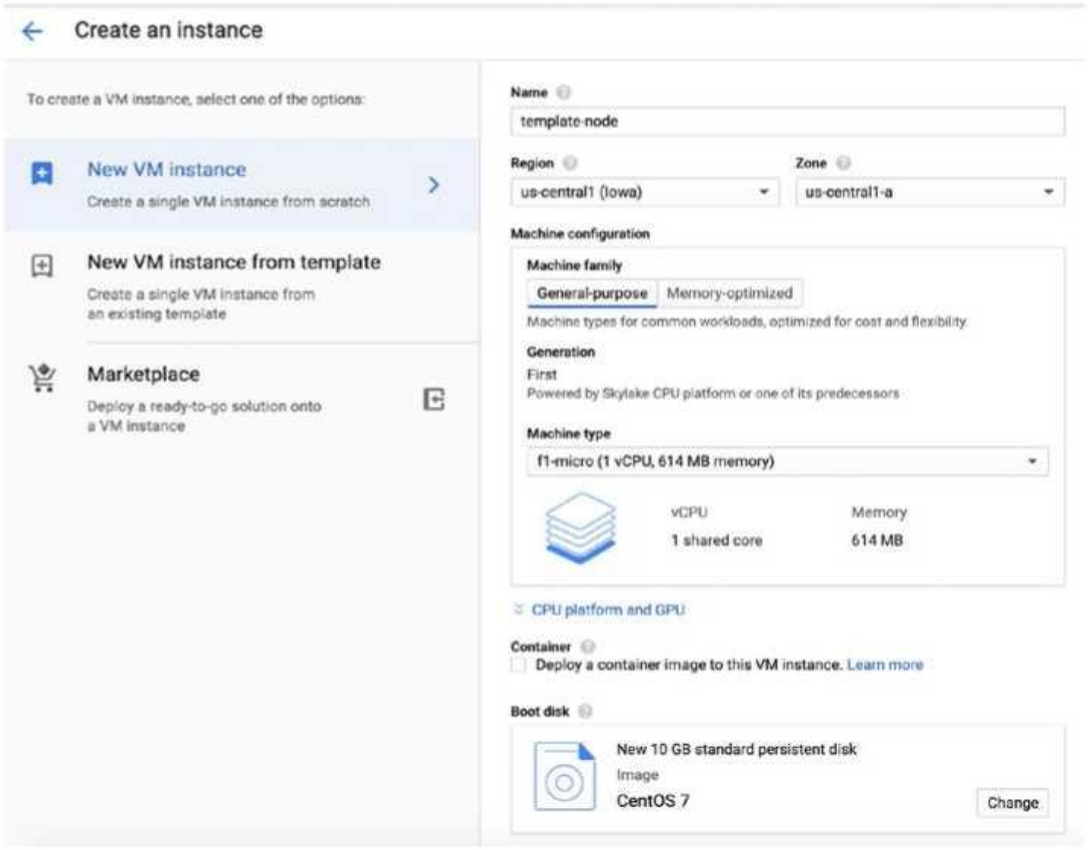

Рисунок 3.10 Створення екземпляра GCP VM

Коли машину створено, ви можете увійти за допомогою опції «Підключити». У спадному меню в стовпці «Підключення» виберіть «Відкрити у вікні браузера», і це дасть вам сеанс ssh у вашому браузері. Ця інсталяція CentOS дуже схожа на інсталяцію Linode, за винятком того, що (а) вона не включає nano (що можна виправити за допомогою простої команди yum install -y nano), і (б) вона автоматично створює для вас користувача та входить у систему як цього користувача. Ви можете легко переключитися на root, виконавши команду: sudo su - З цього моменту ви можете встановити nano та виконати всі кроки налаштування, по суті ідентичні. Одна цікава відмінність між GCP і Linode полягає в тому, що на Linode публічна IP- адреса вашого вузла фізично прикріплена до вашого вузла. тобто, коли ви видаєте ip addr show, він показує

публічну IP-адресу. Коли ви додаєте приватну IP-адресу, вона додається до вашого вузла. Однак у GCP ваш вузол починається з приватної та публічної адреси. Однак приватна адреса є єдиною, яка фізично зіставлена з пристроєм. Публічна IP-адреса налаштована на мережевому обладнанні для пересилання цих запитів на ваш комп'ютер. Таким чином, у всіх ваших конфігураціях на вашому комп'ютері ви використовуватимете приватну IP-адресу, яка входить у вашу коробку. Крім того, вам не потрібно турбуватися лише про прослуховування вашої приватної IP-адреси, оскільки це все, що у вас є. Мережа GCP контролює, які служби можуть отримувати доступ до вашої скриньки ззовні (саме тому ви встановили прапорці HTTP та HTTPS під час налаштування екземплярів віртуальної машини, щоб наказати GCP направляти такі типи запитів на вашу приватну IP-адресу). Загалом, якщо не вказано інше, усе в GCP обмежено локальною мережею.

Для того, щоб використовувати наш шаблон-вузол як сервер бази даних, нам потрібно буде підготувати коробку для віддаленого доступу до нашої бази даних. Для цього нам потрібно зробити наступне:

1. Змініть /var/lib/pgsql/data/postgresql.conf і встановіть listen_addresses='*'. Нам не потрібно спеціально встановлювати для нього приватну IP-адресу, оскільки це єдина IP-адреса, яку ми маємо, і вона не буде доступною з Інтернету, якщо ми не налаштуємо GCP, щоб це дозволити. Перезапустіть PostgreSQL за допомогою systemctl, перезапустіть postgresql, щоб зміни набули чинності.

2. Змініть брандмауер, щоб дозволити підключення до сервера PostgreSQL. Це будуть команди firewall-cmd --add-port 5432 /tcp і та сама команда з доданим прапорцем --permanent.

3. Змініть код PHP getReadOnlyConnection() і getReadWriteConnection(), щоб вони підключалися до правильної приватної IP-адреси. Тепер сервер готовий до використання кластером із збалансованим навантаженням.

Створення образу реплікації в GCP є дещо надто складним триетапним процесом. Спочатку вам потрібно створити «моментальний знімок» вашого

екземпляра, потім вам потрібно створити «зображення» з цього знімка, і, нарешті, вам потрібно буде створити «шаблон екземпляра». Знімок, по суті, є резервною копією машини. Образ, по суті, є знімком, який призначений для створення нових завантажувальних образів для машин. Нарешті, шаблон екземпляра поєднує зображення разом із налаштуваннями машини (розмір, конфігурація тощо), які можна використовувати для дуже швидкого розгортання ідентичних машин. Створення знімка є досить простим процесом. У головному меню (клацніть три смужки поруч із написом «Google Cloud Platform») перейдіть до розділу «Compute», виберіть «Compute Engine», а потім виберіть «Snapshots». Натисніть кнопку «Створити знімок». Він запитає назву для знімка, вихідний диск і розташування. Назвіть його як завгодно, виберіть наявний екземпляр віртуальної машини як вихідний диск, і ви можете залишити це місце («Мультирегіональний» є найбільш гнучким вибором). Натисніть «Створити», і GCP створить для вас новий знімок. Тепер GCP дозволяє створювати екземпляри зі знімків.

Однак, щоб використовувати ще більше функцій GCP, найкраще створити те, що GCP називає «зображенням» із вашого знімка. Для цього перейдіть у головне меню, потім у розділ «Compute», виберіть «Compute Engine», а потім виберіть «Images». Він завантажить величезний список попередньо налаштованих зображень. Ви можете їх ігнорувати. Ми хочемо створити власний імідж. Натисніть кнопку «Створити зображення», щоб почати. Дайте зображенню назву (наприклад, шаблон-зображення) і встановіть для «Джерела» значення «Знімок». Відкриється меню із запитом, з якого знімка ви хочете створити зображення. Виберіть знімок, який ви щойно створили. Якщо ви хочете, ви можете додати ім’я «Родина». Це дозволить вам створювати оновлені версії цього екземпляра з тим самим ім’ям «Сімейства». Тепер натисніть «Створити». Тут ви можете створювати нові машини із зображення. Під час створення нової машини у розділі «Завантажувальний диск» ваш образ буде доступний на вкладці «Користувацькі зображення». Нарешті, нам потрібно запакувати це зображення в шаблон екземпляра. Шаблони екземплярів

знаходяться в головному меню в розділах «Compute Engine» і «Instance Templates». Процес створення шаблону екземпляра такий самий, як і для створення звичайного екземпляра віртуальної машини. Різниця полягає в тому, що він не створюватиме жодних екземплярів відразу, натомість його можна використовувати пізніше для швидкого розгортання повністю попередньо налаштованих машин. Під час створення шаблону екземпляра переконайтеся, що ви встановили завантажувальний образ як образ, який ви щойно створили на попередньому кроці.

ВИСНОВКИ

Дослідження, проведене в рамках цієї роботи, підтверджує, що побудова масштабованих хмарних додатків є ключовим напрямком сучасної індустрії інформаційних технологій, який дозволяє організаціям ефективно відповідати на зростаючі вимоги користувачів і динамічні зміни ринкових умов. Аналіз основ хмарних технологій і сервісів показав, що чітке розуміння типів сервісів (SaaS, PaaS, IaaS), інструментів контейнеризації, таких як Docker, і технологій інфраструктури як сервісу (IaaS) є необхідною передумовою для створення гнучких і масштабованих систем. Docker забезпечує стандартизацію розгортання додатків, тоді як IaaS надає гнучкість у управлінні ресурсами, що є критично важливим для масштабування.

Розгляд процесів розгортання та масштабування хмарної інфраструктури підкреслив важливість правильного налаштування віртуальних серверів, служб баз даних і хмарних кластерів для забезпечення високої продуктивності та доступності. Створення віртуальних серверів і налаштування баз даних дозволяють швидко розгортати додатки, а формування хмарних кластерів забезпечує розподіл навантаження між вузлами. Використання кешування виявилося ефективним методом підвищення масштабованості, оскільки воно зменшує затримки та оптимізує обробку запитів, що є особливо важливим для додатків із високим трафіком.

Оптимізація та розповсюдження хмарних рішень, розглянуті в третьому розділі, показали, що впровадження кешування на рівні додатків, реплікація баз даних і використання мереж доставки контенту (CDN) значно підвищують продуктивність і доступність систем. Реплікація баз даних забезпечує відмовостійкість і швидкий доступ до даних у розподілених середовищах, тоді як CDN дозволяє оптимізувати доставку статичного контенту, зменшуючи навантаження на сервери. Використання Google Cloud Platform продемонструвало переваги хмарних платформ у забезпеченні гнучкості, автоматизації та інтеграції сучасних інструментів для розгортання масштабованих додатків.

Узагальнюючи, побудова масштабованих хмарних додатків вимагає комплексного підходу, який поєднує вибір відповідних технологій, оптимізацію інфраструктури та впровадження стратегій масштабування. Результати дослідження можуть бути використані розробниками та архітекторами для створення ефективних хмарних систем, здатних підтримувати високі навантаження та забезпечувати безперебійну роботу. Перспективи подальших досліджень включають інтеграцію штучного інтелекту для автоматичного управління хмарними ресурсами, використання серверлес-архітектур для додаткової оптимізації витрат і розробку гібридних хмарних рішень для підвищення безпеки та гнучкості. Ці напрямки сприятимуть створенню наступного покоління хмарних додатків, які відповідатимуть викликам майбутнього.

СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. Adler, B. *Building Scalable Applications with Docker*. — Manning Publications, 2020. — 350 с. — Режим доступу: https://www.manning.com/books/docker-in-action-second-edition

2. Amazon Web Services. AWS Architecture Center: Scalable Application Design. *aws.amazon.com*, 2023. — Режим доступу: https://aws.amazon.com/architecture

3. Amazon Web Services. AWS Well-Architected Framework: Scalability Pillar. *aws.amazon.com*, 2024. — Режим доступу: https://aws.amazon.com/architecture/well-architected

4. Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., ... & Zaharia, M. A view of cloud computing. *Communications of the ACM*, 2010, 53(4), 50–58. — Режим доступу: https://doi.org/10.1145/1721654.1721672

5. Bass, L., Clements, P., & Kazman, R. *Software Architecture in Practice*. — 4th ed. — Addison-Wesley, 2021. — 464 с. — Режим доступу: https://www.pearson.com/store/p/software-architecture-in-practice/P100001039246

6. Bass, L., M., K., & P, I. *Cloud Computing Architecture: Concepts and Design*. — Boston, MA: Addison-Wesley Professional, 2018. — 432 с.

7. Beyer, B., Jones, C., Petoff, J., & Murphy, N. R. *Site Reliability Engineering: How Google Runs Production Systems*. — O'Reilly Media, 2016. — 552 с. — Режим доступу: https://www.oreilly.com/library/view/site-reliability-engineering/9781491929117

8. Borisov, S., & Ivanyuk, I. AWS Well-Architected Framework: Designing Scalable Cloud Architectures. *dou.ua*, 2024, 15 с. — Режим доступу: https://dou.ua/lenta/articles/aws-well-architected-framework

9. Burns, B., Grant, B., Oppenheimer, D., et al. *Kubernetes Patterns: Reusable Elements for Designing Cloud-Native Applications*. — O'Reilly Media, 2019. — 266 с. — Режим доступу: https://www.oreilly.com/library/view/kubernetes-patterns/9781492050278

10. Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., & Brandic, I. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 2009, 25(6), 599–616. — Режим доступу: https://doi.org/10.1016/j.future.2008.12.001

11. Chappell, D. *Introducing the Azure Services Platform.* — San Francisco, CA: Chappell & Associates, 2008, 1–28. — Режим доступу: http://www.davidchappell.com

12. Erl, T., Cope, R., & Naserpour, A. *Cloud Computing Design Patterns.* — Prentice Hall, 2015. — 592 с. — Режим доступу: https://www.pearson.com/store/p/cloud-computing-design-patterns/P100000924200

13. Erl, T., Puttini, R., & Mahmood, Z. *Cloud Computing: Concepts, Technology & Architecture.* — Prentice Hall, 2013. — 528 с.

14. Fehily, C. *Google Cloud Platform in Action.* — Manning Publications, 2018. — 632 с. — Режим доступу: https://www.manning.com/books/google-cloud-platform-in-action

15. Fehling, C., Leymann, F., Retter, R., Schupeck, W., & Arbit, P. *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications.* — Springer, 2014. — 367 с. — Режим доступу: https://doi.org/10.1007/978-3-319-04849-9

16. Furht, B., & Escalante, A. *Handbook of Cloud Computing.* — Springer, 2010. — 634 с. — Режим доступу: https://www.springer.com/10.1007/978-3-319-90455-9

17. Gannon, D., Barga, R., & Sundaresan, N. Cloud-native applications. *IEEE Cloud Computing*, 2017, 4(5), 16–21. — Режим доступу: https://doi.org/10.1109/MCC.2017.4250939

18. GigaCloud. Хмарні технології: що це та які переваги надають людям та бізнесу. *gigacloud.ua*, 2023. — Режим доступу: https://gigacloud.ua

19. Google Cloud. Architecting with Google Kubernetes Engine. *cloud.google.com*, 2024. — Режим доступу: https://cloud.google.com/kubernetes-engine/docs

20. Google Cloud. Google Cloud Architecture Framework: Building Scalable Applications. *cloud.google.com*, 2023. — Режим доступу: https://cloud.google.com/architecture/framework

21. Hightower, K., Burns, B., & Beda, J. *Kubernetes: Up and Running: Dive into the Future of Infrastructure.* — Sebastopol, CA: O'Reilly Media, 2022. — 328 с.

22. Kavis, M. J. *Architecting the Cloud: Design Decisions for Cloud Computing Service Models (SaaS, PaaS, and IaaS).* — Wiley, 2014. — 224 с.

23. Kleppmann, M. *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems.* — O'Reilly Media, 2017. — 616 с. — Режим доступу: https://www.oreilly.com/library/view/designing-data-intensive-applications/9781491903063

24. Krutz, R. L., & Vines, R. D. *Cloud Security: A Comprehensive Guide to Secure Cloud Computing.* — Wiley, 2010. — 384 с.

25. Marinescu, D. C. *Cloud Computing: Theory and Practice.* — Morgan Kaufmann, 2022. — 672 с. — Режим доступу: https://doi.org/10.1016/B978-0-323-85277-7.00001-0

26. Mell, P., & Grance, T. The NIST definition of cloud computing. *National Institute of Standards and Technology*, 2011, 53(6), 1–7. — Режим доступу: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

27. Microsoft Azure. Azure Architecture Center: Scalability and Performance. *azure.microsoft.com*, 2024. — Режим доступу: https://learn.microsoft.com/en-us/azure/architecture

28. Morris, K. *Infrastructure as Code: Dynamic Systems for the Cloud Age.* — 2nd ed. O'Reilly Media, 2021. — 428 с. — Режим доступу: https://www.oreilly.com/library/view/infrastructure-as-code/9781098114688

29. Netwave. Побудова ефективних хмарних інфраструктур. *netwave.ua*, 2024. — Режим доступу: https://netwave.ua

БІБЛІОГРАФІЧНА ДОВІДКА

Тема дипломної роботи бакалавра: " Побудова масштабованих хмарних додатків "

Обсяг пояснювальної записки: 60 аркушів

Дата закінчення дипломної роботи 10 червня 2025р.

Підпис студента _____