Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
(повне найменування вищого навчального закладу)
Факультет комп'ютерно-інформаційних систем і програмної інженерії
(назва факультету)
Кафедра комп'ютерних систем та мереж
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістра
(освітній ступінь))

на тему: **Технології автоматизованого розгортання електронних магазинів на основі OpenCart з використанням DevOps- практик**

Виконав: студент (ка) 6 курсу, групи СІм-61
спеціальності 123 «Комп'ютерна інженерія»
(шифр і назва спеціальності)

	(підпис)	Барна І.М. (прізвище та ініціали)
Керівник	(підпис)	Луцків А.М. (прізвище та ініціали)
Нормоконтроль	(підпис)	Луцик Н.С. (прізвище та ініціали)
Завідувач кафедри	(підпис)	Осухівська Г.М. (прізвище та ініціали)
Рецензент	(підпис)	Цуприк Г.Б. (прізвище та ініціали)

Тернопіль
2022

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
(повне найменування вищого навчального закладу)
Факультет комп'ютерно-інформаційних систем і програмної інженерії
Кафедра комп'ютерних систем та мереж

ЗАТВЕРДЖУЮ
Завідувач кафедри *Осухівська Г.М.*
«_____» 2022 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ

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

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

студенту *Барні Івану Михайловичу*
(прізвище, ім'я, по-батькові)

1. Тема проекту (роботи) *Технології автоматизованого розгортання електронних магазинів на основі OpenCart з використанням DevOps- практик*
Керівник проекту (роботи) *Луцків Андрій Мирославович, к.т.н., доц.*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом ректора від «*06*» *грудня* 2022 року №*4/7-986*
2. Термін подання студентом завершеної роботи
3. Вихідні дані до роботи *Принципи підходу IaaS, DevOps практики автоматизації процесу розгортання ПЗ, особливості платформи OpenCart*
4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1. Аналіз підходів до організації та використання хмарних сервісів 2. Побудова рішень на основі DevOps практик з використанням Kubernetes cluster. 3. Реалізація DevOps практик на основі OpenStack, LAMP та OpenCart 4. Охорона праці та безпека в надзвичайних ситуаціях. Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Актуальність і мета дослідження. 2. Задачі дослідження, , об'єкт і предмет, наукова новизна і практична цінність дослідження. 3. Характеристики хмарних сервісів. 4. Задачі і принципи IaaS.5.Архітектура рішення на основі Kubernetes cluster. 6. Архітектура рішення на основі OpenStack. 7. Формальне представлення DevOps практик. 8. Висновки

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці та	*Осухівська Г.М.*		
безпека в надзвичайних			
ситуаціях			

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	*Аналіз підходів до організації та використання хмарних*		*виконано*
	сервісів		
2.	*Побудова рішень на основі DevOps практик з використан-*		*виконано*
	ням Kubernetes cluster.		
3.	*3. Реалізація DevOps практик на основі OpenStack, LAMP*		*виконано*
	та OpenCart		
4.	*Охорона праці та безпека в надзвичайних ситуаціях*		*виконано*
5.	*Оформлення пояснювальної записки*		*виконано*
6.	*Оформлення графічного матеріалу*		*виконано*
7.	*Попередній захист кваліфікаційної роботи магістра*		*виконано*
8.	*Захист кваліфікаційної роботи магістра*		

Студент _____ *Барна І.М.*
(підпис) (прізвище та ініціали)

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

АНОТАЦІЯ

Тема кваліфікаційної роботи: “Технології автоматизованого розгортання електронних магазинів на основі OpenCart з використанням DevOps- практик” // Кваліфікаційна робота // Барна Іван Михайлович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп’ютерно-інформаційних систем та програмної інженерії, група СІм-61 // Тернопіль, 2022 // с. – 79, рис. – 28, табл. – 3, аркушів А1 – 8, додат. – 1, бібліогр. – 26.

Ключові слова: розгортання, автоматизація, електронний магазин, DevOps, практики.

Мета роботи полягає у дослідженні і побудові DevOps практик для підвищення ефективності адміністрування та розгортання електронних магазинів на основі OpenCart.

Проведено аналіз особливостей та принципів організації електронних магазинів, визначено переваги і недоліки технології хмарних сервісів на основі PaaS та SaaS, що передбачає надання частини готових сервісів з невеликими налаштуваннями з боку користувачів і водночас накладає обмеження на тип використовуваної інфраструктури, а також супроводжується підвищеним тарифним планом у порівнянні з IaaS.

Запропоновано рішення застосування DevOps практик, що передбачають використання кластеру Kubernetes та розгортання інфраструктури за допомогою технологічного рішення Terraform і Helm Chart.

Побудовано альтернативне рішення щодо автоматизованого розгортання електронних магазинів, що передбачає використання платформи OpenStack з розгортанням на ній віртуальних машин з гостьовими операційними системами, комплексом серверного програмного забезпечення LAMP та безпосередньо самого OpenCart.

ABSTRACT

The theme of the thesis: " Technologies for the automated deployment of electronic stores based on OpenCart using DevOps practices" /Master thesis / Barna Ivan Mychaylovych / Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and software engineering, group CIm -61 // Ternopil, 2022// p. - 79, fig. – 28, table. – 3, Posters A1 – 8, Add – 1, Ref. – 26.

Keywords: deployment, automation, e-shop, DevOps, practices.

The purpose of the work is to research and build DevOps practices to increase the efficiency of administration and deployment of electronic stores based on OpenCart.

An analysis of the features and principles of the organization of electronic stores was carried out, the advantages and disadvantages of cloud services technology based on PaaS and SaaS were determined, which involves the provision of a part of ready-made services with small settings on the part of users and at the same time imposes restrictions on the type of infrastructure used, and is also accompanied by an increased tariff plan in compared to IaaS.

A solution for applying DevOps practices involving the use of a Kubernetes cluster and infrastructure deployment using the Terraform and Helm Chart technological solution is proposed.

An alternative solution for the automated deployment of electronic stores has been built, which involves the use of the OpenStack platform with the deployment of virtual machines with guest operating systems, a complex of LAMP server software, and directly OpenCart itself.

ЗМІСТ

ВСТУП

Актуальність теми. Сьогодні розробка сучасних комп’ютерних та інформаційних систем супроводжується виконанням різнопланових та функціонально різних процесів. Однак ефективність процесу практичної реалізації складових систем напряму залежить від використання засобів та інструментів їх автоматизації. Особливо доцільно застосовувати засоби і практики автоматизації при розгортанні інфраструктури та забезпеченні надійності її функціонування. DevOps практики поєднують у собі «розробку» та «операції». Це стосується набору практик, які забезпечують ефективну співпрацю та комунікацію між командами розробки та IT-операцій протягом усього процесу розробки продукту. Ці практики спрямовані на вдосконалення процесів розробки, тестування, розгортання, управління та обслуговування програмного забезпечення шляхом їх оптимізації, прискорення та підвищення передбачуваності, масштабованості, гнучкості та безпеки. Для цього використовуються автоматизація процесів, безперервна інтеграція (CI), безперервна доставка (CD), безперервне розгортання, інфраструктура як код, мікросервіси, керування конфігурацією та інші методи DevOps.

Крім того, DevOps представляє собою техніку розробки, яка підтримує Agile-методологію та підтримує багато її характеристик, таких як безперервне вдосконалення, швидка доставка та плавне спілкування. DevOps і Agile чудово підходять один одному.

Реалізація DevOps залежить від кожної компанії залежно від її цілей, процесів і навіть корпоративної культури. Однак можна визначити низку основних принципів DevOps, яких дотримується більшість команд. Перевагами DevOps є сприяння створенню середовища для співпраці через спілкування, взаємну довіру, обмін навичками та ідеями при вирішенні проблем.

Встановлення культури наскрізної підзвітності, за якої вся команда несе відповідальність за результати та немає «вказівних пальців» між експертами «Dev» і «Ops». Зосередженість на постійному вдосконаленні на основі інформації клієнтів

і розвитку технологій з метою оптимізації якості продукту, вартості та швидкості доставки. За можливості використовується автоматизація для оптимізації й прискорення процесів розробки й розгортання, а також підвищення ефективності і надійності кінцевого продукту. Забезпечення стратегії, орієнтованої на клієнта, із швидкими циклами зворотного зв'язку для задоволення мінливих потреб клієнтів, беручи уроки з помилок і створюючи середовище, де їх можна перетворити на нові можливості. Технології електронної комерції увійшли у повсякденне життя і чим далі, тим ефективніше їх використання. Однак при комплексному підході до побудови платформ електронної комерції виникає багато рутинних операцій, пов'язаних з ручним налаштуванням і супроводом інфраструктури. Тому формування практик автоматизованого розгортання та адміністрування електронних магазинів забезпечать можливість знизити витрати часу та коштів на їх впровадження і супровід. Оскільки напрям DevOps на сучасному етапі перебуває у фазі становлення та інтенсивного розвитку, то актуальними є задачі щодо побудови практик адміністрування та розгортання електронних магазинів на основі OpenCart.

Мета роботи полягає у дослідженні і побудові DevOps практик для підвищення ефективності адміністрування та розгортання електронних магазинів на основі OpenCart.

Об'єктом дослідження є процеси побудови та налаштування інфраструктури для ефективного функціонування електронних магазинів.

Предметом дослідження є методи, інструменти і DevOps практики для розгортання і конфігурування інфраструктури для функціонування електронних магазинів на базі платформи OpenCart.

Для досягнення мети роботи магістра необхідно розв'язати **наступні задачі:**

– виконати аналіз існуючих підходів та методів автоматизації процесів розробки та супроводу комп'ютерних систем, зокрема її програмних складових;

– дослідити особливості інфраструктури та її конфігурації для ефективного функціонування електронних магазинів на базі OpenCart;

– обґрунтувати вибір засобів для автоматизованого розгортання

інфраструктури для роботи з електронними магазинами;

– розробити та формалізувати DevOps практики для адміністрування та розгортання електронних магазинів на основі OpenCart.

– практично реалізувати налаштування конфігурації інфраструктури з автоматичним її розгортанням для функціонування електронних магазинів;

– забезпечити можливість гнучкого налаштування електронних магазинів під потреби кінцевого користувача.

Методи дослідження: При вирішенні задач кваліфікаційної роботи застосовувались такі методи і засоби: аналіз та узагальнення – при проведенні аналізу існуючих підходів і практик автоматизації процесів розгортання програмного забезпечення у хмарних сервісах; теорії множин – при математичному описі рішень і DevOps практик автоматизованого розгортання електронних; магазинів; проектування та програмування – при побудові архітектури рішень на основі DevOps практик та їхній реалізації; експеримент та вимірювання – при практичному застосуванні технологій LAMP та OpenCart.

Наукова новизна отриманих результатів. Наукова новизна результатів дослідження полягає в наступному:

– уперше формалізовано структуру та залежності інфраструктури і програмного забезпечення за допомогою представлення DevOps практик у вигляді множин та відповідних компонентів, що дало змогу забезпечити ілюстрацію вкладеності рівнів з одночасним відображенням їх структури і в практичному сенсі визначило порядок застосування DevOps практик;

– уперше запропоновано рішення застосування DevOps практик, що передбачають використання кластеру Kubernetes та розгортання інфраструктури за допомогою технологічного рішення Terraform і Helm Chart і дають змогу гнучкого налаштування додатків і сервісів користувача при імплементації магазинів електронної комерції на основі OpenCart.

Практичне значення одержаних результатів. Впровадження запропонованих DevOps практик, які становлять рішення автоматизованого розгортання електронних магазинів, дозволило забезпечити ефективність процесу налаштування та відповідність критеріям надійності і функціональності електронних магазинів на основі OpenCart

Публікації. Результати кваліфікаційної роботи апробовані на X науково-технічній конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі, системи та технології» (8-9 грудня 2022 року) як тези конференцій.

1. Луцків А.М., Барна І.М. Т Особливості задач і функцій DevOps фахівців. Матеріали X науково-технічної конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі, системи та технології» (8-9 грудня 2022 року). Тернопіль: ТНТУ. 2022. С. 65.

2. Луцків А.М., Барна І.М. Аналіз сервісно-орієнтованої архітектури в процесі застосування DevOps практик. Матеріали X науково-технічної конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі, системи та технології» (8-9 грудня 2022 року). Тернопіль: ТНТУ. 2022. С. 66.

Структура роботи. Кваліфікаційна робота містить розрахунково-пояснювальну записку та графічний матеріал. До складу записки входить вступу, 4 розділи, загальні висновки, список використаних джерел і додатки. Обсяг роботи: розрахунково-пояснювальної записки – 85 арк. формату А4, графічна частина – 8 аркушів формату А1.

РОЗДІЛ 1
АНАЛІЗ ПІДХОДІВ ДО ОРГАНІЗАЦІЇ ТА ВИКОРИСТАННЯ ХМАРНИХ СЕРВІСІВ

1.1. Аналіз способів організації та ведення електронної комерції

У сучасну технологічну епоху онлайн-бізнес все більше покладається на хмарні сервіси і ресурси для полегшення своєї щоденної діяльності. Деякі компанії та окремі особи використовують інфраструктуру як послугу (IaaS) для формування власних фреймворків, а деякі можуть використовувати платформу як послугу (PaaS) для створення програмного забезпечення. Ще одна категорія бізнесу використовує програмне забезпечення як послугу (SaaS), щоб отримати вигоду від повного програмного рішення. У результаті хмарні обчислення надзвичайно розвинулись за останні кілька років.

Світовий дохід від хмарних сервісів зріс з 90 мільярдів доларів у 2016 році до 409 мільярдів доларів у 2021 році. Хмарні рішення допомагають збирати, зберігати та обробляти величезну кількість даних перед тим, як їх представити клієнту. Крім того, це надає низку спеціалізованих послуг, щоб допомогти бізнесу збільшити оборот, а також впровадити прості та ефективні рішення для щоденних бізнес-потреб.

Більше компаній і магазинів електронної комерції відмовляються від IT-рішень, які розміщені на «землі», і починають включати послуги, які забезпечують IT-інфраструктуру, платформи та програмне забезпечення для їхніх щоденних операційних потреб.

Хмарні обчислення зберігають дані та інформацію в мережі Інтернет, а не на одному жорсткому диску в офісі. Це акт передачі обчислювальних послуг через Інтернет, широко відомий як «хмара». Хмара швидка, безпечна та доступна за ціною, що робить її ідеальною для бізнесу порівняно зі застарілими системами зберігання. Крім того, дані знаходяться в Інтернеті, тобто доступ до них завжди доступний звідусіль, включаючи мобільні пристрої.

Використання хмари означає, що підприємствам не потрібно купувати апаратне забезпечення чи наймати технічних спеціалістів для встановлення та обслуговування інфраструктури. Натомість вони можуть придбати простір для зберігання, мережу та сервери в хмарі та продовжувати масштабувати свої операції відповідно до своїх бізнес-потреб. Як правило, хмарні компанії використовують термін «як послуга», щоб підказати, який тип продукту вони пропонують.

Кожна хмарна служба забезпечує те, що підприємствам потрібно менше локальної інфраструктури для встановлення та керування, залишаючи більше часу для зосередження на збільшенні продажів та утриманні клієнтів. Більшість хмарних служб є модульними, тобто можна включити одну частину або всі сегменти інфраструктури управління третьою стороною.

Існує три основні типи варіантів хмарних сервісів як послуг, які охоплюють різні сфери управління, необхідні користувачеві. Їх часто називають хмарними службами або послугами хмарних обчислень і забезпечують різні аспекти IT-інфраструктури – залежно від використовуваної служби (рис. 1.1).

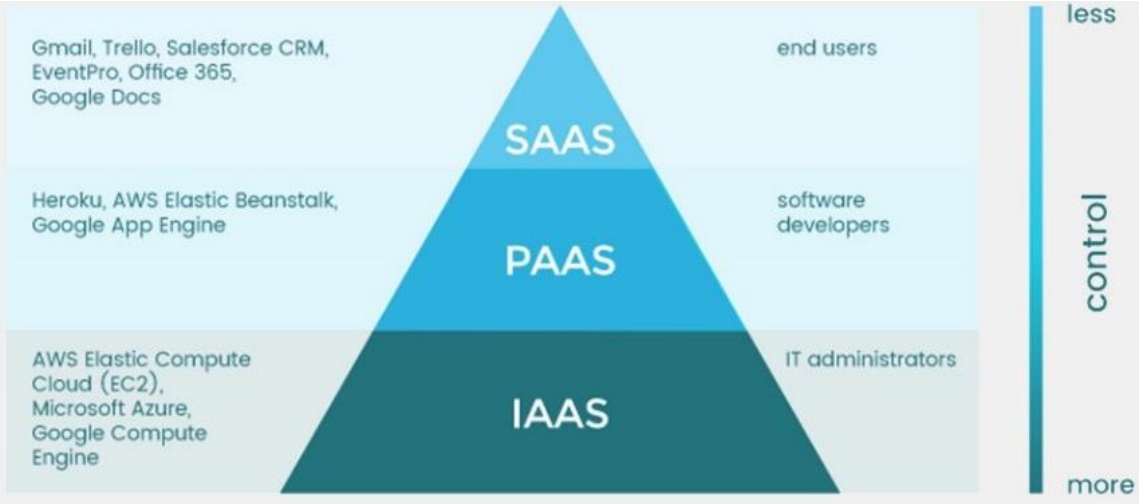

Рис. 1.1. Види хмарних сервісів як послуг

Три основні послуги хмарних обчислень: інфраструктура як послуга (IaaS), платформа як послуга (PaaS) і програмне забезпечення як послуга (SaaS).

Суть IaaS – постачальник послуг надає програмне забезпечення та програми через Інтернет. Вони також забезпечують необхідне сховище, мережу,

віртуалізацію та сервери. Його можна розглядати як серверну IT-інфраструктуру для запуску програм і робочих навантажень у хмарі.

PaaS – платформа як послуга: постачальник послуг пропонує доступ до програмних інструментів, які дозволяють розробникам створювати та доставляти програми на замовлення.

PaaS надає базову інфраструктуру, з якої розробник може відштовхуватися при імплементації своїх рішень. Засоби розробки апаратного та програмного забезпечення входять до більшості продуктів PaaS.

SaaS – програмне забезпечення як послуга: постачальник надає клієнтам платний доступ до хмарного сховища, мереж, серверів та інших обчислювальних ресурсів. Це дозволяє клієнтам орендувати повні програмні рішення з мінімальними вхідними даними з боку клієнта. Кожна з цих служб відображає, як хмара використовується для бізнесу, вимагаючи від клієнта різних рівнів введення.

На рис. 1.2 показано відповідальність користувачів послуг та вендорів.

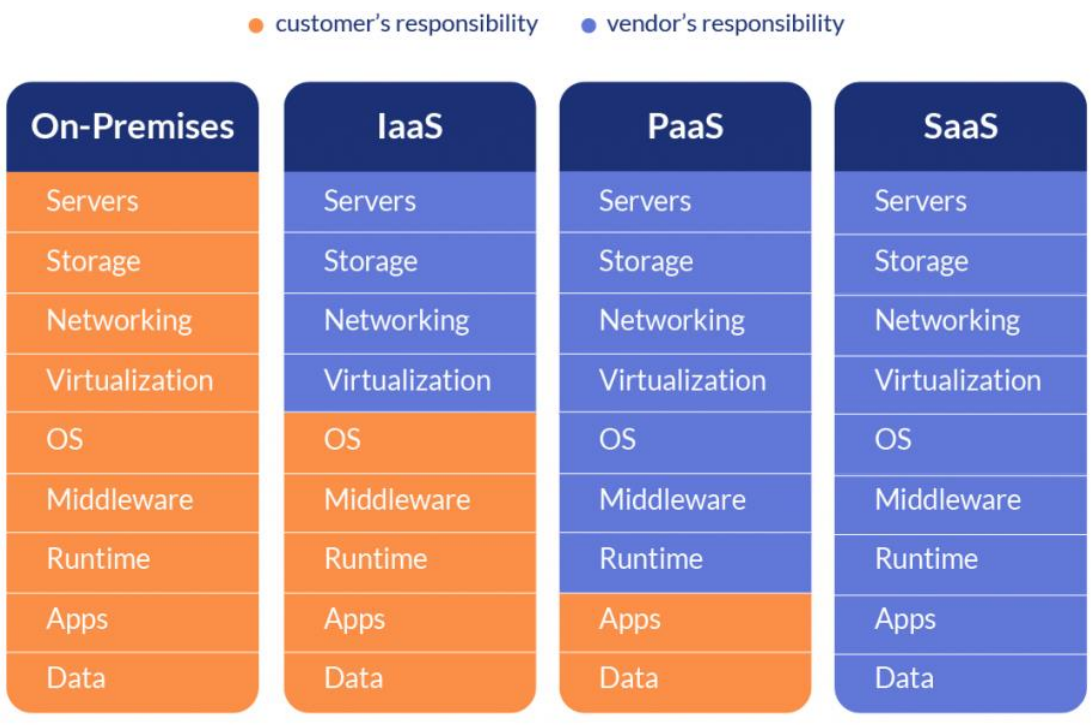

Рис. 1.2. Розподіл відповідальності при використанні хмарних сервісів як послуг

IaaS вимагає найбільшого управління від клієнта, тоді як SaaS вимагає найменшого. Однак важливо пам’ятати, що IaaS, PaaS і SaaS не є взаємовиключними. У результаті компанії, як правило, використовують більше одного, а великі компанії часто використовують усі три рішення для задоволення своїх потреб.

1.2. Інфраструктура як послуга при організації електронної комерції

Продукти IaaS для хмарних обчислень, зазвичай, вимагають від клієнта більше управління та досвіду і не завжди ідеально підходять для стартапів електронної комерції.

Інфраструктура як послуга – це модель хмарних обчислень, яка зазвичай є послугою з оплатою за використання. IaaS можна розглядати як послугу на вимогу, де сторонній продукт IaaS надаватиме такі послуги, як:

– технологія віртуалізації;
– сервери;
– зберігання даних;
– мережа.

Доступ за вимогою до хмарної обчислювальної інфраструктури можна налаштувати та використовувати так само, як обладнання використовувалося б на місці. Однак різниця полягає в тому, що постачальник послуг керуватиме та обслуговуватиме апаратне забезпечення у своїх власних центрах обробки даних, залишаючи розробникам більше часу, щоб зосередитися на розробці необхідного програмного забезпечення.

Організації використовуватимуть свої платформи та програми в інфраструктурі постачальника IaaS і отримають до неї доступ через мережу Інтернет. У результаті, постачальник IaaS, зазвичай, може запропонувати кращий продукт, ніж може надати команда на місці.

У рамках SaaS клієнт/бізнес усе ще має керувати наступним:

– зберігання даних;

– час виконання;
– проміжне програмне забезпечення;
– операційні системи.

Чудова особливість IaaS полягає в тому, що користувачеві не потрібно підтримувати або оновлювати локальний центр обробки даних, оскільки всім цим займається постачальник. Таким чином, замість того, щоб купувати апаратне забезпечення, користувачі можуть платити за IaaS на вимогу та масштабувати його за допомогою свого бізнес-трафіку.

Користувач може отримати доступ до всієї інфраструктури та контролювати її за допомогою інтерфейсу прикладного програмування (API) або інформаційної панелі, наданої постачальником послуг.

Клієнти IaaS можуть вибирати між віртуальними машинами, розміщеними на спільному фізичному апаратному забезпеченні, або використовувати окремі сервери на спеціальному обладнанні. Виділені сервери працюють краще, що призводить до швидшого часу завантаження та вищих коефіцієнтів конверсії.

IaaS був оригінальним наданням «як послуга», причому більшість великих постачальників хмарних послуг починали з певної форми IaaS.

Однак з часом підприємства електронної комерції відійшли від постачальників IaaS, оскільки вони все ще повинні розробляти власне програмне забезпечення замість того, щоб зосереджуватися на збільшенні продажів.

Є кілька сценаріїв, у яких компанії використовують IaaS. Чудовим способом використання IaaS є створення середовищ розробки та тестування до того, як вони будуть передані клієнтам. Високопродуктивні обчислення (HPC) і великі дані є яскравими прикладами того, як компанії використовують IaaS.

Постачальник IaaS пропонує економічно ефективну та масштабовану інфраструктуру для забезпечення високої потужності обробки, необхідної для аналізу великих даних. IaaS ідеально підходить для використання, коли компанія не володіє власним центром обробки даних.

Крім того, стартапи можуть використовувати IaaS, оскільки їм не потрібно вкладати капітал в IT-інфраструктуру, але все одно можуть отримати доступ до

технологій корпоративного рівня. На рис. 1.3 показано загальний принцип організації інфрастуктури як коду.

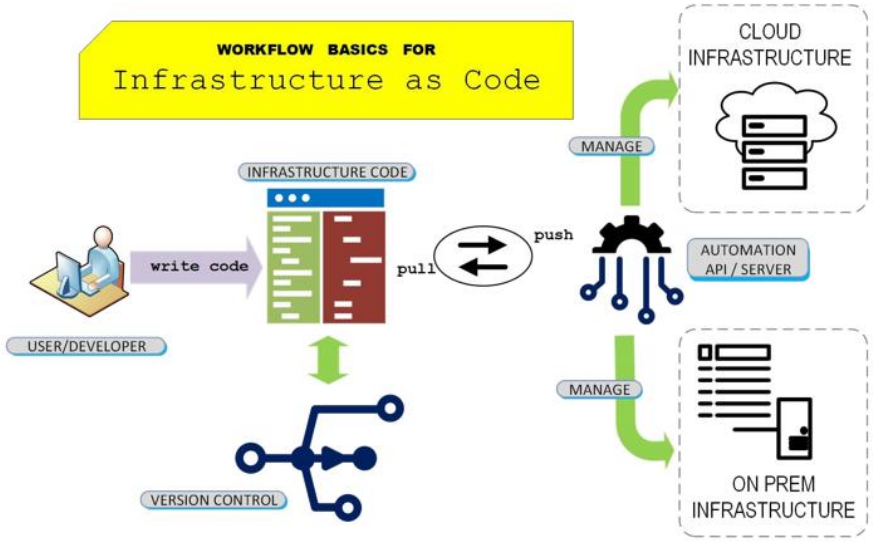

Рис. 1.3. Базовий сценарій використання IaaS

IaaS розроблено таким чином, щоб відповідати більшості бюджетів, оскільки хмарні обчислення є платними за використані ресурси, тобто клієнти можуть купувати ресурси на вимогу. Крім того, компанії, які перейшли з локальної інфраструктури, але хочуть зберегти існуюче програмне забезпечення, як правило, тяжіють до IaaS. Це пояснюється тим, що вони можуть використовувати IT-інфраструктуру постачальника IaaS, продовжуючи підтримувати власне програмне забезпечення, з яким вони знайомі. Як правило, як бізнес електронної комерції, не буде використовувати IaaS, оскільки ця модель не може підтримувати зростаючі вимоги до ефективності клієнтів.

Нарешті, IaaS часто використовується в ситуаціях аварійного відновлення. Оскільки все зберігається в хмарі на кількох серверах, ймовірність того, що стихійне лихо вплине на декілька серверів, дуже низька.

Є багато переваг використання інфраструктури як послуги. Наприклад, IaaS дозволяє купувати лише необхідні компоненти та масштабувати їх відповідно до змін бізнесу. Він також видаляє зайві та невикористані ресурси, які запускаються лише за потреби.

Ще одна перевага IaaS полягає в тому, що це, як правило, послуга з оплатою за використання ресурсів та на вимогу, що полегшує підприємствам керування кошторисними витратами для свого бюджету.

Крім того, інфраструктура масштабується залежно від потреб. Наприклад, коли компанія відчуває стрибки трафіку, інфраструктуру можна відповідно збільшуватись та зменшуватись. Оскільки клієнтам не потрібно купувати апаратне забезпечення, використання IaaS забезпечує підприємствам низькі накладні витрати з невеликими витратами на обслуговування, що робить його дуже доступним. У той же час організації все ще мають повний контроль над своєю IT-інфраструктурою та можуть вносити зміни, коли це необхідно.

Як правило, постачальники IaaS матимуть вищий рівень комплексної безпеки, ніж безпека на місці. Відповідно до закону, великі центри обробки даних мають забезпечити безпеку даних, які вони зберігають, і багато постачальників IaaS наймають експертів із безпеки, щоб гарантувати її. Оскільки дані зберігаються в хмарі на кількох серверах, немає єдиної точки відмови. Використання IaaS надає клієнтам новітні технології. Ці хмарні служби часто встановлюють новітні технології швидше, ніж їх можна встановити на місці. Клієнту не потрібно бути фахівцем у сфері IT, а також не потрібно довіряти зовнішньому IT-підряднику.

Поряд із перевагами є деякі недоліки, про які слід пам’ятати під час використання IaaS. Одним із недоліків є те, що компаніям все ще потрібно переконатися, що їхні програми та операційні системи працюють належним чином і дотримуються протоколів безпеки. Це означає, що IT-фахівці все ще можуть знадобитися в штаті. Іншим недоліком є те, що усунення несправностей є складнішим, оскільки клієнти не мають повної видимості серверної інфраструктури. Крім того, оскільки третя сторона контролює інфраструктуру, користувачі не мають варіантів пом’якшення наслідків, якщо центр обробки даних зазнає збою.

У пікові сезони щомісячні витрати можуть стати набагато вищими, ніж очікувалося, для компаній, оскільки їм доводиться збільшувати свої обчислювальні ресурси через постачальника IaaS.

Крім того, міграція із застарілих систем може бути складною, оскільки їх потрібно перевірити на сумісність, а для нової системи може знадобитися внутрішнє навчання персоналу. IaaS – це найбільш практичний сервіс хмарних обчислень для клієнта. Постачальник IaaS пропонує лише сервери та API, а бізнес керує всім іншим.

IaaS Amazon Web Service (AWS) є, мабуть, найпоширенішим провайдером IaaS. Він використовується для хмарних обчислень на вимогу та купується на основі періодичної підписки.

AWS допомагає компаніям зберігати дані та надавати контент своїм клієнтам. Сьогодні багато веб-сайтів і блогів в Інтернеті покладаються на доставку AWS. За даними Kinsta, AWS має 5,8% частки веб-хостингу в Інтернеті. Він також має приблизно 34% ринку серед хмарних провайдерів у 100 000 найкращих веб-сайтів (рис. 1.4.).

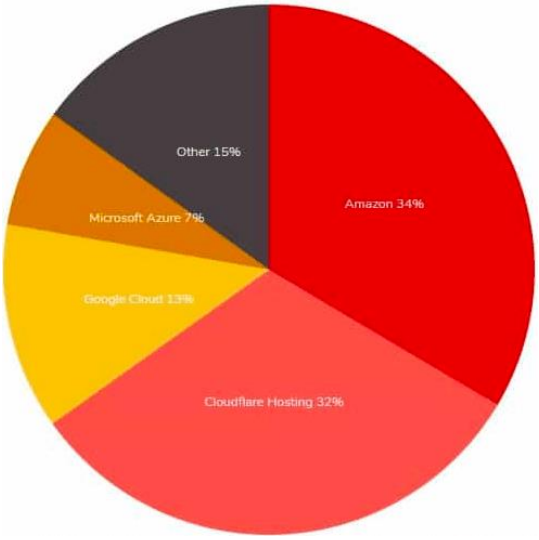

Рис. 1.4. Статистика IaaS компаніями-надавачами послуг

Інші реальні приклади включають Microsoft Windows Azure і Google Compute Engine. Microsoft Azure дозволяє створювати, тестувати та керувати програмами через мережу центру обробки даних Microsoft.

З іншого боку, Google Compute Engine дозволяє клієнтам створювати та запускати віртуальні машини в інфраструктурі Google.

З боку електронної комерції Adobe Commerce (Magento) є одним із найбільших постачальників IaaS. Продавці часто платять Magento за ліцензування програмного забезпечення та використовують стороннього постачальника, наприклад Rackspace, для веб-хостингу.

1.3. PaaS у системах електронної комерції

PaaS, зазвичай, надає розробникам інструменти та середовище розробки, необхідні для створення спеціальних програм. Платформа як послуга – це модель хмарного сервісу, яка надає розробникам структуру хмарних обчислень, інструменти та середовище, необхідні для створення власних масштабованих програм. Як правило, постачальник послуг пропонує:

- час виконання;
- проміжне програмне забезпечення;
- OS;
- віртуалізація;
- сервери;
- зберігання;
- мережа.

Платформу, як послугу, часто розглядають як «зменшену» версію IaaS, оскільки це ще один крок далі від керування локальним центром обробки даних.

У PaaS постачальник розміщує програмне та апаратне забезпечення у власній системі та надає це як платформу для кінцевого користувача через підключення до Інтернету. Однак клієнт все одно повинен управляти додатками і даними.

Зазвичай, клієнт отримує доступ до PaaS через графічний інтерфейс користувача (GUI), не потребуючи завантаження чи локальної інсталяції. Інструменти, що надаються продуктами PaaS, дозволяють розробникам створювати, запускати, керувати та доставляти власні програми без створення та підтримки необхідної платформи. Натомість розробнику потрібно просто написати код, не турбуючись про оновлення програмного забезпечення чи керування

обладнанням. Розробники часто використовують його для створення та налаштування веб-додатків. Крім того, платформа зазвичай надає сховище та набір готових інструментів, які допомагають розробникам створювати та тестувати свої програми.

PaaS має кілька різних варіантів використання. Одна з них – це розробка та керування API, що полегшує командам розробку прикладних програмних інтерфейсів для обміну даними між програмами. PaaS також підтримує широкий спектр мов програмування, таких як Java і Python, що дозволяє розробникам використовувати його для розробки програм інтернету речей (IoT).

Компанії часто використовують платформу для створення спеціальних програм, не витрачаючи значних коштів на інфраструктуру. Для аналогії, якщо потрібно влаштувати концерт, то виконавці не будують арену самі. Натомість береться в оренду відомий заклад і організатори зосереджуються на продажу квитків.

PaaS регулярно використовується для створення індивідуальних рішень SaaS. Переваги використання платформи як послуги надає економічно ефективне рішення, яке дозволяє клієнтам створювати, тестувати, розгортати, запускати та оновлювати програми ефективніше, ніж вони могли б керувати всім на місці.

У результаті використання PaaS це часто призводить до того, що команди розробників швидше постачають свій продукт на ринок. Крім того, PaaS, зазвичай, дозволяють отримати доступ до найновіших технологій та інструментів у своєму стеку програм. Потім кінцеві користувачі можуть тестувати нові операційні системи та інструменти, не вкладаючи власних коштів. Як і IaaS, платформи PaaS, є масштабованими, і за потреби групи розробників можуть придбати додаткові потужності для створення.

Платформа як послуга, зазвичай, допомагає організаціям зосередитися на розробці, не турбуючись про базову інфраструктуру. Постачальник хмарних обчислень зазвичай керує безпекою, операційною системою, серверним програмним забезпеченням і резервним копіюванням. Розробнику просто потрібно зосередитися на кодуванні.

Команди можуть співпрацювати через PaaS, якщо вони працюють віддалено, а висока продуктивність завдяки використанню PaaS може призвести до вищих коефіцієнтів конверсії.

Знову ж таки, є деякі недоліки, про які слід знати при використанні платформи як послуги. Наприклад, у разі збою в роботі постачальника послуг PaaS будь-які програми, створені на платформі, будуть недоступні.

Іншим недоліком є те, що розробники контролюють лише код програми, а не інфраструктуру, яка використовується для її створення. Тому лише малі та середні компанії зазвичай використовують PaaS. Великі компанії, такі як Coca-Cola та Nike, як правило, будують всю інфраструктуру на хмарних сервісах. На жаль, деякі PaaS мають складні умови надання послуг. Наприклад, деякі можуть накладати обмеження на налаштування, які можуть забезпечити лише розробники, і вони також можуть зіткнутися з проблемами під час інтеграції нових програм, які несумісні.

Google App Engine є одним із найбільших постачальників PaaS. Це дозволяє розробникам створювати та розміщувати веб-орієнтовані програми в хмарі Google. Інші приклади PaaS включають Heroku, який дозволяє розробникам створювати додатки, орієнтовані на споживача, Apprenda, який дозволяє розробникам розміщувати цілі портфоліо додатків, і AWS Elastic Beanstalk, який забезпечує можливість розробникам швидко розгортати додаток у хмарі AWS. Adobe Commerce (Magento) також є PaaS для електронної комерції. Це дозволяє рітейлерам об’єднувати свій хостинг як частину пакету. Він забезпечує швидкий розвиток, підвищену безпеку та високу масштабованість для магазинів електронної комерції.

1.4. SaaS в електронній комерції

Модель SaaS є найповнішою формою хмарних сервісів. Вона надає повне онлайн-програмне забезпечення клієнту, яким керує постачальник SaaS. Його також називають службами хмарних додатків. Більшість людей використовували

SaaS, навіть якщо вони цього не усвідомлювали, з такими програмами, як Gmail, Spotify і Netflix.

Програмне забезпечення як послуга зазвичай надається клієнтам за моделлю передплати. Програмне забезпечення завжди розміщується в хмарі та готове до використання для клієнтів (рис. 1.5).

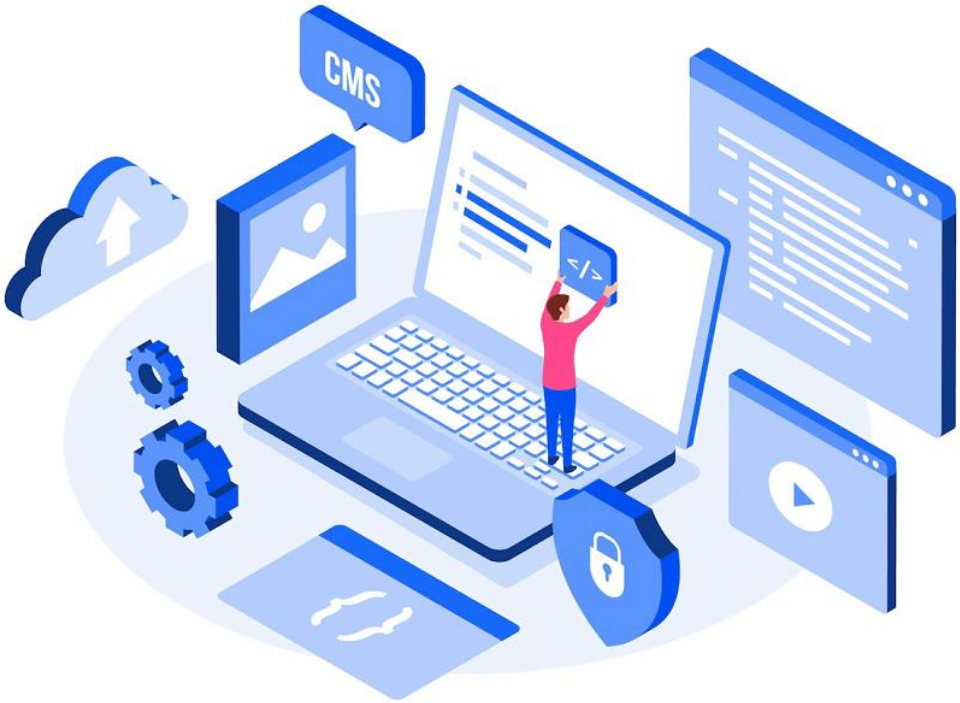

Рис. 1.5. SaaS

Постачальник SaaS опрацьовує всі оновлення програмного забезпечення та виправлення помилок. Крім того, вони забезпечують:

– додатки;
– дані;
– час виконання;
– проміжне програмне забезпечення;
– OS;
– віртуалізація;
– сервери;
– зберігання;

– мережа.

Кінцевому користувачеві не потрібно керувати будь-чим, що стосується продукту, зі свого боку; це все опрацьовується постачальником SaaS. Користувач просто підключається до програми через інформаційну панель постачальника SaaS або через API через Інтернет. Крім того, деякі провайдери SaaS пропонують доступ через веб-браузер або мобільні додатки.

Платформи SaaS чудово підходять, коли потрібно, щоб програма працювала безперебійно з невеликим управлінням з боку клієнта. SaaS ідеально підходить для малого бізнесу або стартапів, які не можуть розробляти власні програми.

SaaS – це найшвидше та найпростіше рішення для електронної комерції та короткострокових проектів. Це дозволяє клієнтам створювати ефективні сайти електронної комерції без розуміння мереж і хостингу. Їм потрібно лише зосередитися на веб-дизайні бізнес-операцій, продажах і створенні контенту. Також чудово використовувати SaaS для періодично використовуваного програмного забезпечення, наприклад податкового та бухгалтерського програмного забезпечення.

SaaS зазвичай перекладає всю інфраструктуру та керування програмами на постачальника SaaS. У результаті користувачеві не потрібно встановлювати будь-які програми. Замість цього вони просто створюють обліковий запис і сплачують комісію, щоб почати використовувати сервіси. SaaS можна застосовувати для співпраці між співробітниками компанії за допомогою індивідуального входу, щоб диференціювати рівні доступу.

Оскільки, SaaS заснований на підписці, клієнти точно знають, скільки вони повинні платити щомісяця. Крім того, вони можуть додати більше користувачів і сховища даних у міру збільшення свого бізнесу. Існує мінімальний ризик, оскільки більшість продуктів SaaS пропонують безкоштовний пробний період, щоб дозволити клієнтам перевірити програмне забезпечення на відповідність їхнім потребам.

На жаль, клієнти не можуть контролювати хмарну інфраструктуру продукту SaaS. Таким чином, якщо у постачальника SaaS виникне збій, то це станеться і з

додатком. Користувачі не можуть контролювати жодну інтеграцію, оскільки всі вони обробляються постачальником. Іноді продукти SaaS несумісні з апаратним і програмним забезпеченням, яке вже використовується в бізнесі.

Є дуже багато прикладів SaaS. Усі програми Google, як-от документи Google, Gmail і таблиці Google, є прикладами програм SaaS. Іншими прикладами є JIRA, Dropbox і DocuSign. Але, звичайно, Netflix є, мабуть, найпопулярнішим SaaS. В електронній комерції Shopify є найпоширенішою платформою SaaS. За даними Backlinko, близько 1,75 мільйона торговців продають на Shopify сьогодні в більш ніж 175 різних країнах. Інші приклади електронної комерції SaaS включають Hubspot, BigCommerce і 3Dcart.

1.5. Порівняння Iaas, Paas та SaaS при застосуванні в електронній комерції

Як згадувалося раніше, SaaS, IaaS і PaaS не є взаємовиключними. Багато організацій використовують більше ніж одну з цих хмарних служб. Вибір правильного типу хмарної служби залежить від вимог клієнта та кваліфікації персоналу компанії. Якщо компанія не має технічного досвіду в IT, nj краще уникати IaaS. Крім того, компанії не знадобиться PaaS, якщо у команді немає розробників.

Як правило, SaaS є найкращим варіантом для більшості рішень електронної комерції, оскільки він зменшує витрати та підвищує ефективність. Чудовий спосіб вибрати, які хмарні сервіси використовувати – подумати про те, чим повинен керувати користувач.

Клієнту нема чим керувати за допомогою SaaS, оскільки постачальник SaaS обробляє все.

Найсуттєвіша відмінність між IaaS і PaaS полягає в тому, що перший призначений для забезпечення контролю, а другий пропонує гнучкість.

IaaS дає адміністраторам більше контролю над своїми операційними системами, тоді як PaaS надає розробникам усі інструменти, необхідні для створення програми.

Наприклад, продукт IaaS допоможе розмістити веб-сайт і його програми, якщо потрібно створити новий веб-сайт. Щоб додати більше користувацьких функцій до веб-сайту, потрібно буде використовувати продукт PaaS для його розробки та встановлення.

У SaaS програмне забезпечення повністю контролюється іншою компанією. З іншого боку, продукти PaaS є основою для створення продуктів на вершині мережі платформи. Наприклад, якщо необхідно створити мобільний додаток для бізнесу, то варто використовувати PaaS для розробки програмного забезпечення та використання інфраструктури постачальника для його запуску. Після випуску, програмне забезпечення вважатиметься SaaS для користувачів. Тому SaaS краще підходить компаніям, які шукають готову до використання послугу, не потребуючи досвіду та часу для самостійної розробки.

PaaS найкраще підходить для компаній, які прагнуть створити нове рішення у своїй існуючій мережі.

Загалом тип послуги хмарних сервісів, повністю залежить від потреб бізнесу. Як правило, якщо клієнтам просто надаються операції електронної комерції, то краще використовувати сервіси SaaS. Ці програми полегшать бездоганну інтеграцію у поточну бізнес-модель і забезпечать ефективний спосіб здійснення продажів.

Рішення PaaS безпосередньо підходять для компаній, у яких є кілька розробників для створення та розгортання програм.

Модель IaaS може забезпечити базову інфраструктуру, необхідну для ведення бізнесу, але все одно потрібно буде створювати, моніторити та керувати встановленням програмного забезпечення. Ці моделі хмарних сервісів допомагають компаніям відійти від локальної інфраструктури до більш масштабованої моделі хмарних обчислень.

1.6. Висновки до розділу

До основних результатів, отриманих у даному розділі, належать:

1. Проведено аналіз особливостей та принципів організації електронних магазинів у результаті якого встановлено, що широкої популярності та ефективності набуває тенденція щодо їх розгортання у хмарних сервісах, що дає змогу знизити витрати на утримання власної інфраструктури, підвищити надійність їх функціонування і доступності, а також зосередити більше уваги на веденні електронного бізнесу.

2. Досліджено принципи організації хмарних сервісів і встановлено, що найбільш трудомістким і водночас найбільш гнучким є підхід, який передбачає використання DevOps практик при організації інфраструктури як коду, що дає змогу забезпечити автоматизоване розгортання додатків та організувати гнучкість їх налаштування, що відповідає потребам замовників.

3. Проаналізовано та визначено переваги і недоліки технології хмарних сервісів на основі PaaS та SaaS, що передбачає надання частини готових сервісів з невеликими налаштуваннями з боку користувачів і водночас накладає обмеження на тип використовуваної інфраструктури, а також супроводжується підвищеним тарифним планом у порівнянні з IaaS.

4. Проведено порівняльний аналіз форм хмарних сервісів у вигляді IaaS, PaaS та SaaS у результаті якого встановлено, що при організації електронної комерції для користувачів з відсутніми навиками DevOps найбільш ефективно використовувати необхідне програмне забезпечення як сервіс, а у двох інших випадках – IaaS або PaaS. Це дало змогу обґрунтувати доцільність та необхідність реалізації рішення на основі IaaS з використанням DevOps практик для подальшого створення та розгортання інфраструктури електронного магазину з можливим повторним використанням.

РОЗДІЛ 2
ПОБУДОВА РІШЕННЯ НА ОСНОВІ DEV OPS ПРАКТИК З ВИКОРИСТАННЯМ KUBERNETES CLUSTER

2.1. Аналіз та обґрунтування вибору архітектури і DevOps практик при реалізації хмарних рішень

Майже кожне IT-рішення сьогодні позначається терміном хмарні обчислення або просто хмара Хмарні обчислення, або хмара, є метафорою пропозиції та використання IT-ресурсів. IT-ресурси в хмарі не бачать користувачі безпосередньо; між ними є шари абстракції. Рівень абстракції, який пропонує хмара, варіюється: від пропозиції віртуальних машин (VM) до надання програмного забезпечення як послуги (SaaS) на основі складних розподілених систем.

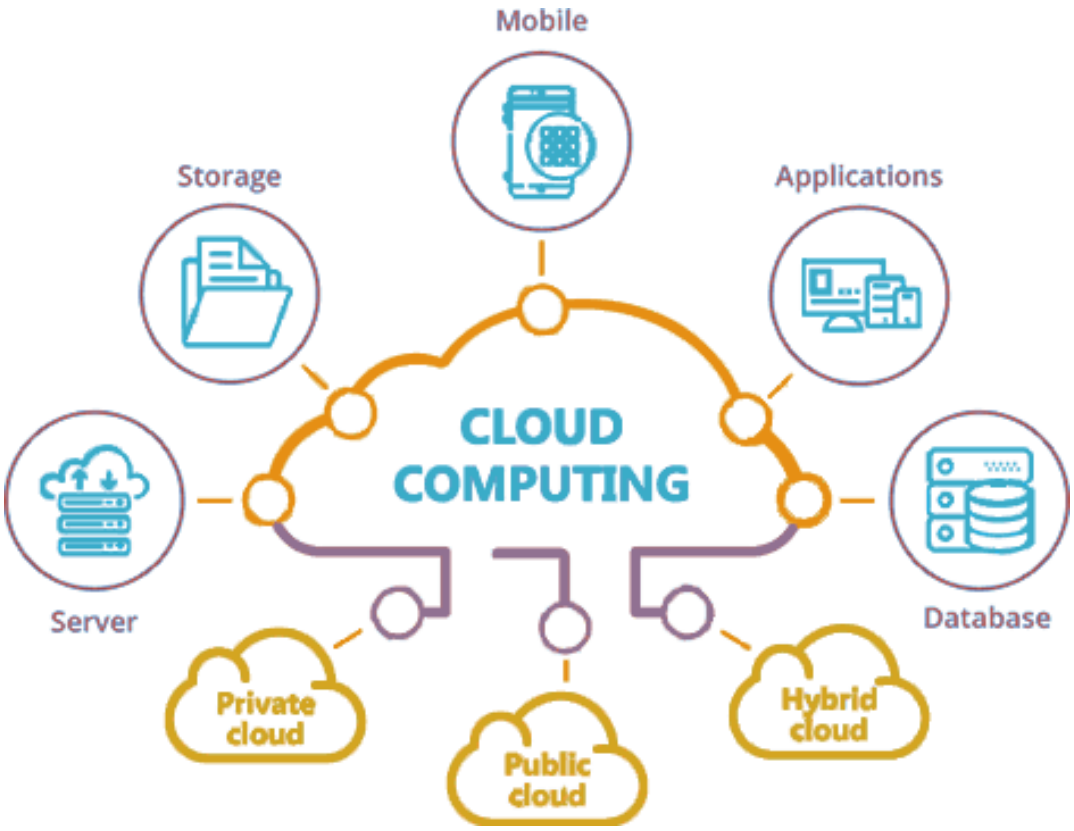

Рис. 2.1. Можливості хмарних рішень

Створення будь-якого нового проекту завжди супроводжується низкою проблем, які необхідно розв’язати, а його успіх – це запитання щодо його реалізації, підтримки та масштабування в майбутньому. Електронна комерція сьогодні розвивається досить бурхливо і вимагає нових рішень для автоматизованого розгортання ресурсів, їх балансування, підтримки і супроводу. Хід проекту безпосередньо залежить від будівельних блоків та архітектури, які використовуються при реалізації проекту. Існує кілька архітектур, які широко використовуються при розробці нового проекту, наприклад монолітна, сервісно-орієнтована архітектура (SOA) та архітектура мікросервісів.

2.1.1. Монолітна архітектура

Монолітна архітектура – це концепція, коли деяке програмне забезпечення розробляється як єдиний блок (рис. 2.2). Усі функціональні можливості, функції та модулі розробляються, інтегруються та розгортаються у вигляді єдиного цілого, тобто це свого ролу платформа, що містить інтерфейс користувача, бізнес-логіку та рівень бази даних на одній сторінці. Монолітний додаток має єдину спільну базу даних для кожної функції та кожного атрибуту, що об’єднані в одній програмі як уніфікована модель.

Монолітна архітектура зручна і легко реалізується невеликими командами під невеликі проекти. Багато стартапів і малих проектів розробляються з використанням монолітної архітектури. Це добре для проекту, коли модулі взаємозалежні та взаємопов’язані.

Монолітна архітектура володіє як мінусами, так і перевагами. До переваг можна віднести простоту розгортання, оскільки в кінцевому підсумку вона розгортається як єдиний міцний об’єкт. Монолітна архітектура має менш поширені утиліти, тому, якщо реалізується ця архітектура, то виникає мало наскрізних проблем. Монолітні програми мають кращу та швидшу продуктивність, оскільки у них мінімальні або іноді зовсім відсутні запити для API, модулі близькі один до одного та існують в одному цілому, що робить їх досить швидкими.

Найбільшим недоліком монолітної архітектури є відмовостійкість. Оскільки моноліт працює як єдиний блок, і якщо є проблема в простій елементарній функції, то вся програма перестане працювати, оскільки програма завжди розгортається як єдиний блок. Коли додаток стає все більшим і більшим, він ускладнюється і ним важче керувати, що може бути спричинено внесенням однієї зміни, яка у результаті змушує протестувати додаток повністю, а це завжди займає багато часу.

Монолітна архітектура знижує гнучкість, оскільки невелике оновлення та розробка функцій завжди вимагають повного розгортання. Оновлення, як і технологічні оновлення, є проблемою в монолітній архітектурі, і здебільшого уникають такого рішення.

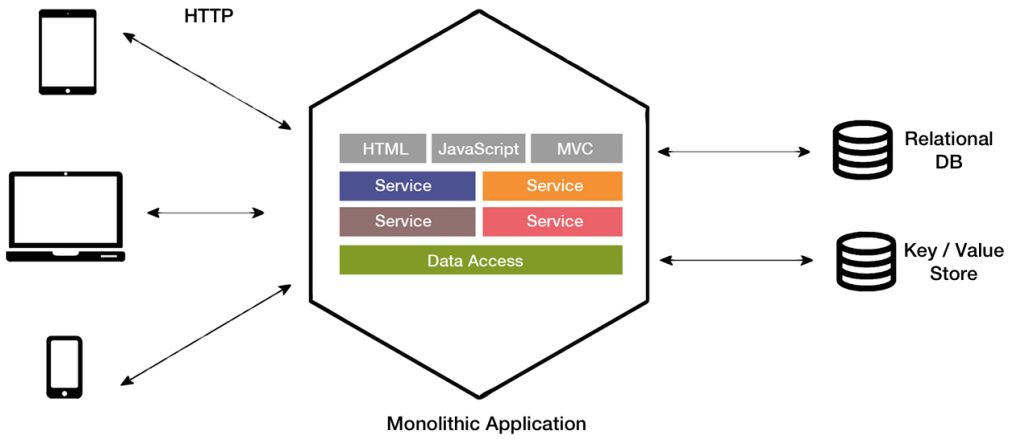

Рис. 2.2. Монолітна архітектура

2.1.2. Сервісно-орієнтована архітектура

Сервісно-орієнтована архітектура деякий час також відома як централізовано орієнтована представляє собою архітектуру, де кілька послуг, які також називаються користувацькими агентами, використовують або створюють взаємодію з централізованою системою, щоб зробити традиційну монолітну менш важкою і слабко пов’язаною.

Архітектура, орієнтована на обслуговування, спеціально розроблена для обміну даними між системами. Кожен сервіс, що забезпечує функціональність на рівні абстракції, розглядається як чорна скринька і є автономним з метою уникнення зайвих витрат на кожну нову розробку (рис. 2.3).

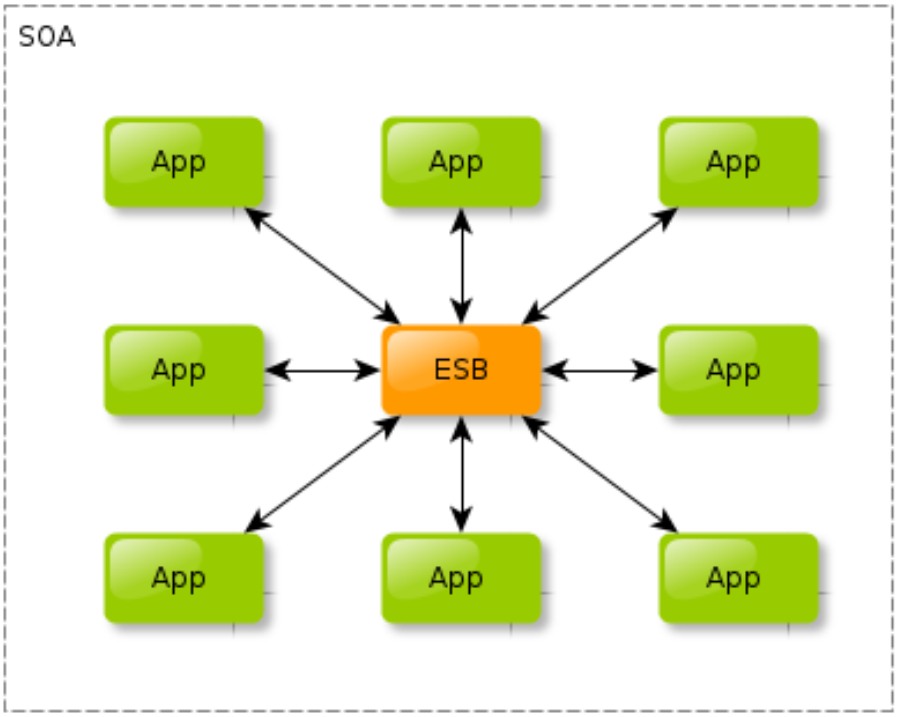

Рис. 2.3. Сервісно-орієнтована архітектура

Сервіс абстрактний і може бути розроблений за будь-якою технологією. Зв'язок між ними може здійснюватися за допомогою центральної точки підключення, яка називається Enterprise Service Bus (ESB). ESB піклується про всі сервіси та допомагає їм взаємодіяти один з одним.

SOA має свої плюси і мінуси, оскільки SOA надає можливість повторного використання всіх незалежних служб, наприклад, якщо потрібно створити модуль аутентифікації входу, замість того, щоб робити його з самого початку, можна використовувати службу аутентифікації facebook/google в SOA. Якимось чином гетерогенні послуги SOA роблять систему відмовостійкістю.

Якщо одна служба не працює, система не перестане працювати. Найбільшою перевагою SOA є можливість повторного використання сервісу. Якщо сервіс створений, то його завжди можна використовувати знову у нових розробках.

Одним із недоліків SOA є витрати на ESB, зміна служби також спричиняє її зміну. ESB є SPOF, якщо він не працює або в разі його пошкодження – вся програма не буде працювати. Кількість послуг також збільшують витрати та час відгуку. SOA добре підходить для додатків на рівні підприємства.

2.1.3. Мікросервісна архітектура

Архітектура мікросервісів також розглядається як децентралізована архітектура, багато в чому схожа на SOA, але не пов'язана з будь-яким центральним сервісом. Мікросервісна архітектура – це архітектура, яка побудована з використанням автономних служб, які спілкуються один з одним за допомогою REST.

Як і мікросервіс SOA, вона не будується на такому сервісі, який може знову використовувати будь-який інший проект, а замість цього його фокус полягає в тому, щоб зробити кожну функціональність окремою як:

– сервіс, який має бути мікровмісним і автономним;
– додатком;
– бізнесом;
– інфраструктурою та корпоративним рівнем.

Кожен підрозділ служби має свою базу даних. Усі ці мікросервіси потім приєднуються до відповідного інтерфейсу користувача, щоб відображатися як робоча програма. З інтерфейсу користувача викликаються створені мікросервіси через інші виклики.

В архітектурі мікросервісів (рис. 2.4) є кілька концепцій, щоб виявити сервіс і щоб інші служби могли спілкуватися один з одним, оскільки відомо, що кожен компонент або невелика функціональність розроблені як мікросервіс, і вони повинні спілкуватися один з одним, щоб працювати належним чином.

Кожен мікросервіс має свою модель бази даних, яка є незалежною, і вони зв’язуються з іншими службами, коли це необхідно.

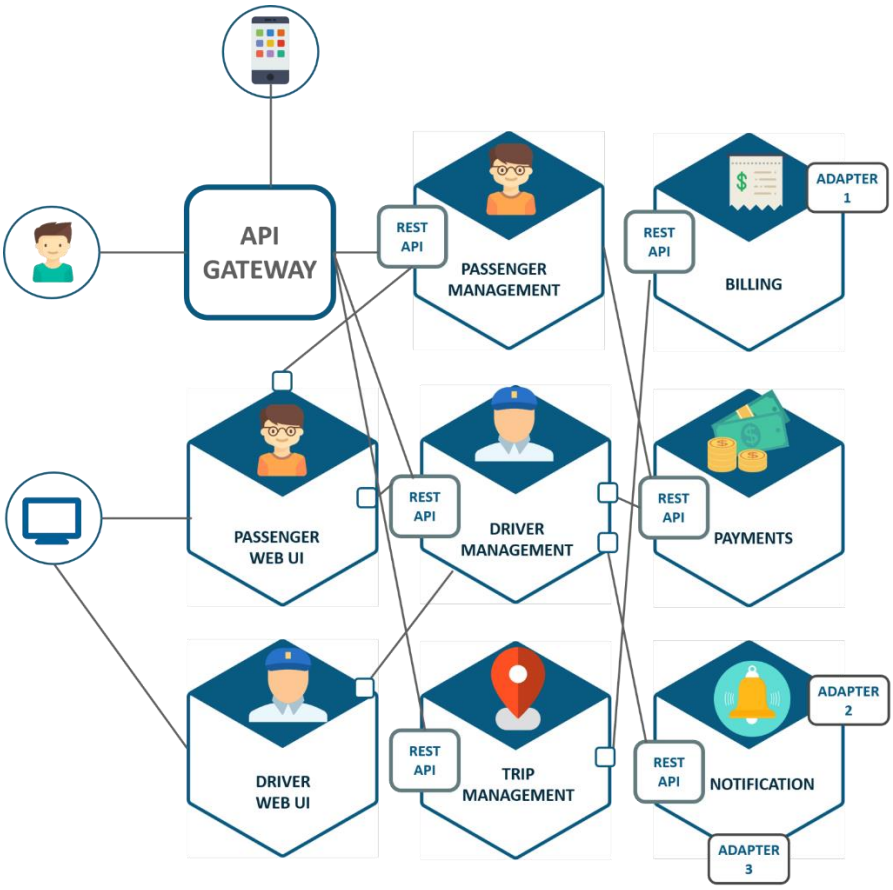

Рис. 2.4. Мікросервісна архітектура

При використанні архітектури мікросервісів, кожен сервіс незалежний і якимось чином їм потрібно спілкуватися один з одним. Для цього використовується ряд додаткових служб і сервісів.

Реєстр служб – усі мікросервіси реєструються на одному сервері, який називається реєстром служб, їх може бути декілька

Сервіс Discovery – коли мікросервісам потрібно спілкуватися один з одним, вони зв’язуються з сервером імен або сервером виявлення, щоб одержати такі дані, як IP-адресу і порт служби, ця концепція називається виявленням служби.

Крім перерахованого вище, як і для будь-якої архітектури сервісу, потрібно реалізувати кілька речей, таких як балансування навантаження, відмовостійкість, шлюзи API, перевірки справності тощо.

Однією з найбільших переваг є простота розгортання, розробки та тестування. До переваг мікроскрвісної архітектури належить:

– швидкість і безперервний розвиток;
– відсутність залежності під час роботи з індивідуальним сервісом;
– Висока масштабованість і відмовостійкість.

Деякі мінуси полягають у повільній роботі системи, оскільки центральна служба викликає багато служб. Як і SOA, мікросервіси також хороші для проектів на рівні підприємства.

Провівши детальний аналіз архітектур при побудові проектів, доцільно при організації електронних магазинів на основі OpenCart орієнтуватись на DevOps практики, які забезпечать реалізацію мікросервісної архітектури при автоматизованому розгортанні необхідних компонентів архітектури та інфраструктури у хмарі.

2.2. Побудова архітектури рішень для автоматизованого розгортання електронних магазинів з використання DevOps практик

Аналізуючи сучасні DevOps практики, які дозволяють автоматизувати процес розгортання програмного забезпечення незалежно від його типу і застосування, найбільш популярними є технологія Kubernetes, а також технології віртуалізації з подальшим розгортанням Guest OS і налаштуванням системного і прикладного програмного забезпечення.

У роботі, для автоматизованого розгортання електронних магазинів з використанням OpenCart пропонується два шляхи імплементації, в основі яких лежить:

– Kubernetes cluster (рис. 2.5);
– OpenStack (рис. 2.6).

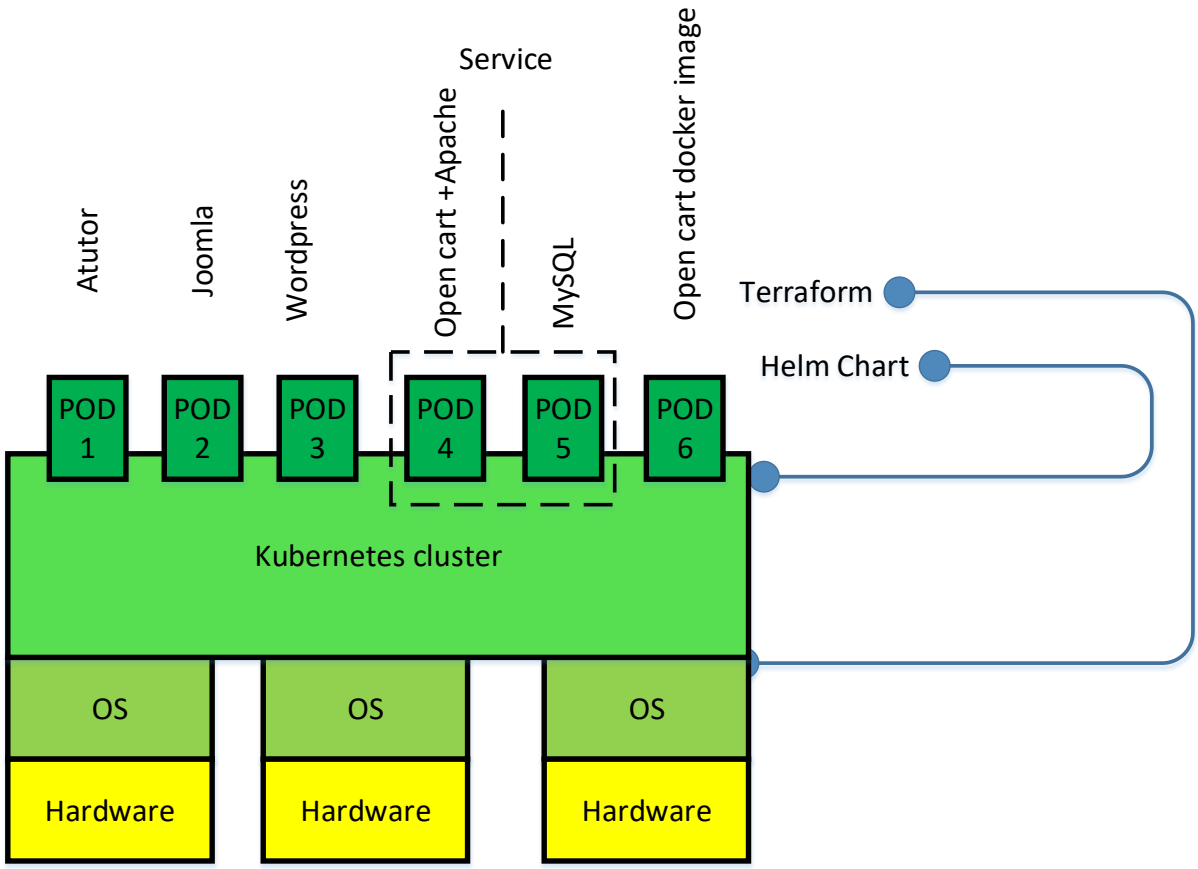

Рис. 2.5. Рішення автоматизації розгортання електронних магазинів на основі Kubernetes cluster

Як видно з рис. 2.5 архітектура рішення автоматизованого розгортання електронних магазинів включає:

– апаратне забезпечення провайдера хмарного сервісу (Hardware);
– операційна система хмарного сервісу (OS);
– Kubernetes cluster;
– POD.

Для розгортання кластеру пропонується використати технологічне рішення, що дає змогу описати конфігурацію кластера Kubernetes cluster за допомогою Terraform.

Налаштування конфігурації програм або прикладних сервісів пропонується виконати за допомогою Helm Chart.

Проведемо більш детальний аналіз основних властивостей та функціональності DevOps практик при автоматизації процесу розгортання електронних рішень.

2.2.1. Особливості використання та функції Kubernetes

Kubernetes – це портативна, розширювана платформа з відкритим вихідним кодом для керування контейнерними робочими навантаженнями та службами, яка полегшує як декларативне налаштування, так і автоматизацію. Він має велику екосистему, що швидко розвивається. Сервіси, підтримка та інструменти Kubernetes широко доступні. Назва Kubernetes походить з грецької, що означає керманич або пілот. K8s як абревіатура є результатом підрахунку восьми літер між «K» і «s». Google відкрила проект Kubernetes у 2014 році і він поєднує понад 15 років досвіду Google у створенні робочих навантажень у масштабі з найкращими у своєму роді ідеями та практиками спільноти.

Контейнери – це хороший спосіб об’єднати та запускати додатки. У виробничому середовищі потрібно керувати контейнерами, які запускають програми, і гарантувати, що немає простоїв. Наприклад, якщо контейнер виходить з ладу, необхідно запустити інший контейнер.

Kubernetes надає структуру для стійкої роботи розподілених систем. Він піклується про масштабування та відновлення після відмови додатку, надає шаблони розгортання тощо. Наприклад: Kubernetes може легко керувати розгортанням Canary для системи.

Виявлення служб і балансування навантаження – Kubernetes може розкривати контейнер за допомогою імені DNS або власної IP-адреси. Якщо трафік до контейнера великий, Kubernetes може балансувати навантаження та розподіляти мережевий трафік, щоб розгортання було стабільним.

Оркестровка сховищ Kubernetes дозволяє автоматично монтувати систему зберігання на вибір користувача послуги, наприклад локальні сховища, публічні хмарні постачальники тощо.

Автоматичне розгортання та відкочування – можна описати бажаний стан власних розгорнутих контейнерів за допомогою Kubernetes, і він може змінити фактичний стан на бажаний із контрольованою швидкістю. Наприклад, ви можете автоматизувати

Kubernetes може використовуватися для створення нових контейнерів розгортання, видалення існуючих контейнерів і адаптації всіх їхніх ресурсів до нового контейнера.

Автоматичне пакування в контейнер – Kubernetes кластер вузлів, які він може використовувати для виконання контейнерних завдань. DevOps інженер повідомляє Kubernetes, скільки процесорів та пам’яті (RAM) потрібно кожному контейнеру. Kubernetes може розмістити контейнери на вузлах так, щоб якнайкраще використовувати виділені ресурси.

Kubernetes із самовідновленням перезапускає контейнери, які вийшли з ладу, замінює контейнери, видаляє контейнери, які не реагують на визначену користувачем перевірку працездатності, і не рекламує їх клієнтам, доки вони не будуть готові до обслуговування.

Управління безпекою і конфігурацією – Kubernetes дозволяє зберігати та керувати конфіденційною інформацією, як-от паролі, маркери OAuth і ключі SSH. Можна розгортати та оновлювати ключі і конфігурацію програми без перебудови образів контейнерів і без розкриття ключів у конфігурації стеку.

Kubernetes не є традиційною комплексною системою PaaS (платформа як послуга). Оскільки Kubernetes працює на рівні контейнера, а не на апаратному рівні, він надає деякі загальноприйнятні функції, загальні для пропозицій PaaS, як-от розгортання, масштабування, балансування навантаження, і дозволяє користувачам інтегрувати свої рішення для журналювання, моніторингу та оповіщення.

Однак Kubernetes не є монолітним, і ці рішення за замовчуванням є необов’язковими та підключаються. Kubernetes надає блоки для створення платформ розробників, але зберігає вибір користувача та гнучкість там, де це важливо.

Kubernetes не обмежує типи підтримуваних програм, прагне підтримувати надзвичайно різноманітні робочі навантаження, включаючи робочі навантаження без збереження стану, із збереженням стану та навантаження обробки даних. Якщо програма може працювати в контейнері, вона повинна чудово працювати і в Kubernetes.

Дана платформа не розгортає вихідний код і не створює додаток. Робочі процеси безперервної інтеграції, доставки та розгортання (CI/CD) визначаються організаційною культурою та перевагами, а також технічними вимогами.

Kubernetes не надає служб прикладного рівня, таких як проміжне програмне забезпечення (наприклад, шини повідомлень), інфраструктури обробки даних (наприклад, Spark), бази даних (наприклад, MySQL), кеші чи кластерні системи зберігання (наприклад, Ceph) як вбудовані служби. Такі компоненти можуть працювати на Kubernetes і/або до них можуть отримати доступ програми, що працюють на цій платформі, через переносні механізми, такі як Open Service Broker.

Kubernetes не надає рішень щодо реєстрації, моніторингу чи оповіщення. Вона забезпечує деякі інтеграції як доказ концепції та механізми для збору та експорту показників.

Платформа надає декларативний API, на який можуть націлюватися довільні форми декларативних специфікацій. Не надає та не приймає жодних комплексних систем конфігурації, обслуговування, керування чи самовідновлення машини.

Крім того, Kubernetes — це не просто оркестрація. Фактично, це усуває потребу у ній. Технічне визначення оркестрації – це виконання визначеного робочого процесу: спочатку виконується A, потім B, потім C. На відміну від цього, Kubernetes складається з набору незалежних процесів керування, які постійно переміщують поточний стан у заданий бажаний стан. Не має значення, як добираються від A до C. Централізований контроль також не потрібен. Це призводить до створення системи, яка є простішою у використанні та більш потужною, надійною, стійкою та надійною.

2.2.2. Pods у кластері Kubernete

Pods (Поди) – це найменші розгорнуті обчислювальні одиниці, які можна створювати та керувати ними в Kubernetes. «Стручок» (наприклад, гороховий стручок) — це група з одного або кількох контейнерів зі спільним сховищем і мережевими ресурсами, а також специфікацією того, як запускати контейнери.

Вміст модуля завжди розміщується разом і за розкладом, а також працює в спільному контексті. Pod моделює специфічний для програми «логічний хост»: він містить один або більше контейнерів програми, які відносно тісно пов’язані. У нехмарних контекстах програми, які виконуються на одній фізичній або віртуальній машині, аналогічні хмарним програмам, які виконуються на тому самому логічному хості.

Окрім контейнерів додатків, Pod може містити контейнери ініціалізації, які запускаються під час запуску Pod.

Незважаючи на те, що Kubernetes підтримує більше середовищ виконання контейнерів, ніж просто Docker, проте останній є найпоширенішим середовищем виконання, і це допомагає описувати Pods, використовуючи певну термінологію Docker.

Спільний контекст Pod — це набір просторів імен Linux, контрольних груп і потенційно інших аспектів ізоляції – тих самих речей, які ізолюють контейнер. У контексті модуля окремі програми можуть застосовувати додаткові субізоляції. Pod схожий на набір контейнерів зі спільними просторами імен і спільними томами файлової системи. Нижче наведено приклад Pod, який складається з контейнера, на якому запущено образ nginx:1.14.2 (рис. 2.6).

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

Рис. 2.6. Приклад контейнера з образом nginx

2.2.3. Функціональні можливості Terraform

HashiCorp Terraform – це інструмент інфраструктури як коду, який дозволяє визначати як хмарні, так і локальні ресурси в зрозумілих для людини конфігураційних файлах, які можна забезпечувати версійність, повторно використовувати та ділитися ними. Потім можна використовувати послідовний робочий процес для надання та керування всією інфраструктурою протягом її життєвого циклу. Terraform може керувати компонентами низького рівня, такими як обчислення, сховище та мережеві ресурси, а також компонентами високого рівня, такими як записи DNS і функції SaaS.

Terraform створює та керує ресурсами на хмарних платформах та інших службах через свої інтерфейси прикладного програмування (API). Постачальники дозволяють Terraform працювати практично з будь-якою платформою чи службою з доступним API (рис. 2.7).

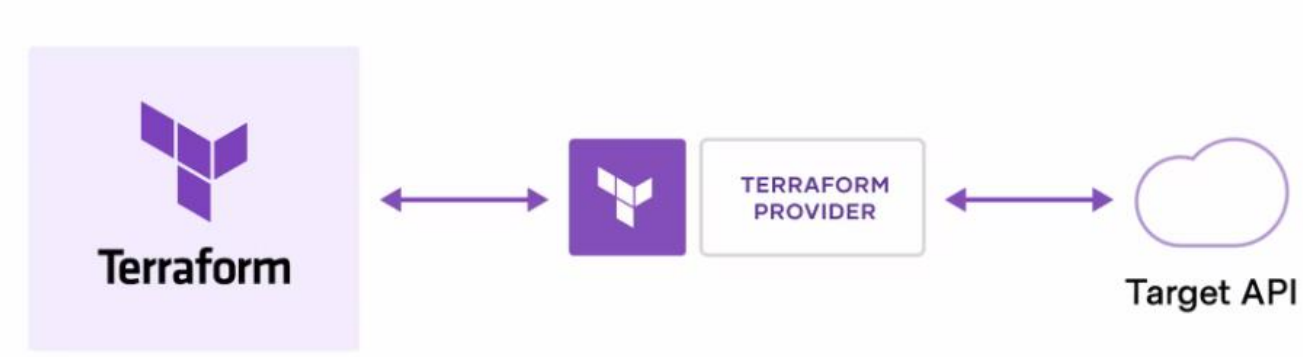

Рис. 2.7. Схема функціонування Terraform

HashiCorp і спільнота Terraform вже написали тисячі провайдерів для управління багатьма різними типами ресурсів і послуг. Їх можна знайти у реєстрі Terraform, зокрема Amazon Web Services (AWS), Kubernetes та інші. Основний робочий процес Terraform складається з трьох етапів (рис. 2.8.).

Запис – визначає ресурси, які можуть бути в кількох хмарних постачальників і послуг. Наприклад, можна створити конфігурацію для розгортання програми на віртуальних машинах у мережі віртуальної приватної хмари (VPC) із групами безпеки та балансувальником навантаження.

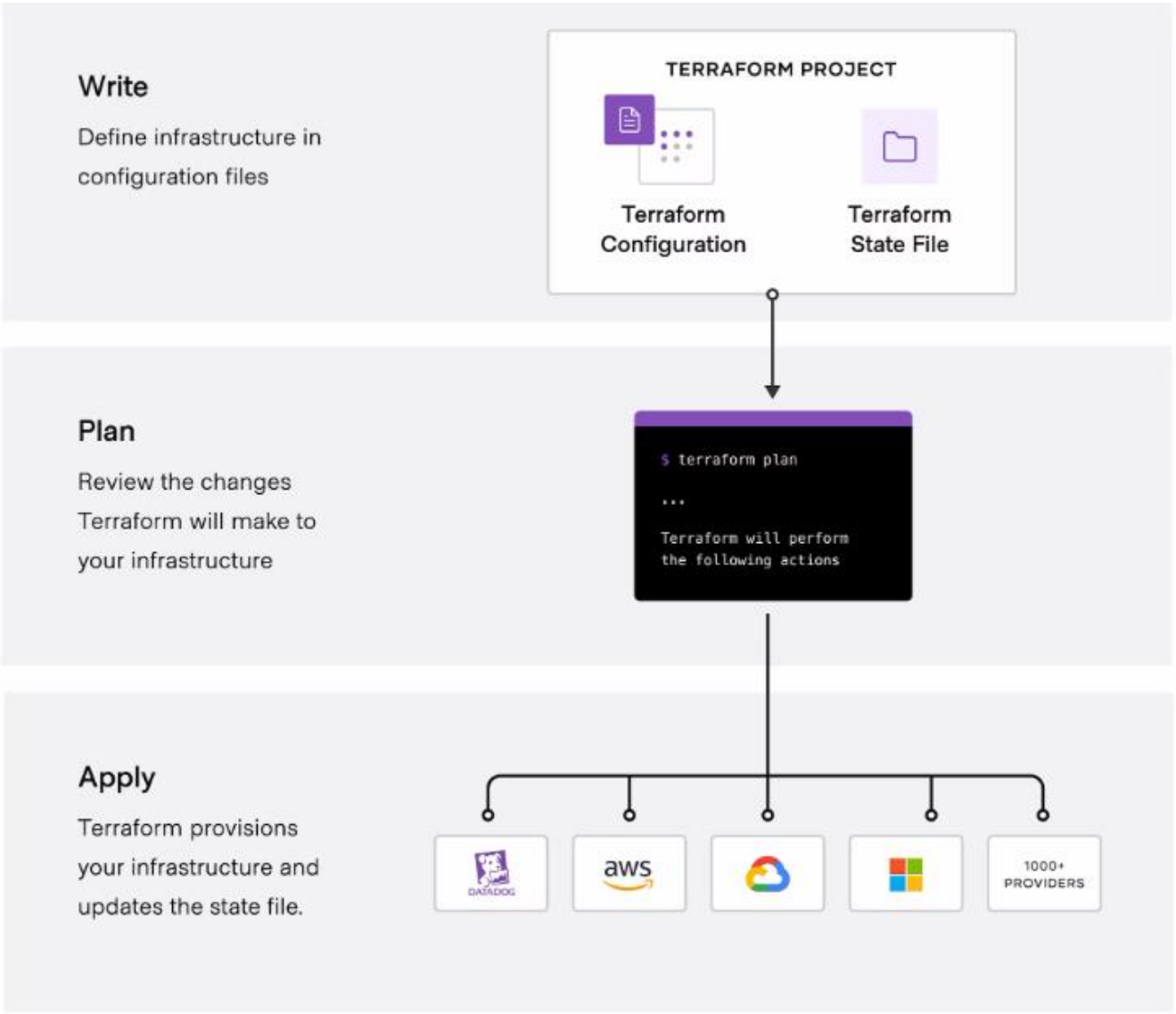

Рис. 2.8. Робочий процес Terraform

Terraform створює план виконання, що описує інфраструктуру, яку він створить, оновить або знищить на основі існуючої інфраструктури та конфігурації.

Після схвалення, Terraform виконує запропоновані операції в правильному порядку, зважаючи на будь-які залежності ресурсів. Наприклад, якщо оновити властивості VPC і змінити кількість віртуальних машин у цьому VPC, Terraform відтворить VPC перед масштабуванням віртуальних машин.

Terraform забезпечує, оновлює та знищує інфраструктурні ресурси, такі як фізичні машини, віртуальні машини, мережеві комутатори, контейнери тощо.

Конфігурації – це код, написаний для Terraform із використанням мови конфігурації HashiCorp (HCL) для опису бажаного стану ресурсів інфраструктури.

Провайдери – це плагіни, які Terraform використовує для керування цими ресурсами. Кожна підтримувана служба або платформа інфраструктури має постачальника, який визначає, які ресурси доступні, і виконує виклики API для керування цими ресурсами.

Модулі – багаторазові конфігурації Terraform, які можна викликати та налаштовувати іншими конфігураціями. Більшість модулів керують кількома тісно пов’язаними ресурсами від одного постачальника.

Реєстр Terraform спрощує використання будь-якого провайдера чи модуля. Щоб використовувати провайдер або модуль із цього реєстру, просто потрібно додати його до конфігурації; коли запускається `terraform init`, Terraform автоматично завантажує все, що йому потрібно.

2.3. Формалізація структури рішення на основі Kubernetes

Оскільки, рішення автоматизованого розгортання електронних магазинів на основі використання Kubernetes cluster, володіє властивостями шарів (рис. 2.5), то доцільно при математичному описі скористатися теорією функцій і теорії множини. Це дозволить відобразити залежності і структуру такого рішення і в подальшому ефективно використовувати для вирішення подібних задач.

В загальному випадку, сукупність DevOps практик автоматизації процесів для одержання результату у вигляді розгорнутого додатку можна представити наступним чином

$$Dev(Practice) \xrightarrow{apply} App \qquad (2.1)$$

де Dev – функція, що описує налаштування деякої DevOps практики;

$Practice$ – конкретна DevOps практика/практики;

App – додаток, розгортання якого необхідно забезпечити.

За структурою використовуваних практик при автоматизованому розгортанні електронних магазинів згідно архітектури рішення представленого на рис. 2.5, їх можна представити як сукупність інструментів, які описують конфігурацію кластера та розгортання додатків

$$Sol = \{T_i\{Ch_j\}\} \quad (2.2)$$

де Sol – рішення, що забезпечує автоматизацію процесу розгортання конфігурації і додатків;

T_i – інструменти налаштування конфігурації, $i = 1, n$ – ідентифікатор інструменту, n – кількість інструментів;

Ch_j – інструменти розгортання контейнерів або додатків, які використовуються для їх доступу і функціонування у кластері, $j = 1, m$ – ідентифікатор chart, m – кількість використовуваних інструментів розгортання.

У випадку архітектури рішення автоматизованого розгортання електронних магазинів інструментом глобального опису конфігурації і налаштування виступає Terraform, а інструментом, що забезпечує розгортання і коректність роботи додатків – Helm Chart.

Оскільки, у Terraform модна описати налаштування конфігурації безпосередньо самого кластера і контейнерів, то справедливо для даного випадку буде опис наступного вигляду

$$Terr = \{KubCl\ \{Pods_i\}\} \quad (2.3)$$

де $KubCl$ – множина параметрів, які описують кластер Kubernetes;

$Pods_i$ – сукупність представлень (контейнер) для функціонування додатків на базі розгорнутого кластеру.

Для забезпечення гнучкості функціонування додатків, або розмежування прав доступу до них можливе їх представлення як у вигляді окремого контейнера, так і об’єднання у вигляді сервісу, тобто

$$App_i \rightarrow \{Pods_1 \dots Pods_k\}$$

$$App_i \rightarrow Pods_i \tag{2.4}$$

Інфраструктуру провайдера хмарного сервісу можна представити у вигляді множини, компонентами якої є характеристики апаратного забезпечення, необхідного для функціонування електронного магазину та метаопис використовуваних операційних систем

$$PrInf = \{HW_i, OS_i\} \tag{2.5}$$

де $PrInf$ – інфраструктура провайдера хмарних сервісів;

HW_i – апаратне забезпечення хмарного провайдера;

OS_i – операційні системи, які використовується провайдером хмарних послуг.

Якщо описувати архітектурне рішення за допомогою шарів, то на найнижчому рівні L_0 буде знаходитись $PrInf$, тобто можна записати

$$PrInf = \{HW_i, OS_i\} \Rightarrow L_0 \tag{2.6}$$

На наступному рівні знаходиться кластер Kubernetes, що формально можна записати як

$$KubCl \Rightarrow L_1 \tag{2.7}$$

На наступному рівні L_2 знаходяться поди з контейнерами, тобто

$$\{Pods_i\} \Rightarrow L_2 \quad (2.8)$$

Найвищий рівень рішення щодо DevOps практик містить множину застосунків та контейнерів

$$\{Pods_i, App_i\} \Rightarrow L_3$$

або

$$\{\{Pods_{ij}\}, App_i\} \Rightarrow L_3 \quad (2.9)$$

Загалом на рівні шарів запропоноване рішення можна представити у вигляді формули 2.10

$$Sol = \{L_0\{L_1\{L_2\{L_3\}\}\}\} \quad (2.10)$$

Таким чином, формально представлено структуру рішення за шарами, а також за структурою використовуваних інструментів для автоматизації процесу розгортання електронних магазинів.

2.4. Висновки до розділу

Основні наукові та практичні результати, які отримані у даному розділі полягають в наступному:

1. Проаналізовано типові архітектури організації програмного забезпечення та обгрунтовано доцільність використання DevOps практик для організації та забезпечення залежності компонентів на основі архітектури мікросервісів при автоматизованому розгортанні електронних магазинів, що дало змогу в подальшому побудувати два альтернативних рішення: на основі Kubernetes та з використанням OpenStack.

2. Запропоновано рішення застосування DevOps практик, що передбачають використання кластеру Kubernetes та розгортання інфраструктури за допомогою технологічного рішення Terraform і Helm Chart і дають змогу гнучкого налаштування додатків і сервісів користувача при імплементації магазинів електронної комерції на основі OpenCart.

3. Формалізовано структуру та залежності інфраструктури і програмного забезпечення за допомогою представлення DevOps практик у вигляді множин та відповідних компонентів, що дало змогу забезпечити ілюстрацію вкладеності рівнів з одночасним відображенням їх структури і в практичному сенсі визначило порядок застосування DevOps практик.

4. Наведено особливості практичного застосування інструментів для налаштування кластера Kubernetes та його структурних компонентів, а також провайдера Vault, особливостей Terraform та Helm Chart, що дало змогу оцінити складність робіт, яка проявляється у залученні фахівців з високими технологічними скілами та ефективності даного рішення з точки зору надійності та часу імплементації.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ DEVOPS ПРАКТИК НА ОСНОВІ OPENSTACK, LAMP ТА OPENCART

3.1. Побудова та формалізація архітектурного рішення на основі OpenStack та LAMP

У попередньому розділі запропоновано та реалізовано рішення із застосуванням DevOps практик, які використовують Kubernetes cluster і дають змогу організувати автоматизоване розгортання електронних магазинів на основі OpenCart. Альтернативою попередній архітектурі є рішення, які проілюстровані на рис. 3.1.

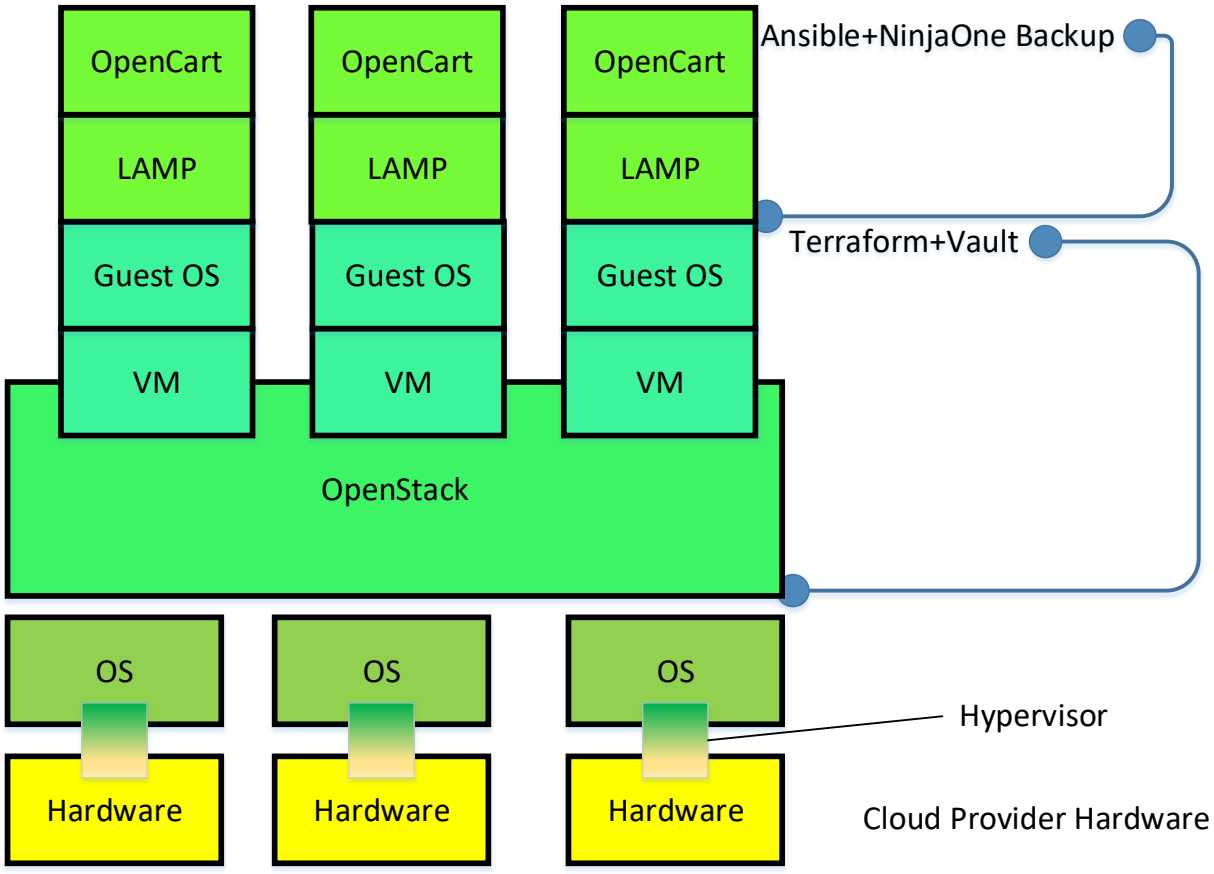

Рис. 3.1. Альтернативне рішення при розгортанні електронних магазинів на основі OpenCart

Як видно, з рис. 3.1 при організації автоматизованого розгортання електронних магазинів використовується апаратне забезпечення постачальника хмарних сервісів та операційні системи і за допомогою гцпервізора, що функціонує безпосередньо на апаратному забезпеченні дає змогу забезпечити безпеку для наступних рівнів.

За допомогою OpenStack формується платформа для розгортання віртуальних машин на яких будуть працювати гостьові операційні системи. На цих операційних системах потрібно встановити необхідне програмне забезпечення (LAMP), яке забезпечує розгортання електронного магазину на основі OpenCart. Для опису конфігурації використовується Terraform, а для розгортання та формування образів – відповідно Ansible та Ninja One BackUp.

Формалізацію рішення (рис. 3.1) пропонується виконати за допомогою нотацій, які представлені у розділі 2 з певними змінами.

$$Sol = \{T_i\{Depl_j, Img_k\}\} \tag{3.1}$$

T_i – інструменти налаштування конфігурації, $i = 1, n$ – ідентифікатор інструменту, n – кількість інструментів;

$Depl_j$– інструменти розгортання, $j = 1, m$ – ідентифікатор інструменту, m – кількість використовуваних інструментів розгортання.

Img_k – інструменти формування образів допоміжного програмного забезпечення та додатків.

Для опису конфігурації за допомогою OpenStack використовується Terraform, розгортання інфраструктури Ansible, а формування образів – Ninja One BackUp. Конфігурацію Terraform можна представити у наступному вигляді

$$Terr = \{OpenStack\{VM_i, GOS_i\}\} \tag{3.2}$$

де $OpenStack$ – сукупність параметрів для опису $OpenStack$;

VM_i – віртуальні машини, які використовуються для розгортання додатків;

GOS_i – гостьові операційні системи на VM_i.

Оскільки, для ефективного функціонування електронних магазинів на основі OpenCart необхідно наявність такого додаткового програмного забезпечення як Apache, MySQL та PHP, то доцільно застосувати технологію LAMP, тобто додаток в загальному випадку можна описати як:

$$App_i \rightarrow \{LAMP, OpenCart\} \tag{3.3}$$

Формальне представлення ресурсів хмарного провайдера описується як показано нижче

$$PrInf = \{HW_i, Hyper_i, OS_i\} \tag{3.4}$$

де $Hyper_i$ – множина використовуваних гіпервізорів.

Залежність між апаратним забезпеченням та операційними системами через гіпервізор, можна записати як

$$HW_i \xrightarrow{Hyper_i} OS_i \tag{3.5}$$

Для опису рівнів архітектурного рішення автоматизованого розгортання електронних магазинів використовується такі ж формули, як і в другому розділі з певними змінами. Представлення нульового рівня зображається як

$$\{HW_i, Hyper_i, OS_i\} \Rightarrow L_0 \tag{3.6}$$

Перший рівень рішення описується нижче наведеною формулою

$$OpenStack \Rightarrow L_1 \quad (3.7)$$

Наступний рівень представляє собою сукупність віртуальних машин із встановленими на них гостьовими операційними системи

$$\{VM_i, GOS_i\} \Rightarrow L_2 \quad (3.8)$$

Завершальним рівнем архітектурного рішення є рівень, що описується наступним виразом

$$\{LAMP, OpenCart\} \Rightarrow L_3 \quad (3.9)$$

Як і в першому випадку, рішення щодо автоматизованого розгортання електронного магазину представляється у вигляді вкладених шарів, наведених у формулі 2.10. Далі перейдемо до опису та налаштування основних компонентів запропонованого на рис. 3.1 рішення.

3.2. Гіпервізор при організації рішення розгортання електронних магазинів

Гіпервізор, також відомий як монітор віртуальної машини або VMM, що по факту є програмним забезпеченням, яке створює та запускає віртуальні машини (ВМ). Гіпервізор дозволяє одному хост-комп'ютеру підтримувати кілька гостьових віртуальних машин шляхом віртуального спільного використання ресурсів, таких як пам'ять і обробка.

Гіпервізори дають змогу використовувати більше доступних ресурсів системи та забезпечують більшу мобільність IT, оскільки гостьові віртуальні машини не залежать від апаратного забезпечення хоста. Це означає, що їх можна легко переміщувати між різними серверами.

Оскільки, кілька віртуальних машин можуть працювати на одному фізичному сервері з гіпервізором, гіпервізор зменшує:

– простір даних;
– використовувану енергію;
– вимоги до технічного обслуговування.

Існує два основних типи гіпервізора, які називаються «bare metal» і «hosted» Гіпервізор першого типу діє як легка операційна система та працює безпосередньо на апаратному забезпеченні хоста, тоді як гіпервізор другого типу працює як програмний рівень в операційній системі, як і інші комп'ютерні програми.

Найпоширенішим типом гіпервізора є чистий гіпервізор (тип 1), де програмне забезпечення віртуалізації встановлюється безпосередньо на апаратне забезпечення, де зазвичай інсталюється операційна система. Оскільки голі гіпервізори ізольовані від схильної до атак операційної системи, вони надзвичайно безпечні.

Крім того, вони, зазвичай, працюють краще та ефективніше, ніж гіпервізори другого типу. З цих причин більшість корпоративних компаній обирають голі гіпервізори для потреб центрів обробки даних.

У той час як голі гіпервізори працюють безпосередньо на обчислювальному обладнанні, «hosted» гіпервізори працюють поверх операційної системи (ОС) головної машини.

Хоча гіпервізори другого типу працюють в ОС, додаткові (і різні) операційні системи можуть бути встановлені поверх гіпервізора. Недоліком «hosted» гіпервізорів є те, що затримка вища, ніж у голих гіпервізорів. Це пояснюється тим, що зв'язок між обладнанням і гіпервізором має проходити через додатковий рівень ОС. Гіпервізори другого типу іноді називають гіпервізорами клієнтів, оскільки вони найчастіше використовуються для кінцевих користувачів і тестування програмного забезпечення, де більша затримка не викликає проблем.

Технологія апаратного прискорення може швидше створювати віртуальні ресурси та керувати ними, підвищуючи швидкість обробки як для голих, так і для hosted гіпервізорів. Тип апаратного прискорювача, відомий як віртуальний

виділений графічний прискорювач (vDGA), піклується про надсилання та оновлення високоякісної тривимірної графіки. Це звільняє основну систему для інших завдань і значно збільшує швидкість відображення зображень.

Для таких галузей, як розвідка нафти та газу, де є потреба швидко візуалізувати складні дані, ця технологія може бути дуже корисною. Обидва типи гіпервізорів можуть запускати кілька віртуальних серверів для кількох орендарів на одній фізичній машині.

Провайдери публічних хмарних послуг здають серверний простір на різних віртуальних серверах різним компаніям. На одному сервері може розміщуватися кілька віртуальних серверів, які виконують робочі навантаження для різних компаній. Цей тип спільного використання ресурсів може призвести до ефекту «шумового сусіда», коли один із орендарів виконує велике навантаження, що заважає продуктивності сервера для інших орендарів. Крім того, це створює більший ризик для безпеки, ніж використання виділеного «голого металу» сервера.

Стандартний сервер, який має повний контроль одна компанія, завжди забезпечуватиме вищу продуктивність, ніж віртуальний сервер, який ділиться пропускною спроможністю, пам’яттю та обчислювальною потужністю фізичного сервера з іншими віртуальними серверами.

Апаратне забезпечення для «голих металевих» серверів також можна оптимізувати для підвищення продуктивності, чого не можна сказати про спільні публічні сервери. Підприємства, яким необхідно дотримуватися нормативних актів, що вимагають фізичного розділення ресурсів, повинні будуть використовувати власні сервери, які не використовують спільні ресурси з іншими орендарями.

Оскільки хмарні обчислення стають повсюдними, гіпервізор став безцінним інструментом для запуску віртуальних машин і стимулювання інновацій у хмарному середовищі. Гіпервізор – це рівень програмного забезпечення, який дозволяє одному хост-комп’ютеру одночасно підтримувати кілька віртуальних машин, гіпервізори є ключовим елементом технології, яка робить можливим хмарні обчислення. Гіпервізори забезпечують хмарні програми доступними для

користувачів у віртуальному середовищі, водночас дозволяючи IT-спеціалістам зберігати контроль над інфраструктурою хмарного середовища, програмами та конфіденційними даними.

Цифрова трансформація і зростання очікувань клієнтів спонукають до більшої довіри до інноваційних програм. У відповідь багато підприємств переносять свої віртуальні машини в хмару. Однак необхідність переписувати кожну існуючу програму для хмари може використовувати дорогоцінні IT-ресурси та призвести до розриву інфраструктури. На щастя, будучи невід'ємною частиною платформи віртуалізації, гіпервізор може допомогти швидко перенести програми в хмару.

У результаті підприємства можуть скористатися багатьма перевагами хмари, зокрема скороченням витрат на апаратне забезпечення, підвищеною доступністю та більшою масштабованістю, для швидшого повернення інвестицій.

Гіпервізори підтримують створення та керування віртуальними машинами (ВМ), абстрагуючи програмне забезпечення комп'ютера від його апаратного забезпечення і роблять віртуалізацію можливою, перекладаючи запити між фізичними та віртуальними ресурсами.

Гіпервізори «bare-metal» іноді вбудовані в мікропрограму на тому ж рівні, що й базова система вводу-виводу (BIOS) материнської плати, щоб надати операційній системі на комп'ютері доступ до програмного забезпечення віртуалізації та використовувати його.

3.3. Застосування OpenStack при автоматизації розгортання OpenCart

З рис. 3.1 видно, що після апаратного забезпечення, гіпервізора та операційної системи запропоновано використовувати OpenStack/

Платформа OpenStack – це рішення з відкритим кодом, яке використовує поширені механізми віртуалізації для побудови та керування приватними хмарами. Проект складається з набору компонентів, які разом надають функціонал хмарної платформи: створення віртуальних комп'ютерних ресурсів, мереж для їх взаємодії,

сховищ даних, а також програмні та веб-інтерфейси для ідентифікації та роботи користувачів. OpenStack підтримує популярні механізми віртуалізації: KVM, Xen, VMware, vSphere, Hyper-V, LXC, Docker.

Абстрагування додатків від хоста – контейнер з'єднується з хостом певним інтерфейсом, контейнеризований додаток не залежить від архітектури або ресурсів хоста. Для хоста контейнер є якимось абстрактним "чорним ящиком", не має значення, що в ньому.

Масштабування – на одній машині може бути запущено декілька контейнерів і в той же час вони можуть бути запущені і на тестовому сервері. Дана можливість називається простим лінійним масштабуванням.

Управління версіями і залежностями – завдяки використанню контейнерів, розробки прив'язують всі компоненти і залежності до додатка, що дозволяє працювати як з цільним об'єктом. На хості не потрібні встановлення додаткових компонент або залежностей для запуску програми, яка знаходиться всередині контейнера, досить можливості запуску докер контейнера.

Ізолювання середовища – ізоляція в контейнерах не досягає такого ж рівня як при віртуалізації, але тим не менше має легке середовище виконання і відноситься до ізоляції на рівні процесів. При цьому контейнер працює на тому ж самому ядрі, що дозволяє йому швидко запускатися. Запуск великої кількості контейнерів на робочій машині не буде серйозної проблемою.

Однак, існують проблеми контейнеризації:

– чутливість налаштування – при великих розмірах масштабованості необхідне дуже чітке і якісне налаштування систем;

– зворотна сумісність – докер швидко розвивається і одним з мінусів такого розвитку буває обмежена зворотна сумісність по деякими напрямках;

– продуктивність – додаткові надбудови на системі в будь-якому випадку призводять до збільшення навантаження і витрат ресурсів;

– підтримка – для підтримки і супроводу докер-контейнерів необхідні не тільки навички системного адміністратора, але і хороші знання докера.

Одним з найбільш поширених варіантів використання контейнерів, а також найбільшою мірою, яка сприяє їх широкому поширенню, є мікросервіси (microservices).

При необхідності масштабування «монолітної» програми вибір, як правило, обмежений тільки варіантом вертикального масштабування (scale up), і зростаючі потреби задовільняються використанням більш потужного комп'ютера з великим об'ємом оперативної пам'яті і більш продуктивний процесор. На противагу такому підходу мікросервіси призначені для горизонтального масштабування (scale out), коли зростання потреб задовільняється додаванням кількох серверів з розподілом навантаження між ними. Що і підштовхує до переходу від традиційної віртуалізації до контейнеризації мікросервісов.

OpenStack — це платформа з відкритим вихідним кодом, яка використовує об’єднані віртуальні ресурси для створення приватних і публічних хмар і керування ними. Інструменти, що входять до складу платформи OpenStack, називаються «проектами», обробляють основні служби хмарних обчислень, а саме: обчислення, мережу, сховище, ідентифікацію та зображення.

Більше дюжини додаткових проектів також можна об’єднати разом, щоб створити унікальні хмари, які можна розгортати. У віртуалізації такі ресурси, як сховище, процесор і оперативна пам’ять, абстрагуються від різноманітних програм певних постачальників і поділяються гіпервізором перед розподілом за потреби. OpenStack використовує послідовний набір програмних інтерфейсів прикладних програм (API), щоб абстрагувати ці віртуальні ресурси на 1 крок далі в окремі пули, які використовуються для забезпечення стандартних інструментів хмарних обчислень, з якими адміністратори та користувачі взаємодіють безпосередньо.

OpenStack і платформи керування віртуалізацією розташовані на основі віртуалізованих ресурсів і можуть виявляти, звітувати та автоматизувати процеси в різних середовищах постачальників. Але в той час як платформи керування віртуалізацією спрощують маніпулювання характеристиками та функціями віртуальних ресурсів, OpenStack фактично використовує віртуальні ресурси для запуску комбінації інструментів. Ці інструменти створюють хмарне середовище,

яке відповідає 5 критеріям Національного інституту стандартів і технологій щодо хмарних обчислень: мережа, об’єднані ресурси, інтерфейс користувача, можливості надання та автоматичний контроль/розподіл ресурсів.

OpenStack — це, по суті, набір команд, відомих як скрипти. Ці сценарії об’єднані в пакети, які називаються проектами, які передають завдання, які створюють хмарне середовище. Для створення таких середовищ OpenStack покладається на 2 інші типи програмного забезпечення:

– віртуалізація, яка створює рівень віртуальних ресурсів, абстрагованих від апаратного забезпечення;

– базова операційна система (ОС), яка виконує команди, надані сценаріями OpenStack.

Сам OpenStack не віртуалізує ресурси, а скоріше використовує їх для створення хмар. OpenStack також не виконує команди, а передає їх базовій ОС. Усі 3 технології — OpenStack, віртуалізація та базова ОС – мають працювати разом. Саме через цю взаємозалежність багато хмар OpenStack розгортається за допомогою Linux, що стало натхненням для рішення RackSpace і NASA випустити OpenStack як програмне забезпечення з відкритим кодом.

Архітектура OpenStack складається з багатьох проектів з відкритим кодом. Ці проекти застосовуються для налаштування «підхмари» та «надхмари» OpenStack, які використовуються системними адміністраторами та користувачами хмари відповідно.

Підхмари містять основні компоненти, необхідні системним адміністраторам для налаштування та керування середовищами OpenStack кінцевих користувачів, відомими як «надхмари». Існує 6 стабільних основних служб, які обслуговують обчислення, мережу, сховище, ідентифікацію та зображення, тоді як більше десятка додаткових сервісів відрізняються за ступенем розвитку. Ці основні сервіси є інфраструктурою, яка дозволяє решті проектів обробляти інформаційні панелі, оркестровку, ініціалізацію «голого металу», обмін повідомленнями, контейнери та керування.

– Nova – це повний інструмент для керування обчислювальними ресурсами OpenStack і доступу до них, який забезпечує планування, створення та видалення;

– – Neutron об’єднує мережі в інших службах OpenStack;

– Swift – це високовідмовостійка служба зберігання об’єктів, яка зберігає та отримує неструктуровані об’єкти даних за допомогою RESTful API;

– Cinder – забезпечує постійне сховище блоків, доступних через API самообслуговування;

– Keystone – автентифікує та авторизує всі служби OpenStack. Це також каталог кінцевих точок для всіх служб.

– Glance – зберігає та отримує образи дисків віртуальних машин із різних місць.

3.4. Провайдер Vault

Провайдер Vault дозволяє Terraform читати, записувати та налаштовувати HashiCorp Vault. Terraform може використовуватися адміністраторами Vault для його налаштування і заповнення ключами. У цьому випадку стан і будь-які плани, пов’язані з конфігурацією, потрібно зберігати та передавати обережно, оскільки вони міститимуть у відкритому вигляді будь-які значення, записані в Vault.

Наразі Terraform не має механізму для редагування або захисту ключів, які надаються через конфігурацію.Для доступу та організації сховища необхідно звернути особливу увагу на документацію кожного ресурсу про те, як будь-які ключі зберігаються в стані, і ретельно розглянути чи таке використання сумісне з їх політикою безпеки. Якщо не зазначено інше, ресурси, які записують ключі Vault, розроблені таким чином, що вони вимагають лише доступу до створення та оновлення на відповідних ресурсах, щоб різні маркери могли використовуватися для читання чи записуі таким чином, обмежити доступ до скомпрометованого маркера.

Більшість провайдерів Terraform вимагають облікових даних для взаємодії зі сторонньою службою, яку вони обгортають. Цей провайдер дозволяє отримувати

такі облікові дані від Vault, що означає, що операторам або системам, на яких запущено Terraform, потрібен лише доступ до відповідного привілейованого токена Vault, щоб тимчасово орендувати облікові дані для інших постачальників.

Наразі Terraform не має механізму для редагування або захисту ключів, які повертаються через джерела даних, тому вони одержуються через цей провайдера і зберігаються у стані Terraform у файлах плану, а в деяких випадках у виводі консолі, створеному під час планування та застосування. Тому всі ці артефакти мають бути захищені відповідним чином. Щоб обмежити доступ до таких ключів, провайдер робить запит маркера Vault із відносно коротким TTL (20 хвилин, за замовчуванням), що, у свою чергу, означає відклик всіх виданих облікових даних після цього часу. Але він не може відкликати будь-які статичні ключі, наприклад ті, що зберігаються в «загальному» сервері ключів Vault. Запитуваний маркер TTL можна контролювати за допомогою аргументу постачальника max_lease_ttl_seconds.

Важливо враховувати, що Terraform читає з джерел даних на етапі планування та записує результат у план. Таким чином, наступне звернення, ймовірно, не вдасться виконати, якщо його запустити після закінчення терміну дії проміжного токена через відкликання ключів, які зберігаються в плані.

3.5. Застосування технології LAMP для розгортання електронного магазину на основі OpenCart

Перш за все необхідно в якості операційної системи на віртуальній машині необхідно встановити операційну систему CentOs, що належить до класу UNIX-прдібних операційних систем. Важливою перевагою цієї системи є те, що для її ефективного використання висуваються менші апаратні ресурси, ніж до інших з такого класу. Далі необхідно встановити MySQL. Для цього спочатку відключаються наступні сервіси операційної системи (рис. 3.8.)

```
systemctl stop firewalld.service
  systemctl disable firewalld
  systemctl status firewalld
```

Рис. 3.8. Відключення сервісів ОС

Важливим також є правильне налаштування DNS, що прописується у файлі resolv.conf і представлено нижче на рис. 3.9.

```
/etc/resolv.conf
nameserver 8.8.8.8
nameserver 8.8.4.4
```

Рис. 3.9. Налаштування DNS

Наступний крок передбачає відключення сервісу SELinux, шляхом виконання команди «*setenforce 0*»та встановлення «*SELINUX=disabled*» у файлі «*vi /etc/sysconfig/selinux*»

Далі потрібно оновити репозиторій:

```
yum -y install epel-release
```

Для інсталяції MySQL, зокрема light версії MariaDB, потрібно виконати команду:

```
yum provides mysql
```

За допомогою команди «*rpm -Uvh https://dev.mysql.com/get/mysql80-community-release-el7-5.noarch.rpm*» встановлюється в локальний репозиторій дозволених програм.

Безпосередньо сам процес встановлення MySQL передбачає виконання наступних команд:

```
yum -y install mysql-community-server
yum -y install mysql
```

Після цього потрібно додати службу MySQL в автозавнатаження та запустити шляхом виконання команд:

```
systemctl enable mysqld
systemctl start mysqld
```

Для перевірки статусу та паролю, а також зміни його зміни після встановлення MySQL потрібно виконати команди, як показано на рис. 3.10.

```
systemctl status mysqld
more /var/log/mysqld.log | grep password
/usr/bin/mysql_secure_installation
```

Рис. 3.10. Перевірка статусу паролю

Провівши такі налаштування потрібно перезавантажити сервіс mysql, виконавши команду:

```
systemctl restart mysqld.service
```

Перевірка логіну та паролю виконується за допомогою наступних команд:

```
mysql -u root -p
mysql -u root --password="************"
mysql -h 192.168.0.22 \
  -u ******* --password="***********"
```

Створення локального та мережевого користувача в СКБД MySQL передбачає виконання наступних скриптів:

– для локального користувача:

```
CREATE USER **********@'localhost' IDENTIFIED BY '*********';
  GRANT ALL ON *.* TO '***********'@'localhost';
```

– для мережевого користувача:

```
CREATE USER **********'@'%' IDENTIFIED BY '************';
GRANT ALL ON *.* TO '*************'@'%';
```

Далі потрібно створити базу даних для OpenCart шялхом виконання запиту:

```
create database opencart character set utf8 collate utf8_bin;
```

Якщо все пройшло успішно, то виконавши команду show databases, можна побачити таблиці бази даних OpenCart.

В подальшому будуть налаштовані Apache та PHP, що передбачають типову інсталяцію, приклади якої наведені у відкритих джерелах.

Так, для інсталяції та базового налаштування Apache використовується команди, наведені на рис.3.11.

```
yum -y install httpd
systemctl start httpd
systemctl enable httpd
systemctl status httpd
vi /etc/httpd/conf/httpd.conf
systemctl restart httpd
yum -y install php php-mysql php-gd php-ldap php-odbc php-pear
php-xml php-xmlrpc php-mbstring php-snmp php-soap php-mcrypt curl zlib
```

Рис. 3.11. Встановлення Apache та PHP

Наступний крок полягає у встановленні та налаштуванні самого OpenCart, як показано у вигляді лістингу на рис. 3.12.

```
mkdir opencart
yum -y install wget
wget
https://github.com/opencart/opencart/releases/download/2.3.0.2/2.3.0
.2-compiled.zip
yum -y install unzip
mv 2.3.0.2-compiled.zip opencart/
unzip 2.3.0.2-compiled.zip
mv 2.3.0.2-compiled.zip ~
cd ..
cp -avr opencart /var/www/html
cd /var/www/html
ls -al
chmod -R 775 opencart
chown -R apache:apache opencart
cd /var/www/html/opencart/upload
cp config-dist.php config.php
cp config-dist.php config.php
cd /var/www/html
chown -R apache:apache opencart
systemctl restart httpd
http://192.168.[redacted]/opencart/upload/install/index.php
```

Рис. 3.12. Встановлення та налаштування OpenCart

У результаті коректного встановлення OpenCart зайшовши на local host повинна завантажитися сторінка з налаштуваннями і введення даних користувача, після цього можна буде перейти до безпосереднього налаштування електронного магазину (рис. 3.13).

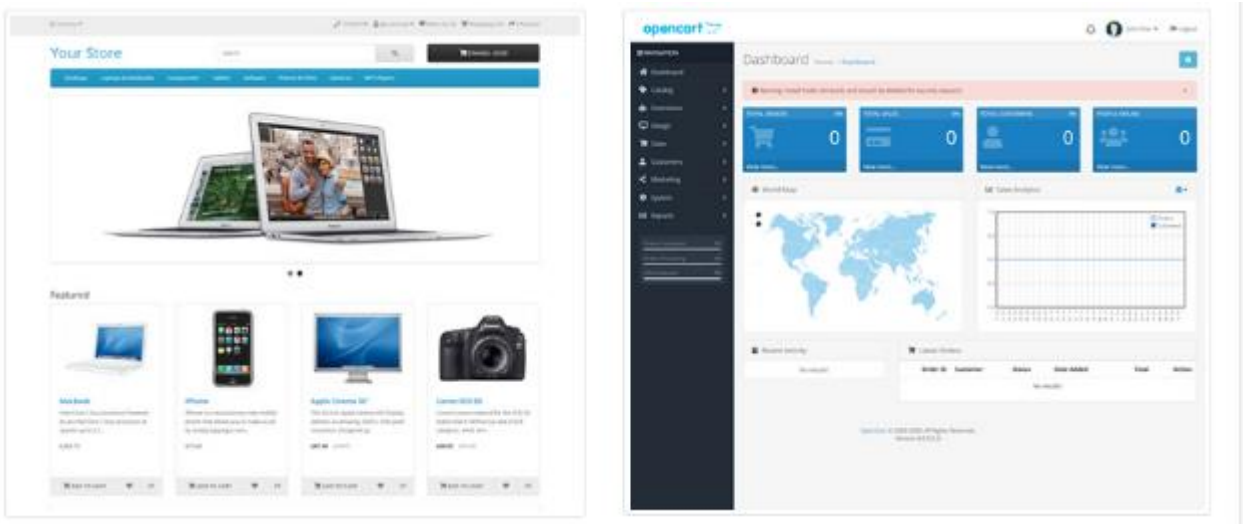

Рис. 3.13. Інтерфейс успішно розгорнутого електронного магазину (зліва – інтерфейс користувача, справа- інтерфейс адміністратора)

Розгортання наведене на рис. 3.12 більш ефективно виконується за допомогою Ansible з готовим playbook, який можна заванатажити за посиланням https://github.com/mrlesmithjr/ansible-opencart..

Після цього, власники електронного магазину можуть налаштувати його у відповідності до своїх потреб.

3.6. Висновки до розділу

Основні результати даного розділу полягають в наступному:

1. Побудовано альтернативне рішення щодо автоматизованого розгортання електронних магазинів, що передбачає використання платформи OpenStack з розгортанням на ній віртуальних машин з гостьовими операційними системами, комплексом серверного програмного забезпечення LAMP та

безпосередньо самого OpenCart, що дало змогу забезпечити простоту застосування і функціональність DevOps практик та імплементацію працездатного сервісу.

2. Запропоновано формалізоване представлення побудованого рішення автоматизованого розгортання електронних магазинів на основі OpenCart, що передбачає застосування інструментів Terraform та Vault, а також Ansible і Ninhja One BackUp, що дало змогу, як і в попередньому рішенні, представити вкладеність абстрактних рівнів і структуру їх комопнентів.

3. Проведено аналіз Dev Ops практик, що використовуються при побудові запропонованого рішення автоматизованого розгортання та налаштування електронних магазинів, що дало змогу практично реалізувати та успішно розгорнути електронний магазин з використання на верхньому рівні комплексу LAMP та OpenCart.

РОЗДІЛ 4
ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1. Охорона праці

Тема кваліфікаційної роботи присвячена дослідженню DevOps практик при автоматизації процесу розгортання електронних магазинів на основі OpenCart. При роботі із засобами автоматизації потрібно забезпечити дотримання вимог з охорони праці, техніки безпеки та протипожежної безпеки при використанні ПК.

Основними регламентуючими нормативними документами охорони праці користувачів комп'ютерів є:

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

– НАПБ А.01.001-2014 «Правила пожежної безпеки в Україні».

Вимоги до приміщень, згідно з [26, 27], щодо розташування робочого місця передбачають виконання наступних вимог:

– мінімальна площа, яка виділяється на одне робоче місце повинна становити мінімум 6,0 м^2, при об'ємі – мінімум 20,0 м^3;

– розташування робочих місць користувачів ПК заборонено у цокольних або підвальних приміщеннях.

При організації робочих місць у НПАОП 0.00-7.15-18 передбачено наявність природного і штучного освітлення. Зазвичай, природне освітлення поступає у приміщення через вікна та світлові прорізи і забезпечує коефіцієнт освітленості на рівні не менше 1,5%. Орієнтація вікон – на північ або північний схід. Штучне освітлення забезпечують відповідні джерела, наприклад, люмінесцентні лампи.

Приміщення з комп'ютерною технікою не повинні межувати з будівлями, де рівень шуму чи вібрації перевищує визначені допустимі значення. Покриття підлоги повинне бути матовим з коефіцієнтом відбиття 0,3-0,5. Для внутрішнього оздоблення приміщень слід використовувати дифузно-відбивні матеріали з коефіцієнтами відбиття для стелі 0,7-0,8, для стін 0,5-0,6 [26].

У приміщеннях, де організовано робочі місця користувачів ПК, повинні бути забезпечені аптечками першої медичної допомоги. Вологе прибирання у таких приміщеннях є обов’язковим кожного дня.

Щодо ергономічної організації робочого місця, то воно також повинно відповідати вимогам, наведеним у [23, 24]. Конструкція робочого місця повинна забезпечити підтримання оптимальної робочої пози. У відповідності до НПАОП 0.00-7.15-18, обладнання і організація робочого місця працюючих з ЕОМ мають забезпечувати відповідність конструкції всіх елементів робочого місця та їх взаємного, розташування ергономічним вимогам з урахуванням характеру і особливостей трудової діяльності.

Висота робочого столу з ПК повинна бути виконана в діапазоні 680...800 мм, а ширина і глибина – 600...1400 мм і 800..1000 мм відповідно. Стіл також повинен мати достатній простір для ніг, що забезпечить зручну осанку користувача.

Стілець на робочому місці користувача ПК повинен бути підйомно-поворотним, регульованим за висотою, за кутом і за нахилом сидіння та спинки [26].

Екран комп’ютера повинен бути розміщений на відстані 600...700 мм від очей користувача. Розташування монітору має забезпечувати зручність зорового спостереження у вертикальній площині під кутом +30 градусів до нормальної лінії погляду працівника [26].

Електромережі штепсельних з'єднань та електророзеток для живлення ПК потрібно виконувати за магістральною схемою. При організації робочих місць електромережу штепсельних розеток для живлення ПК у центрі приміщення прокладають у каналах або під знімною підлогою в металевих трубах або гнучких металевих рукавах [26].

Щодо безпеки при роботі з ПК, щодня перед початком роботи необхідно очищати монітор від пилу та інших забруднень. Після закінчення роботи з ПК, він та периферійні пристрої повинні бути відключені від електричної мережі. У разі виникнення певної аварійної ситуації необхідно негайно відключити ПК від

електричної мережі. Не допускається виконувати обслуговування, ремонт та налагодження ПК безпосередньо на робочому місці [26].

Основні вимоги до пожежної безпеки вказані в НАПБ А.01.001-2014 «Правила пожежної безпеки в Україні». Згідно з [26], на та під приміщеннями, в яких розміщені ЕОМ, а також у суміжних із ними приміщеннях не дозволяється розташування приміщень категорій А та Б за вибухопожежною небезпекою.

Фальшпідлога у приміщеннях з ЕОМ має бути з негорючих матеріалів або матеріалів груп горючості Г1, Г2 з межею вогнестійкості не менше 0,5 години. Простір під нею слід розділяти негорючими діафрагмами на відсіки площею не більше 250 м^2. Діафрагми повинні мати межу вогнестійкості не менше 0,75 год. Звукопоглинаюче облицювання стін та стель цих приміщень слід виготовляти з негорючих матеріалів або матеріалів груп горючості Г1, Г2.

Вогнегасники слід встановлювати у легкодоступних та помітних місцях (коридорах, біля входів або виходів з приміщень тощо), а також у пожежонебезпечних місцях, де найбільш вірогідна поява осередків пожежі. При цьому необхідно забезпечити їх захист від попадання прямих сонячних променів та безпосередньої (без загороджувальних щитків) дії опалювальних та нагрівальних приладів.

Вибір типу та необхідна кількість вогнегасників визначається відповідно до Типових норм належності вогнегасників, затверджених наказом Міністерства України з питань надзвичайних ситуацій та у справах захисту населення від наслідків Чорнобильської катастрофи від 02.04.2004 № 151.

У кваліфікаційній роботі досліджено DevOps практики для автоматизованого розгортання електронних магазинів, тому важливим та актуальним було провести аналіз основних вимог до приміщень та робочих місць з ПК, що дозволило забезпечити комфортні і безпечні умови праці інженерів комп’ютерних систем. Також проаналізовано правила електробезпеки під час роботи з ПК та вимоги до пожежної безпеки в приміщенні.

4.2. Шум, вібрація, ультразвук, електромагнітні випромінювання у виробничих приміщеннях для роботи з ВДТ та захист від них

Під шумом розуміють набір багаточисельних звуків, які швидко змінюються за частотою, силою і складаються з ряду гармонік [28]. З фізичної точки зору звуки є механічними коливальними рухами частинок пружного середовища в діапазоні частот, що чує людина. Звукові гармоніки розповсюджуються у вигляді хвиль. Шум є загально-біологічним подразником, діє не тільки на органи слуху, але може викликати порушення роботи серцево-судинної і нервової систем, зумовлювати професійні захворювання [28]. Основними характеристиками звукових коливань є інтенсивність (сила), частота і форма звукової хвилі. Інтенсивність визначається енергією, що переноситься за 1 с звуковою хвилею через поверхню площею 1 м^2, яка перпендикулярна напрямку розповсюдження звукової хвилі. Діапазон тисків, що сприймає вухо людини, дуже широкий (10-12Вт/м^2 – поріг больового відчуття, верхня межа) [28].

З розвитком промисловості все більший контингент людей підпадає під вплив вібрацій, які являють собою механічні коливання, що передаються тілу людини. Основні параметри вібрацій – частота та амплітуда коливань, але на відміну від шуму, при якому енергія механічних коливань передається через повітряне середовище, при дії вібрацій вона розповсюджується по тканинах і викликає їх коливання або тіла людини в цілому [28]. Найбільш небезпечна вібрація частотою 16-250 Гц, дія якої призводить до вібраційної хвороби. Нормування шуму здійснюється згідно з “Санітарними нормами допустимих рівнів шуму на робочих місцях”. В Україні застосовується принцип нормування шуму на основі граничних спектрів (гранично допустимих рівнів звукового тиску) в октавних смугах частот та еквівалентних рівнів звуку. Гранично-допустимі рівні шумів санітарними нормами встановлені для кожного класу [28]:

– для високочастотних шумів (вище 800 Гц) – 75-85 дБ;
– для середньо частотних шумів (300-800 Гц) – 85-90 дБ;
– для низькочастотних шумів (до 300 Гц) – 90-100 дБ.

Шумові явища мають якість кумуляції, накопичуючись в організмі, вони все більше і більше пригнічують нервову систему. Відомо, що після шумової дії інтенсивністю 120 дБ протягом однієї години потрібно 5 годин, щоб гострота слуху повернулась до норми. Стабільні широкосмугові шуми, які перевищують граничний рівень, викликають зниження темпу, ефективності й якості роботи операторів [28].

Ультразвук широко застосовують у технологічних процесах виготовлення радіоелектронної апаратури (промивка деталей, зварювання мініатюрних вузлів тощо) з частотою вище 2220 кГц. При цьому густина енергії ультразвукових коливань у мільйони разів більша густини енергії звуків, які ми чуємо. Тому під його дією відбувається нагрівання тіла, а при дії коливань через рідкі і тверді середовища відбувається розривання і руйнування тканин [28].

Захист від ультразвуку, який діє через повітряне середовище, досягається шляхом звукоізоляції установок (листова сталь, дюралюміній, що обклеєні гумою або руберойдом, гетинакс) або розміщення їх в окремій звукоізолюючій кабіні. Ультразвукові установки повинні мати блокування, яке відключає генератор ультразвукових коливань в момент відкривання кришок або кожухів [28].

Для запобігання шкідливої дії шуму і вібрації на організм працюючих проводяться технічні, організаційні і медико-профілактичні заходи. Одним з основних технічних заходів є зменшення при експлуатації та на стадії проектування, конструювання обладнання причин шуму і вібрації в самому джерелі утворення. Досягають цього завдяки використанню раціональної конструкції обладнання, заміни ударної дії деталей і машин коливальною, з’єднання елементів гнучкими зв’язками, врівноважування обертових частин механізмів, заміни металевих деталей пластмасовими, забезпечення різних власних частот коливань механізму з частотою збуджуючої сили [28].

Якщо неможливо ізолювати чи знизити шум і вібрацію самого джерела, потрібно:

– ізолювати джерело шуму або вібрації від навколишнього середовища засобами вібро- та звукоізоляції;

– раціонально планувати виробничі приміщення, що мають інтенсивні джерела шуму;

– збільшувати звукопоглинання внутрішніх поверхонь приміщення шляхом звукопоглинальних покриттів.

Якщо не вдається зменшити рівень шуму і вібрації на робочому місці до нормативних значень та необхідно використовувати засоби індивідуального захисту: рукавиці, взуття, навушники, м’які шоломи, які зменшують рівень звукового тиску на 40-50 дБ.

У процесі виробництв, експлуатації і зберігання комп’ютерної і радіоелектронної апаратури можуть виникати механічні і динамічні дії, що характеризуються широким діапазоном частот коливань, а також амплітудою, прискоренням і часом дії [28].

При експлуатації високочастотного обладнання всередині виробничих приміщень зниження напруженості електромагнітного випромінювання досягається такими методами:

– захист часом – обмеження часу перебування людини в електромагнітному полі, що залежить від інтенсивності опромінення або напруженості ЕМП.

– захист відстанню застосовується при неможливості послабити інтенсивність опромінення в заданій зоні іншими методами: збільшують відстань між джерелом випромінювання і обслуговуючим персоналом;

– добре виконане екранування джерела і усунення нещільності у фланцевих з’єднаннях, фідерів, зазорів у обшивці корпусів, нещільних електричних контактів;

– проведення дистанційного контролю й управління роботою передавачів з екранованого приміщення;

– засобами індивідуального захисту.

В залежності від типу джерела випромінювання, його потужності, характеру технологічного процесу може застосовуватись один з вказаних методів або будь-яка їх комбінація.

ВИСНОВКИ

Основні наукові та практичні результати полягають в наступному.

1. Проведено аналіз особливостей та принципів організації електронних магазинів у результаті якого встановлено, що широкої популярності та ефективності набуває тенденція щодо їх розгортання у хмарних сервісах, що дає змогу знизити витрати на утримання власної інфраструктури, підвищити надійність їх функціонування і доступності, а також зосередити більше уваги на веденні електронного бізнесу.

2. Досліджено принципи організації хмарних сервісів і встановлено, що найбільш трудомістким і водночас найбільш гнучким є підхід, який передбачає використання DevOps практик при організації інфраструктури як коду, що дає змогу забезпечити автоматизоване розгортання додатків та організувати гнучкість їх налаштування, що відповідає потребам замовників.

3. Проаналізовано та визначено переваги і недоліки технології хмарних сервісів на основі PaaS та SaaS, що передбачає надання частини готових сервісів з невеликими налаштуваннями з боку користувачів і водночас накладає обмеження на тип використовуваної інфраструктури, а також супроводжується підвищеним тарифним планом у порівнянні з IaaS.

4. Проаналізовано типові архітектури організації програмного забезпечення та обгрунтовано доцільність використання DevOps практик для організації та забезпечення залежності компонентів на основі архітектури мікросервісів при автоматизованому розгортанні електронних магазинів, що дало змогу в подальшому побудувати два альтернативних рішення: на основі Kubernetes та з використанням OpenStack.

5. Запропоновано рішення застосування DevOps практик, що передбачають використання кластеру Kubernetes та розгортання інфраструктури за допомогою технологічного рішення Terraform і Helm Chart і дають змогу гнучкого налаштування додатків і сервісів користувача при імплементації магазинів електронної комерції на основі OpenCart.

6. Формалізовано структуру та залежності інфраструктури і програмного забезпечення за допомогою представлення DevOps практик у вигляді множин та відповідних компонентів, що дало змогу забезпечити ілюстрацію вкладеності рівнів з одночасним відображенням їх структури і в практичному сенсі визначило порядок застосування DevOps практик.

7. Наведено особливості практичного застосування інструментів для налаштування кластера Kubernetes та його структурних компонентів, а також провайдера Vault, особливостей Terraform та Helm Chart, що дало змогу оцінити складність робіт, яка проявляється у залученні фахівців з високими технологічними скілами та ефективності даного рішення з точки зору надійності та часу імплементації.

8. Побудовано альтернативне рішення щодо автоматизованого розгортання електронних магазинів, що передбачає використання платформи OpenStack з розгортанням на ній віртуальних машин з гостьовими операційними системами, комплексом серверного програмного забезпечення LAMP та безпосередньо самого OpenCart, що дало змогу забезпечити простоту застосування і функціональність DevOps практик та імплементацію працездатного сервісу.

9. Запропоновано формалізоване представлення побудованого рішення автоматизованого розгортання електронних магазинів на основі OpenCart, що передбачає застосування інструментів Terraform та Vault, а також Ansible і Ninhja One BackUp, що дало змогу, як і в попередньому рішенні, представити вкладеність абстрактних рівнів і структуру їх комопнентів.

10. Проведено аналіз Dev Ops практик, що використовуються при побудові запропонованого рішення автоматизованого розгортання та налаштування електронних магазинів, що дало змогу практично реалізувати та успішно розгорнути електронний магазин з використання на верхньому рівні комплексу LAMP та OpenCart.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Big Data What it is and why it matters URL: https://www.sas.com/en_us/insights/big-data/what-is-big-data.html (дата звернення 10.11.2022 р.)

2. Min Chen, Shiwen Mao, Yin Zhang, Victor C.M. Leung. Big Data. Related Technologies, Challenges, and Future Prospects. Spinger, 2014. 100 с.

3. Amazon EMR Documentation URL: https://docs.aws.amazon.com/emr (дата звернення 11.11.2022 р.)

4. Google Dataproc documentation URL: https://cloud.google.com/dataproc/docs (дата звернення 15.11.2022 р.)

5. Create a cluster. URL: https://cloud.google.com/dataproc/docs/guides/create-cluster#dataproc-create-cluster-gcloud (дата звернення 23.11.2022 р.)

6. Schults C. What Is Infrastructure as Code? How It Works, Best Practices, Tutorials URL: https://stackify.com/what-is-infrastructure-as-code-how-it-works-best-practices-tutorials/ (дата звернення 18.11.2022 р.)

7. Grady B. Object Oriented Design: With Applications / Booch Grady. Boston, MA: Pearson Education, 2007. 551 с.

8. Brikman Y. Terraform: Up & Running: Writing Infrastructure as Code. Sebastopol, CA: O'Reilly Media, 2019. 368 с.

9. Abd-Allah A. Extending reliability block diagrams to software architectures / Center for Software Engineering. Computer Science Department. University of Southern California. Los Angeles. Technical Report: USC-CSE-97-501. URL:http:// sunset.usc.edu/publications/ TECHRPTS/1997/usccse97-501/usccse97-501.ps (дата звернення: 23.11.2022 р.)

10. Луцків А.М., Барна І.М. Т Особливості задач і функцій DevOps фахівців. Матеріали X науково-технічної конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі, системи та технології» (8-9 грудня 2022 року). Тернопіль: ТНТУ. 2022. С. 65.

11. Луцків А.М., Барна І.М. Аналіз сервісно-орієнтованої архітектури в процесі застосування DevOps практик. Матеріали X науково-технічної конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі, системи та технології» (8-9 грудня 2022 року). Тернопіль: ТНТУ. 2022. С. 66

12. Domingus J. Cloud Native DevOps with Kubernetes. 2nd Ed. O'Reilly Media.2022. 456 p.

13. Domingus J., Arundel J. Cloud Native DevOps with Kubernetes: Building, Deploying, and Scaling Modern Applications in the Cloud 2nd. O'Reilly Media. 2022. 383 p.

14. Understanding OpenStack. URL: https://www.redhat.com/en/topics/openstack (дата звернення 01.12.2022 р.)

15. How Terraform, providers and modules work. URL: https://registry.terraform.io/ (дата звернення 18.11.2022 р)

16. Deploy on Kubernetes. URL: https://docs.docker.com/desktop/kubernetes/ (дата звернення 18.11.2022 р)

17. What is a Kubernetes cluster? URL: https://www.vmware.com/topics/glossary/content/kubernetes-cluster.html (дата звернення 25.11.2022 р)

18. What is Kubernetes infrastructure? URL: https://www.vmware.com/topics/glossary/content/kubernetes-infrastructure.html (дата звернення 25.11.2022 р)

19. Kubernetes Clusters: Everything You Need To Know. URL: https://www.containiq.com/post/kubernetes-cluster (дата звернення 25.11.2022 р)

20. Vault Documentation. URL: https://developer.hashicorp.com/vault/docs?host=www.vaultproject.io (дата звернення 25.11.2022 р)

21. How Ansible works. URL: https://www.ansible.com/overview/how-ansible-works (дата звернення 01.12.2022 р)

22. Red Hat Ansible Automation Platform. URL: https://www.redhat.com/en/technologies/management/ansible (дата звернення 01.12.2022 р)

23. OpenStack Services. URL: https://www.openstack.org/software/project-navigator/openstack-components#openstack-services (дата звернення 05.12.2022 р)

24. The OpenStack Landscape. URL: https://www.openstack.org/software/ (дата звернення 05.12.2022 р)

25. Катренко Л.А., Катренко А.В. Охорона праці в галузі комп’ютингу. Львів: Магнолія-2006. 2012. 544 с.

26. Желібо Є. Безпека життєдіяльності. К.: 2001. 483 с.

Додаток А

Тези конференцій

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

М А Т Е Р І А Л И

X НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

7–8 грудня 2022 року

ТЕРНОПІЛЬ
2022

УДК 004.9
А. Луцків, І. Барна
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

ОСОБЛИВОСТІ ЗАДАЧ І ФУНКЦІЙ DEVOPS ФАХІВЦІВ

UDC 004.9
A. Lutskiv, M. Bondarenko

FEATURES OF TASKS AND FUNCTIONS OF DEVOPS SPECIALISTS

Головна задачам DevOps інженера – максимально збільшити передбачуваність, ефективність та безпеку розробки ПЗ.

Якщо розглядати повний життєвий цикл програмного забезпечення, то на етапі оцінки DevOps фахівці отримують первинну інформацію про необхідність нового кодування та внесення змін до IT-інфраструктури. На етапі проектування визначають вимоги до інфраструктури.

На етапі розробки та тестування займаються розгортанням продукту, а також підтримкою засобів для розробки, інтеграційним та навантажувальним тестуванням ПЗ для перевірки готовності операційного середовища.

Основна частина роботи DevOps інженера посідає етап випуску релізу — поставки продукту замовнику. У центрі уваги перебуває продуктивність всіх потоків процесу доставки. Такий фахівець слідкує за тим, щоб відомі баги ніколи не передавалися на наступний етап робіт, ніколи не розвивалася локальна оптимізація, що веде до створення глобальної деградації.

Рисунок 1. Місце DevOps в процесі розробки комп'ютерних систем

До обов'язків DevOps engineer входить:
- розгортання поставленого розробниками релізу у виробництві;
- інтеграція та поглиблення процесів розробки у постачання;
- стандартизація оточення розробки;
- налаштування інфраструктури на особливості пз, що розробляється;
- підготовка продуктивного середовища до частих внесення змін;
- виявлення та виправлення проблем;
- автоматизація процесів.

УДК 004.9
А. Луцків, І. Барна
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

АНАЛІЗ СЕРВІСНО-ОРІЄНТОВАНОЇ АРХІТЕКТУРИ В ПРОЦЕСІ ЗАСТОСУВАННЯ DEVOPS ПРАКТИК

UDC 004.9
A. Lutskiv, I. Barna

ANALYSIS OF SERVICE-ORIENTED ARCHITECTURE IN THE PROCESS OF APPLICATION OF DEVOPS PRACTICES

Сервісно-орієнтована архітектура деякий час також відома як централізовано орієнтована архітектура, це архітектура, де кілька послуг, які також називаються користувацькими агентами, використовують або створюють взаємодію з централізованою системою, щоб зробити традиційну монолітну архітектуру менш важкою і слабо зв'язаною. Архітектура, орієнтована на обслуговування, спеціально розроблена для обміну даними між системами. Кожен сервіс забезпечує функціональність на рівні абстракції, розглядається як чорний ящик, автономний, щоб уникнути зайвих витрат на кожну нову розробку (рис. 1).

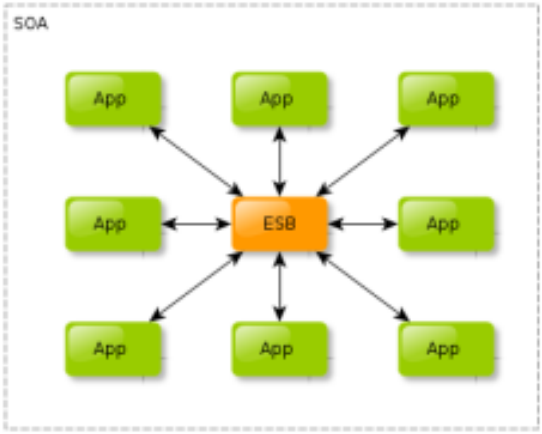

Рисунок 1. Сервісно-орієнтована архітектура

Сервіс абстрактний і може бути розроблений за допомогою будь-якої технології. Зв'язок між ними може здійснюватися за допомогою центральної точки підключення, яка називається Enterprise Service Bus. ESB піклується про всі сервіси та допомагає їм взаємодіяти один з одним.

SOA має свої плюси і мінуси, оскільки SOA надає можливість повторного використання всіх цих незалежних служб, наприклад, якщо потрібно створити модуль аутентифікації входу, замість того, щоб робити його з самого початку, можна використовувати службу аутентифікації facebook/google в SOA. Певним чином гетерогенні послуги SOA роблять систему відмовостійкістю. Якщо одна служба не працює, то система не перестане працювати. Найбільшою перевагою SOA є можливість повторного використання сервісу, якщо створено сервіс, то його завжди можна використовувати знову у нових розробках.