Міністерство освіти і науки України
Луцький національний технічний університет

Проектування інформаційних систем

Методичні вказівки до лабораторних робіт
для здобувачів першого (бакалаврського) рівня вищої освіти
освітньої програми «Комп’ютерні науки»
галузі знань 12 (F) Інформаційні технології
спеціальності 122 (F3) Комп’ютерні науки
денної та заочної форм навчання

Луцьк 2025

УДК 004.65 (07)
П-93

Рекомендовано до видання вченою радою факультету КІТ ЛНТУ,
протокол № від « ___» 2025 року

Голова вченої ради факультету КІТ І.С. Кондіус

Електронна копія друкованого видання передана для внесення в репозитарій ЛНТУ

Директор бібліотеки Н. П. Поліщук

Розглянуто і схвалено на засіданні кафедри комп'ютерних наук
ЛНТУ, протокол № від « » 2025 року

Завідувач кафедри КН В. О. Ліщина

Укладач: В. А. Кошелюк, кандидат технічних наук,
доцент кафедри комп'ютерних наук ЛНТУ

Рецензент: С. В. Лавренчук, кандидат технічних наук,
доцент кафедри комп'ютерної інженерії та
безпеки ЛНТУ

Відповідальний
за випуск: В. О. Ліщина, кандидат технічних наук,
завідувач кафедри комп'ютерних наук ЛНТУ

П-93 **Проектування інформаційних систем**: методичні вказівки до лабораторних робіт для здобувачів першого (бакалаврського) рівня вищої освіти освітньої програми «Комп'ютерні науки» галузі знань 12 (F) Інформаційні технології спеціальності 122 (F3) Комп'ютерні науки денної та заочної форм навчання / уклад. В.А. Кошелюк – Луцьк: ЛНТУ, 2025. 40 с.

Видання містить рекомендації до виконання лабораторних робіт з дисципліни «Проектування інформаційних систем».

Призначене для здобувачів першого (бакалаврського) рівня вищої освіти спеціальності 122 (F3) Комп'ютерні науки.

ЗМІСТ

1. ВСТУП

Лабораторна робота – вид навчального заняття, на якому здобувач під керівництвом науково-педагогічного (педагогічного) працівника і навчально-допоміжного персоналу особисто проводить натурні або імітаційні експерименти чи досліди з метою практичного підтвердження окремих теоретичних положень даної навчальної дисципліни (освітнього компонента), набуває практичних навичок роботи з лабораторним устаткуванням, обладнанням, обчислювальною технікою, вимірювальною апаратурою, методикою експериментальних досліджень у конкретній предметній галузі.

Лабораторні роботи проводяться у спеціально обладнаних навчальних лабораторіях з використанням устаткування, пристосованого до умов освітнього процесу (лабораторні макети, установки тощо). В окремих випадках лабораторні роботи можуть проводитися в умовах реального професійного середовища (наприклад, на виробництві, в наукових лабораторіях тощо). Лабораторна робота проводиться, як правило, зі здобувачами вищої освіти, чисельність яких не перевищує 15. Перелік тем лабораторних робіт визначається робочою програмою навчальної дисципліни. Заміна лабораторних занять іншими видами навчальних занять, як правило, не дозволяється. Для організації та проведення лабораторної роботи необхідно дотримуватися наступних умов:

– наявність спеціально обладнаних приміщень, устаткування, обладнання тощо;

– наявність навчально-методичного забезпечення з урахуванням специфіки занять та із застосуванням новітніх технологій;

– відповідність устаткування, обладнання, приладдя тощо вимогам охорони праці та санітарним нормам;

– необхідність проведення інструктажу здобувачів вищої освіти з питань охорони праці та безпеки, який підтверджується записами у журналі обліку;

– забезпечення матеріальними засобами;

– наявність елементів дослідження і творчого підходу при виконанні окремих завдань, створення наукових продуктів;

– наявність нормативно-методичної літератури.

Курс дисципліни. «Проектування інформаційних систем» має важливе значення в теоретичній підготовці майбутніх фахівців і є обов’язковою компонентою підготовки здобувачів першого (бакалаврського) рівня вищої освіти ОПП «Комп’ютерні науки» спеціальності 122 «Комп’ютерні науки».

Метою вивчення дисципліни «Проектування інформаційних систем» є формування у студентів теоретичних знань і практичних навичок з

розроблення, моделювання та впровадження сучасних інформаційних систем на базі хмарних сервісів Amazon Web Services (AWS). Особливу увагу приділено принципам архітектурного проєктування, безпеки, масштабованості та автоматизації процесів у середовищі AWS, що забезпечує підготовку фахівців до роботи в реальних умовах хмарної інфраструктури.

Предметом дисципліни є процеси, методи та інструменти проектування інформаційних систем із використанням сервісів AWS – від аналізу вимог та вибору архітектури до розгортання, тестування й підтримки систем у хмарному середовищі. До предметної області належать такі сервіси, як Amazon EC2, S3, RDS, Lambda, CloudFormation, IAM, VPC, CloudWatch тощо.

Основні завдання дисципліни:

– ознайомлення студентів із принципами побудови та життєвим циклом інформаційних систем у хмарних середовищах;

– вивчення архітектурних підходів до проектування систем (monolith, microservices, serverless, event-driven);

– формування практичних навичок використання основних сервісів AWS для створення інфраструктури та управління ресурсами;

– освоєння методів забезпечення надійності, безпеки, масштабованості та стійкості систем до відмов;

– розвиток умінь моделювати бізнес-процеси та застосовувати DevOps-підходи при розробленні систем;

– засвоєння принципів економічної доцільності, оптимізації витрат і моніторингу ресурсів у AWS.

Дисципліна забезпечує комплексне розуміння процесу проектування сучасних інформаційних систем у хмарних середовищах і сприяє формуванню компетенцій майбутніх фахівців у сфері IT-архітектури та управління інфраструктурою.

На лабораторних роботах кожен здобувач опрацьовує теоретичні відомості, виконуючи при цьому наведені в них приклади, а потім виконує завдання, оформляє звіт до латораторної роботи та дає відповіді на питання викладача (усно, з переліку контрольних питань). Виконання наступних робіт може опиратися на результати попередніх, тому не рекомендовано їх виконувати в довільному порядку.

Критерії оцінювання знань. Оцінювання лабораторних робіт із використанням контрольних запитань передбачає комплексну перевірку як практичних, так і теоретичних знань студента. Після виконання лабораторного завдання студент відповідає на набір контрольних запитань, що охоплюють основні поняття, алгоритми, методи та результати роботи. Відповіді студентів оцінюються від 0 до 2 балів. Оцінка формується на основі двох складових: правильності виконання практичної частини (1 бал) та точності відповідей на

контрольні запитання (1 бал). При цьому особлива увага приділяється логічності пояснень, обґрунтованості результатів і використанню коректної термінології. Такий підхід стимулює студента до глибшого розуміння матеріалу, розвиток аналітичного мислення та підвищує об'єктивність оцінювання лабораторної роботи.

Основними критеріями при оцінювання лабораторних робіт є: теоретична підготовка (розуміння основних понять та вміння пояснити теоретичні концепції); практичне виконання завдань (повнота та точність); технічна реалізація (правиьність структури налаштування, коректна організація коду, логічна огранізація скриптів тощо); дотримання термінів – оцінюється, наскільки здобувач дотримується дедлайнів і вчасно здає лабораторну роботу.

Для отримання максимального балу за лабораторну роботу також оцінюється інноваційність, ініціативність та креативність, наприклад чи застосовуються додаткові інструменти або методи, які не входять до завдання, але можуть покращити результат. Ці критерії дають змогу об'єктивно оцінити навички студента, його здатність до самостійної роботи, розуміння теорії та застосування знань на практиці.

Найважливішим інформаційним джерелом вивчення навчальної дисципліни «Проектування інформаційних систем» є ресурси мережі Інтернет. Основна частина матеріалу в Інтернеті розрахована на професіоналів, тому при вивченні навчальної дисципліни спочатку необхідно користуватися літературою навчального характеру.

Для зручності користування записами необхідно залишати поля для заміток і вільні рядки для доповнень. Записи не повинні бути одноманітними. В них необхідно виділяти важливі місця, головні слова, які акцентуються різним шрифтом або різним кольором шрифтів, підкреслюванням, замітками на полях, рамками, стовпчиками тощо. Записи можуть бути у вигляді конспекту, простих або розгорнутих тез, цитат, виписок, систематизованих таблиць, графіків, діаграм, схем.

Знання з дисципліни «Проектування інформаційних систем» становлять основу для подальшого поглибленого засвоєння матеріалу з того чи іншого розділу. З позицій випереджаючої освіти навчання тільки за конспектом лекцій і основною літературою, зазначеною у навчальній програмі, є недостатнім. У більшості випадків належна підготовка потребує вмінь швидко знаходити та опрацьовувати необхідний матеріал за першоджерелами, науковою і спеціальною літературою та коректно цитувати знайдене. Перелік такої літератури, як правило, наводиться у навчально-методичному комплексі навчальної дисципліни. Тому завдання студента зводиться до самостійного знаходження цих матеріалів шляхом пошуку у паперових або електронних фондах бібліотек.

Лабораторна робота № 1

Тема. Використання AWS Systems Manager

Мета : вивчення та розуміння основних компонентів сервісу керування, моніторингу та автоматизації роботи інфраструктури на прикладі AWS Systems Manager.

Література: [1, 5, 11, 20]

Теоретичні відомості

AWS Systems Manager – це комплексний сервіс від Amazon Web Services, призначений для централізованого керування, моніторингу та автоматизації роботи інфраструктури як у хмарі, так і в локальних середовищах. Він надає єдину консоль для адміністраторів, DevOps-інженерів і спеціалістів з безпеки, що дозволяє ефективно керувати сотнями або навіть тисячами серверів, віртуальних машин та інших ресурсів AWS.

Основна перевага AWS Systems Manager полягає у створенні єдиного центру управління, який об’єднує дані про стан систем, журнали подій, параметри конфігурації та оновлення. Завдяки цьому користувачі можуть швидко отримувати повну картину своєї інфраструктури, знаходити проблеми та реагувати на них без необхідності вручну входити на кожен сервер.

Серед ключових можливостей сервісу варто виділити:

– Automation – автоматизація повторюваних адміністративних завдань, таких як оновлення ПЗ, резервне копіювання чи зміна конфігурацій;

– Run Command – безпечне виконання команд одночасно на багатьох інстансах без необхідності підключення через SSH або RDP;

– Patch Manager – централізоване керування оновленнями систем, що забезпечує відповідність політикам безпеки;

– Parameter Store – безпечне збереження конфігураційних параметрів і секретів, які можна використовувати в інших сервісах AWS;

– Inventory та Compliance – збір даних про ресурси й контроль за дотриманням стандартів конфігурацій.

AWS Systems Manager допомагає організаціям зменшити операційні витрати, підвищити безпеку та стандартизувати керування інфраструктурою. Його інтеграція з іншими сервісами AWS, такими як CloudWatch, IAM, CloudTrail і EC2, робить його потужним інструментом для побудови стабільного, автоматизованого та безпечного середовища управління ресурсами у великих масштабах.

Завдання для виконання:

1. Створення списків інвентаризації для керованих екземплярів
2. Встановлення власної програми за допомогою команди «Run» у відповідності до рисунка 1.1.
3. Використання сховища параметрів для керування налаштуваннями програми
4. Використання диспетчера сеансів для доступу до екземплярів у відповідності до рисунка 1.2.

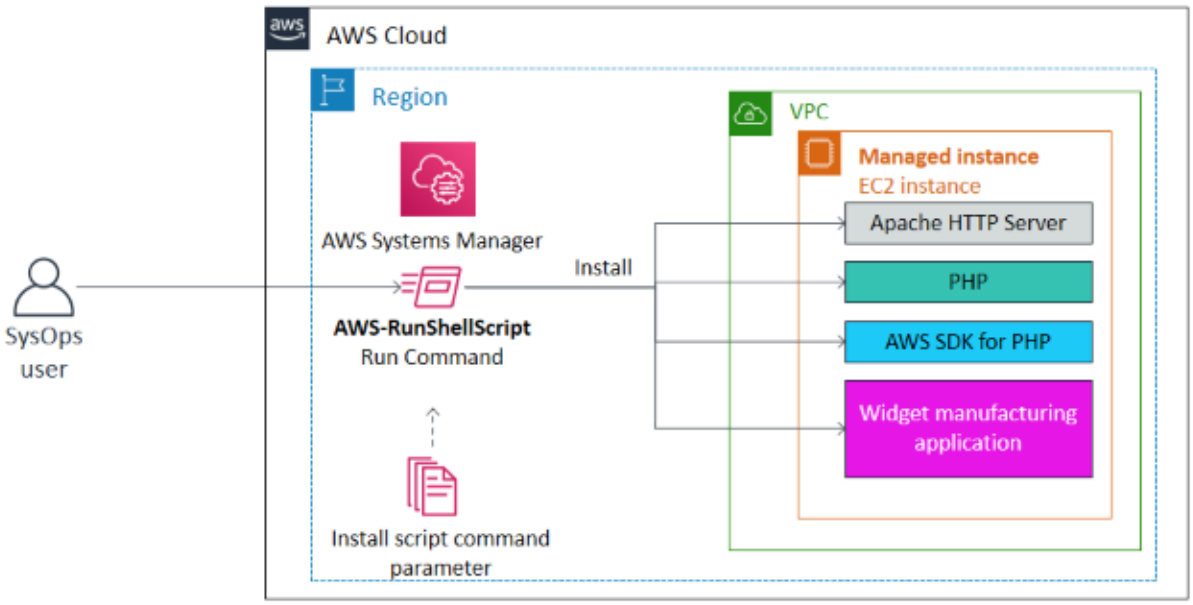

Рисунок 1.1 – Архітектура Systems Manager для RunShellScript

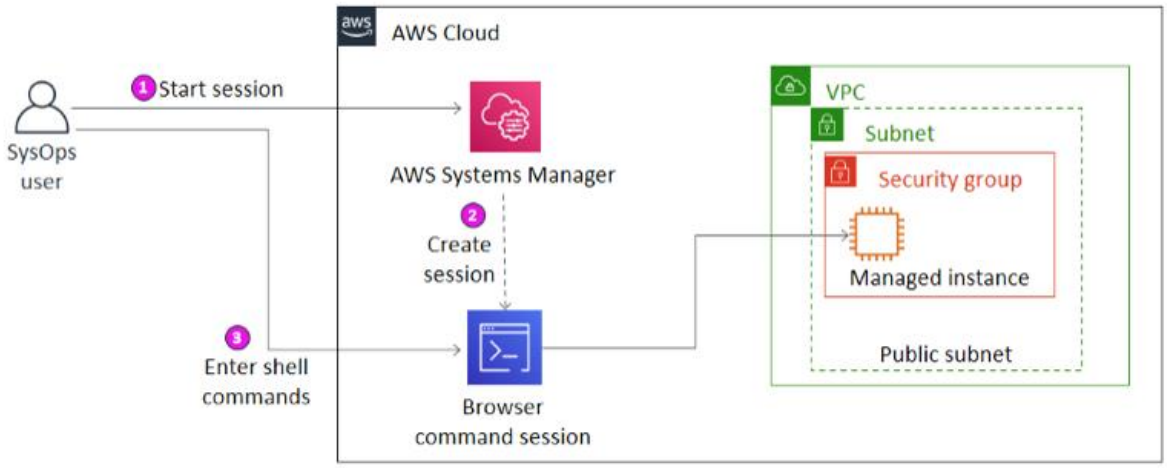

Рисунок 1.2 – Взаємодія Session Manager для Access Instances

Контрольні питання

1. Що таке AWS Systems Manager і які основні завдання він вирішує?
2. Які вимоги потрібно виконати, щоб EC2-інстанс міг керуватися через Systems Manager (SSM)?
3. Для чого використовується SSM Agent та як перевірити його роботу?
4. Що таке Systems Manager Parameter Store та які типи параметрів він підтримує?
5. Що таке Automation Runbooks і для яких сценаріїв вони застосовуються?
6. Як працює SSM Session Manager і які переваги він має порівняно з традиційним SSH-доступом?
7. Що дозволяє робити Systems Manager Patch Manager і як часто можна планувати оновлення?
8. Яку інформацію можна переглянути за допомогою Systems Manager Inventory?
9. Що таке OpsCenter і який тип інцидентів/подій він дозволяє агрегувати та аналізувати?

Лабораторна робота № 2

Тема. Створення екземплярів Amazon EC2

Мета : дослідження та вивчення розгортання екземпляра EC2 за допомогою командного рядка AWS.

Література: [2, 4, 12, 17]

Теоретичні відомості

Традиційні методи розгортання серверів часто є складними та ресурсоємними. Вони включають підготовку фізичного обладнання, встановлення операційних систем, налаштування мережевих параметрів та конфігурацію безпеки. Зазвичай цей процес вимагає координації кількох команд – від системних адміністраторів до мережевих інженерів і фахівців із безпеки – і може займати значний час, особливо якщо йдеться про масштабні проєкти. Крім того, ризики помилок при ручному налаштуванні завжди існують, що може призвести до вразливостей у системі та непередбачуваних затримок у роботі інфраструктури.

Сьогодні розгортання серверів стало значно простішим завдяки хмарним технологіям, зокрема сервісу Amazon EC2 (Elastic Compute Cloud). Amazon EC2 дозволяє створювати віртуальні сервери з необхідними параметрами операційної системи, об’ємом пам’яті, потужністю процесора та налаштуванням мережі за лічені хвилини. Це дає змогу швидко масштабувати інфраструктуру відповідно до потреб бізнесу без затримок, пов’язаних із фізичним обладнанням.

Для системного оператора надзвичайно важливо володіти знаннями щодо автоматизації більшості процесів у хмарному середовищі за допомогою інтерфейсу командного рядка AWS (AWS CLI). Використання AWS CLI дозволяє ефективно створювати та конфігурувати екземпляри EC2, налаштовувати правила безпеки, управляти мережевими ресурсами та контролювати доступ за допомогою ключів доступу. Автоматизація цих операцій значно знижує ризик людських помилок, підвищує загальний рівень безпеки та забезпечує послідовність налаштувань у різних середовищах – розробки, тестування та продуктивної експлуатації.

Застосування AWS CLI дозволяє системним операторам швидко реагувати на зміни в інфраструктурі, легко масштабувати ресурси та впроваджувати нові сервіси без значних витрат часу на ручне налаштування. Крім того, автоматизація допомагає стандартизувати конфігурації, що особливо важливо у великих або динамічних середовищах, де важко підтримувати однорідність налаштувань вручну. Наприклад, створення шаблонів запуску екземплярів EC2 або автоматизоване налаштування груп безпеки дозволяє швидко відтворювати середовища з однаковими параметрами, забезпечуючи стабільну роботу сервісів.

Сучасні хмарні платформи, такі як AWS, роблять процес управління серверами більш ефективним, прозорим і керованим. Навички роботи з EC2, управління безпекою та автоматизації за допомогою AWS CLI стають критично важливими для системних операторів, які прагнуть підтримувати високий рівень доступності, надійності та продуктивності інфраструктури. Володіння

цими інструментами дозволяє не лише економити час, але й забезпечувати масштабованість, гнучкість та безперервність бізнес-процесів у будь-якому масштабі.

Завдання для виконання:

1. Створення та запуск віртуального сервера Amazon EC2 через консоль управління AWS.
2. Розгортання екземпляра EC2 за допомогою командного рядка AWS CLI у відповідності до рисунку 2.
3. Підключення до екземпляра Amazon EC2.
4. Виправлення інсталяції веб-сервера.

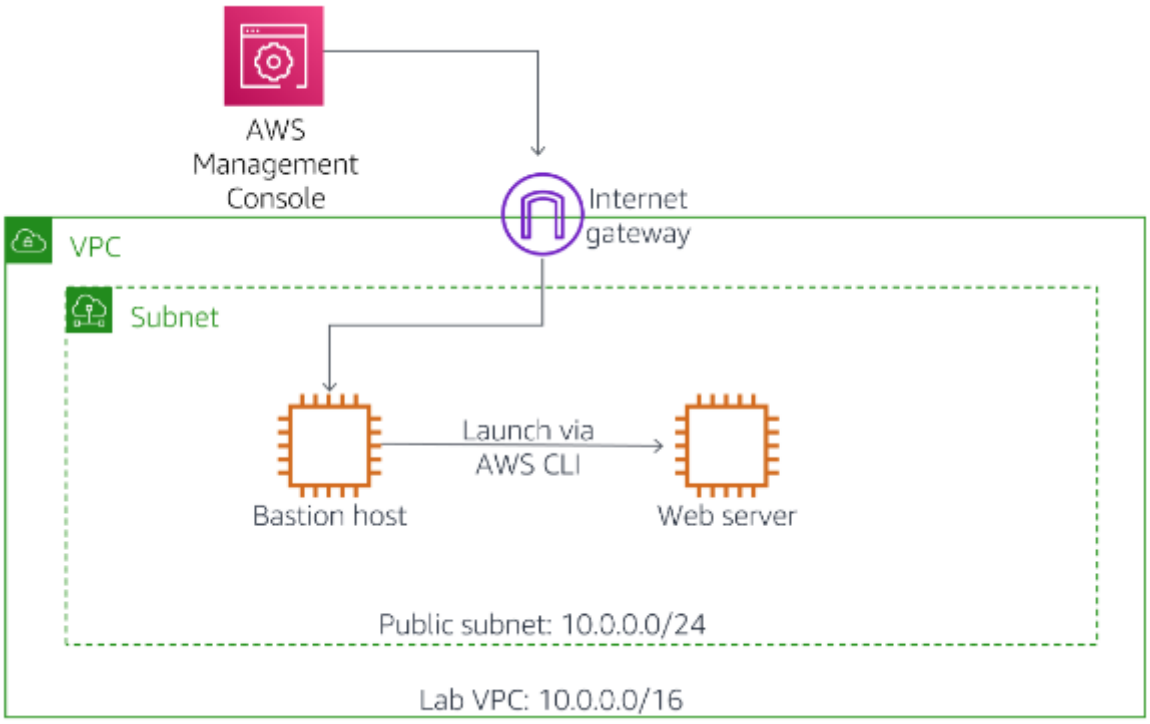

Рисунок 2 – Схема розгортання екземпляра EC2

Контрольні питання

1. Що таке Amazon EC2 і для чого використовуються екземпляри (instances)?
2. Які типи інстансів EC2 існують та чим вони відрізняються (General Purpose, Compute Optimized, Memory Optimized тощо)?
3. Що таке Amazon Machine Image і яку роль відіграє при створенні інстансу?
4. Які кроки потрібно виконати під час створення нового EC2-екземпляру через AWS Management Console?
5. Що таке key pair у EC2 і для чого він потрібен під час створення інстансу?
6. Яким чином вибір типу диску EBS (gp3, io1, sc1) впливає на роботу інстансу?
7. Що таке Security Group і яку роль відіграють правила Inbound/Outbound при створенні EC2-екземпляру?
8. Чим відрізняється Public IP від Elastic IP при запуску EC2-інстансу?
9. Яка різниця між EC2 On-Demand, Reserved Instances та Spot Instances?
10. Які дії потрібно виконати для підключення до EC2-екземпляра через SSH (Linux) або RDP (Windows)?

Лабораторна робота № 3

Тема. Огляд AWS CloudShell та IDE.

Мета : навчитися використовувати CloudShell та IDE.

Література: [3, 8, 13, 19]

Теоретичні відомості

Сучасна розробка програмного забезпечення дедалі більше переходить у хмарні середовища, що забезпечують гнучкість, масштабованість та доступність інструментів із будь-якого пристрою. Одним із таких інструментів є AWS CloudShell – веб-інтерфейс для роботи з хмарними ресурсами Amazon Web Services без потреби встановлення додаткового програмного забезпечення на локальний комп'ютер. CloudShell надає розробнику доступ до командного рядка Linux із попередньо налаштованими AWS CLI, Python, Node.js та іншими популярними утилітами, що значно спрощує роботу з сервісами AWS.

Основною перевагою CloudShell є швидкий доступ до середовища розробки, яке інтегрується з обліковим записом AWS, забезпечуючи автоматичне підтвердження прав доступу до ресурсів. Це особливо корисно при тестуванні скриптів для управління S3, EC2, Lambda та іншими сервісами. Крім того, CloudShell дозволяє зберігати файли між сеансами, що забезпечує зручність у роботі над проектами.

Разом із CloudShell, для ефективної розробки в хмарному середовищі особливо корисно використовувати інтегровані середовища розробки (IDE, Integrated Development Environment). Сучасні IDE, такі як AWS Cloud9, пропонують розширені функціональні можливості, що значно полегшують процес програмування. До таких можливостей належать автодоповнення коду, підсвічування синтаксису, відлагодження (debugging) та інтегрований контроль версій за допомогою Git. Це дозволяє розробникам швидко виявляти помилки, оптимізувати код і відслідковувати зміни без необхідності виходити з середовища розробки.

Особливу цінність надає тісна інтеграція IDE з хмарними сервісами AWS. Завдяки цьому розробник може безпосередньо запускати і тестувати додатки у хмарі, отримуючи миттєвий зворотний зв'язок. Наприклад, можна відразу перевірити роботу серверних функцій, API або баз даних, не витрачаючи час на локальну конфігурацію або розгортання інфраструктури вручну. Крім того, Cloud9 дозволяє працювати з терміналом AWS CloudShell прямо всередині IDE, що забезпечує повний контроль над хмарними ресурсами і спрощує управління ними.

Поєднання AWS CloudShell та IDE створює потужне і зручне середовище для розробки хмарних додатків. Це не тільки підвищує продуктивність і скорочує час розробки, але й мінімізує залежність від локальної конфігурації робочого середовища. Розробники можуть швидко створювати, тестувати та розгортати додатки, одночасно ефективно керуючи ресурсами в хмарі. У сучасній практиці розробки програмного забезпечення таке поєднання стає ключовим елементом, який дозволяє командам швидше адаптуватися до змін, підвищувати якість коду та забезпечувати стабільність і надійність хмарних рішень.

Завдання для виконання:

1. Підключитися до AWS CloudShell та виконати команди інтерфейсу командного рядка AWS (AWS CLI) та код AWS SDK з нього.
2. Підключитися до VS Code IDE та дослідити його функціональність у відповідності до рисунку 3.
3. Копіювати файли до та з Amazon Simple Storage Service (Amazon S3), CloudShell та VS Code IDE.
4. Встановити AWS SDK для Python (Boto3) на екземпляр VS Code IDE.
5. Використовувати VS Code IDE для створення та запуску файлів коду.

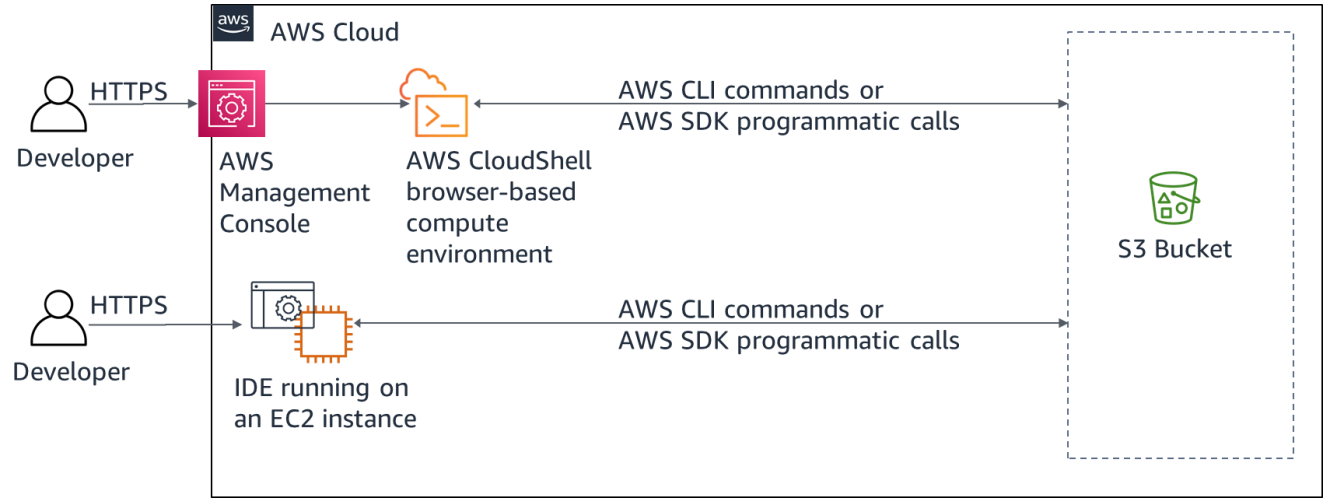

Рисунок 3 – Механізм взаємодії CloudShell та IDE

Контрольні питання

1. Що таке AWS CloudShell і для чого він використовується?
2. Які основні переваги використання AWS CloudShell порівняно з локальним терміналом?
3. У яких регіонах доступний AWS CloudShell і чому це важливо враховувати?
4. Які інструменти та утиліти попередньо встановлені в середовищі AWS CloudShell?
5. Як працює збереження даних та файлів у CloudShell (обсяг диску, персистентність)?
6. Як виконати автентифікацію та отримати доступ до AWS CLI в CloudShell?
7. Які IDE надає AWS для розробників (наприклад, AWS Cloud9) і які їх ключові особливості?
8. Чим AWS CloudShell відрізняється від AWS Cloud9 як інструментів для розробки?
9. Як у Cloud9 налаштовуються середовища розробки та керування залежностями?
10. Які обмеження та ліміти існують при використанні AWS CloudShell і як вони впливають на робочий процес?

Лабораторна робота № 4

Тема. Налаштування та використання хмарного сховища Amazon S3

Мета : розглянути команди налаштування хмарного сховища Amazon S3.

Література: [4, 6, 14, 17]

Теоретичні відомості

У сучасному світі зростаючих обсягів даних надійне та гнучке зберігання інформації стає критично важливим для бізнесу та приватних користувачів. Одним із найпопулярніших рішень для зберігання даних у хмарі є Amazon Simple Storage Service (Amazon S3), що забезпечує масштабованість, безпеку та доступність даних у будь-який час.

Налаштування Amazon S3 починається зі створення облікового запису в AWS (Amazon Web Services). Після реєстрації користувач отримує доступ до консолі управління, де можна створювати «бакети» — основні контейнери для зберігання файлів. При створенні бакету важливо обрати регіон розміщення даних, оскільки це впливає на швидкість доступу та вартість зберігання. Крім того, можна налаштувати версіонування файлів, що дозволяє зберігати кілька версій одного об’єкта та відновлювати їх у разі випадкового видалення або змін.

Безпека даних у S3 забезпечується за допомогою політик доступу та шифрування. AWS дозволяє налаштовувати права доступу на рівні бакету або окремого об’єкта, використовуючи IAM (Identity and Access Management), а також шифрування даних як під час збереження, так і під час передачі. Це дозволяє гарантувати, що доступ до конфіденційної інформації матимуть лише уповноважені користувачі.

Amazon S3 (Simple Storage Service) є потужним хмарним сервісом для зберігання великих обсягів даних, який надає користувачам широкий спектр можливостей для управління файлами та інтеграції з іншими сервісами AWS. Основні операції з S3 включають завантаження та завантаження файлів, організацію структури сховища у вигляді бакетів, а також детальне управління правами доступу для окремих користувачів або груп. Це дозволяє забезпечити безпеку даних і гнучко контролювати, хто і яким чином може отримувати доступ до інформації.

S3 підтримує різні методи доступу до об’єктів: через веб-консоль, командний рядок або програмний інтерфейс (API). Завдяки цьому користувачі можуть легко інтегрувати сховище у власні автоматизовані процеси, наприклад, для резервного копіювання баз даних, логів сервісів або критично важливих файлів. Така автоматизація дозволяє значно зменшити людський фактор і підвищити ефективність управління даними.

Ще однією важливою особливістю є інтеграція з іншими сервісами AWS. Наприклад, S3 можна поєднати з Amazon CloudFront для швидкого та глобального розповсюдження контенту користувачам по всьому світу, або використовувати разом з AWS Lambda для обробки даних без необхідності запускати власні сервери. Це робить S3 не просто сховищем, а гнучкою платформою для побудови масштабованих та безпечних рішень у хмарі.

Таким чином, Amazon S3 є надійним інструментом для організацій будь-якого масштабу. Його основні переваги включають високу масштабованість, гнучкі можливості управління доступом, інтеграцію з іншими хмарними сервісами та простоту налаштування. Використання S3 дозволяє забезпечити безперебійну доступність даних, підвищити ефективність бізнес-процесів і створює міцну основу для розвитку хмарної інфраструктури організації.

Завдання для виконання:

1. Підключення до IDE та налаштування середовища.
2. Створення корзини S3 за допомогою AWS CLI.
3. Налаштування політики корзини для корзини за допомогою SDK для Python у відповідності до рисунку 4.
4. Завантаження об’єктів до корзини для створення веб-сайту.
5. Тестування доступу до веб-сайту.
6. Аналіз коду веб-сайту.

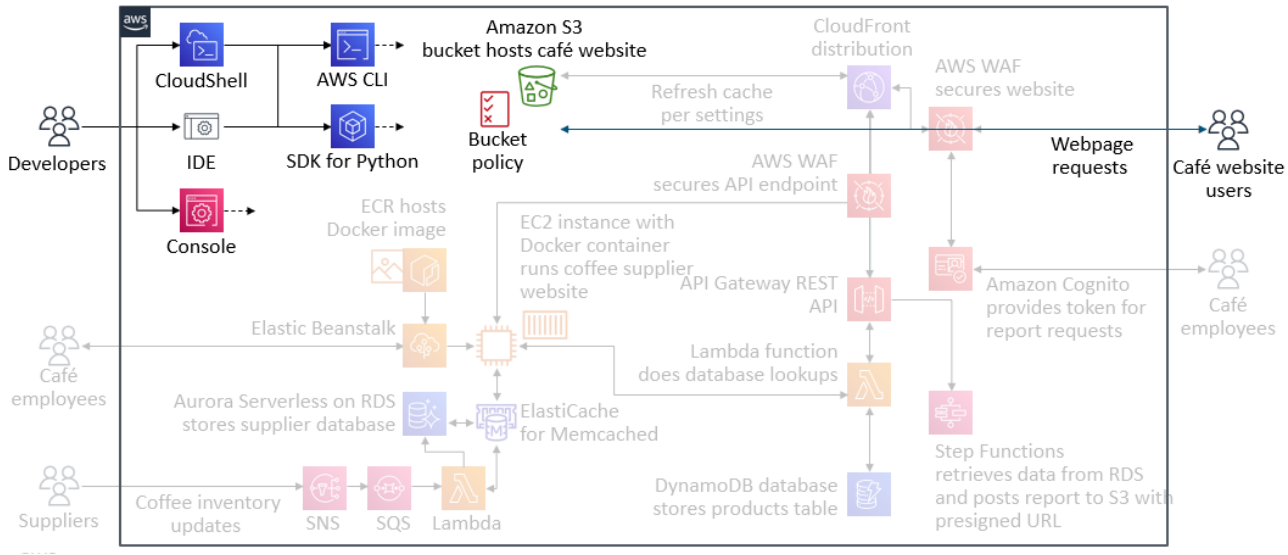

Рисунок 4 – Amazon S3 як частина розробки хмарного застосунку

Контрольні питання

1. Що таке Amazon S3 і для яких завдань воно найчастіше використовується?
2. У чому різниця між Bucket та Object у S3?
3. Які класи зберігання (Storage Classes) доступні в Amazon S3 та в чому їхня відмінність?
4. Що таке S3 Bucket Policy і які завдання вона вирішує?
5. Яка різниця між Bucket Policy та ACL (Access Control List)?
6. Як працює Versioning в S3 і для чого його використовують?
7. Що таке S3 Lifecycle Rules і як вони допомагають оптимізувати витрати?
8. Які способи автентифікації та авторизації використовуються для доступу до S3?
9. Як можна налаштувати статичний веб-сайт у S3 Bucket?
10. У чому полягає принцип роботи S3 Encryption (SSE-S3, SSE-KMS, SSE-C)?

Лабораторна робота № 5

Тема. Створення та керування таблицями у Amazon DynamoDB

Мета : ознайомитися з основами створення та керування таблицями у Amazon DynamoDB та набути практичних навичок роботи з даними у цій базі.

Література: [5, 9, 15, 20]

Теоретичні відомості

Amazon DynamoDB – це високопродуктивна та повністю керована NoSQL база даних, яка дозволяє зберігати та обробляти великі обсяги даних із мінімальною затримкою. Однією з ключових переваг DynamoDB є його здатність масштабуватися горизонтально, забезпечуючи високу доступність та надійність без потреби у налаштуванні серверної інфраструктури. Створення та керування таблицями в цій системі є основним аспектом ефективного використання бази даних.

Процес створення таблиці у DynamoDB починається з визначення основних параметрів: ім’я таблиці, ключів та опційних параметрів індексування. Кожна таблиця повинна мати первинний ключ, який може бути простим (partition key) або складним (partition key + sort key). Первинний ключ визначає спосіб організації даних у таблиці та забезпечує швидкий доступ до них. Наприклад, у таблиці користувачів partition key може бути UserID, a sort key – Timestamp, що дозволяє зберігати історію дій користувачів.

Після створення таблиці користувачі можуть додавати, змінювати та видаляти записи за допомогою операцій PutItem, UpdateItem, GetItem та DeleteItem. DynamoDB підтримує також масові операції (BatchGetItem, BatchWriteItem), що робить роботу з великими обсягами даних більш ефективною. Крім того, база даних дозволяє створювати глобальні та локальні вторинні індекси, які допомагають оптимізувати запити та зменшити навантаження на первинний ключ.

Управління таблицями включає налаштування пропускної здатності та політик масштабування. DynamoDB підтримує два режими: provisioned та on-demand. У режимі provisioned користувач сам визначає кількість одиниць читання та запису, що підходить для передбачуваних навантажень. Режим on-demand автоматично масштабуватиме таблицю відповідно до запитів, що зручно для динамічних або непередбачуваних робочих навантажень.

Для адміністрування DynamoDB надає AWS Management Console, CLI та SDK для різних мов програмування, що робить керування таблицями гнучким і зручним. Додатково можна налаштовувати моніторинг за допомогою Amazon CloudWatch та політики резервного копіювання для забезпечення безпеки та відновлення даних.

Таким чином, створення та керування таблицями у Amazon DynamoDB поєднує простоту використання, масштабованість і надійність. Ретельне планування ключів, індексів та пропускної здатності дозволяє ефективно обробляти великі обсяги даних і забезпечує високу продуктивність додатків у сучасних хмарних середовищах.

Завдання для виконання:

1. Створення таблиці DynamoDB за допомогою SDK для Python.
2. Робота з даними DynamoDB – розуміння умовних виразів DynamoDB (рис. 5).
3. Додавання та зміна одного елемента за допомогою SDK.
4. Додавання кількох елементів за допомогою SDK та пакетної обробки.
5. Запит до таблиці за допомогою SDK.
6. Додавання глобального вторинного індексу до таблиці.

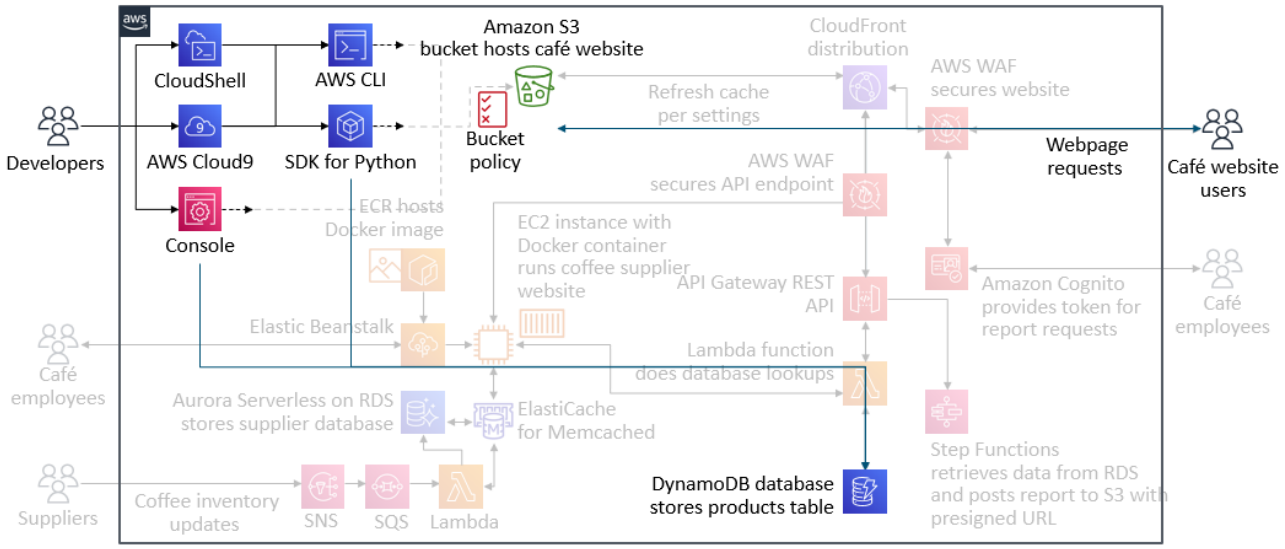

Рисунок 5 – NoSQL бази даних як частина розробки хмарного застосунку

Контрольні питання

1. Що таке таблиця в Amazon DynamoDB та які основні компоненти вона містить?
2. Які типи ключів доступні в DynamoDB та в чому різниця між Partition Key і Sort Key?
3. Які вимоги до створення таблиці в DynamoDB (обов'язкові параметри)?
4. Що таке RCU та WCU, і як вони впливають на продуктивність таблиці?
5. У чому різниця між режимами пропускної здатності On-Demand та Provisioned?
6. Що таке Global Secondary Index (GSI) та Local Secondary Index (LSI)? Які між ними відмінності?
7. Як здійснюється додавання, оновлення та видалення елементів у таблиці DynamoDB?
8. Які інструменти AWS дозволяють керувати таблицями DynamoDB (CLI, SDK, Console)?
9. Що таке TTL (Time to Live) у DynamoDB і для чого він використовується?
10. Які типи операцій читання і запису доступні в DynamoDB (GetItem, PutItem, UpdateItem, DeleteItem, Query, Scan) та чим Query відрізняється від Scan?

Лабораторна робота № 6

Тема. Розробка REST API за допомогою Amazon API Gateway.

Мета : ознайомитися з принципами побудови та розгортання REST API у хмарному середовищі за допомогою сервісу Amazon API Gateway.

Література: [6, 8, 16, 21]

Теоретичні відомості

У сучасному світі веброзробки REST API стали основою взаємодії між клієнтами та серверними додатками. Вони дозволяють створювати масштабовані, безпечні та ефективні сервіси для обміну даними. Одним із найзручніших інструментів для побудови таких API є Amazon API Gateway – керований сервіс AWS, який дає змогу легко створювати, розгортати та управляти REST API будь-якого масштабу.

Amazon API Gateway надає розробникам можливість створювати API без необхідності підтримувати власні сервери. Це дозволяє зосередитись на бізнес-логіці, а не на інфраструктурі. Сервіс виступає як «вхідна точка» для клієнтських запитів, що пересилаються до відповідних бекенд-сервісів – наприклад, до AWS Lambda, EC2, Elastic Beanstalk або навіть до зовнішніх HTTP-ендпоїнтів. Така архітектура ідеально підходить для serverless-підходу, де серверна частина виконується лише у відповідь на запити, що суттєво знижує витрати.

Процес розробки REST API в API Gateway зазвичай починається зі створення ресурсів і методів. Ресурсом може бути, наприклад, об’єкт «/users» або «/orders», а методами – HTTP-операції (GET, POST, PUT, DELETE). Далі кожен метод зв’язується з бекенд-інтеграцією – це може бути виклик функції Lambda або звернення до іншого API. Потім розробник може налаштувати валидацію запитів, авторизацію (через IAM, Cognito або власні токени), а також кешування відповідей для підвищення продуктивності.

Однією з головних переваг Amazon API Gateway є гнучке управління безпекою та масштабуванням. Сервіс автоматично обробляє тисячі запитів на секунду, застосовуючи ліміти швидкості та політики доступу. Крім того, він забезпечує детальний моніторинг через Amazon CloudWatch, що дозволяє відстежувати продуктивність і виявляти потенційні проблеми.

Amazon API Gateway виділяється як одне з найпотужніших рішень для створення REST API завдяки глибокій інтеграції з екосистемою AWS. Цей сервіс органічно поєднується з широким спектром AWS-інструментів, що відкриває нові можливості для архітектури додатків.

Особливо ефективною є інтеграція з AWS Lambda, яка дозволяє реалізувати повністю безсерверну архітектуру. Така комбінація усуває необхідність управління серверною інфраструктурою, забезпечуючи автоматичне масштабування залежно від навантаження. Розробники можуть зосередитися виключно на бізнес-логіці, тоді як AWS бере на себе всі операційні аспекти.

Крім Lambda, API Gateway seamlessly інтегрується з DynamoDB для зберігання даних, Amazon Cognito для автентифікації користувачів, CloudWatch для моніторингу та логування, а також з S3, Step Functions та іншими сервісами.

Це створює цілісну екосистему для побудови складних мікросервісних архітектур.

Підсумовуючи, Amazon API Gateway є оптимальним вибором для розробки сучасних REST API у хмарному середовищі. Він успішно поєднує інтуїтивність налаштування, гнучкість інтеграційних можливостей та надійність перевіреної інфраструктури AWS. Ці характеристики дозволяють командам швидко створювати безпечні, масштабовані та високопродуктивні веб-сервіси, мінімізуючи технічні складнощі та операційні витрати.

Завдання для виконання:

1. Підготовка середовища розробки у відповідності до рисунку 6.
2. Створення першої кінцевої точки API (GET).
3. Створення другої кінцевої точки API (GET).
4. Створення третьої кінцевої точки API (POST).
5. Розгортання API.
6. Оновлення веб-сайту для використання API.

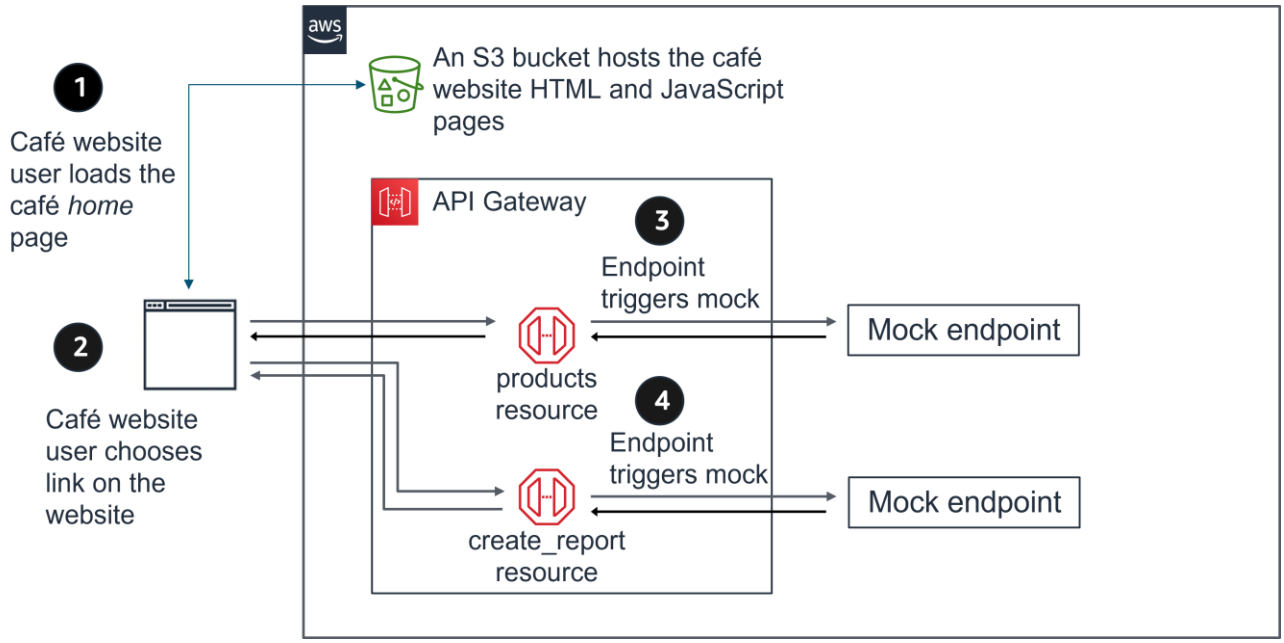

Рисунок 6 – Середовище Amazon API Gateway

Контрольні питання

1. Що таке Amazon API Gateway і яку роль він виконує у розробці REST API?
2. Які типи API підтримує Amazon API Gateway? У чому відмінність між REST API та HTTP API?
3. Поясніть різницю між resource та method у Amazon API Gateway.
4. Що таке інтеграція (integration) у API Gateway і які типи інтеграцій доступні?
5. Як API Gateway забезпечує безпеку API? Назвіть принаймні три механізми контролю доступу.
6. Поясніть, як можна реалізувати обмеження швидкості (throttling) та квоти використання (quotas) для API.
7. Як здійснюється моніторинг та логування REST API у Amazon API Gateway?

Лабораторна робота № 7

Тема. Створення лямбда-функцій за допомогою AWS SDK для Python.

Мета : Ознайомитися з принципами роботи безсерверних обчислень у хмарному середовищі AWS та набути практичних навичок створення, налаштування та виклику AWS Lambda-функцій за допомогою AWS SDK для Python (boto3).

Література: [3, 7, 17, 22]

Теоретичні відомості

У сучасному світі хмарних обчислень автоматизація та масштабованість є ключовими факторами ефективності. Одним із найпотужніших інструментів, які пропонує Amazon Web Services (AWS), є AWS Lambda – сервіс безсерверних обчислень, що дозволяє запускати код у відповідь на події без необхідності керувати серверами. Одним зі способів взаємодії з Lambda є AWS SDK для Python, або boto3, який дає змогу програмно створювати, оновлювати й управляти функціями.

Створення лямбда-функцій за допомогою boto3 відкриває широкі можливості для DevOps-інженерів, розробників і дослідників, які прагнуть автоматизувати процеси розгортання та інтеграції. Основна перевага полягає в тому, що лямбда-функції можна створювати безпосередньо з Python-коду, без потреби у використанні вебконсолі AWS. Це спрощує CI/CD-процеси, дозволяє масштабувати інфраструктуру як код (Infrastructure as Code) і забезпечує гнучкість при розробці мікросервісів.

Для початку роботи необхідно встановити бібліотеку boto3 і налаштувати облікові дані AWS. Після цього користувач може створити клієнт для Lambda:

```
import boto3
lambda_client = boto3.client('lambda')
```

Далі розробник готує архів із кодом функції (наприклад, function.zip), що містить файл lambda_function.py з обробником подій. Потім можна створити функцію за допомогою методу create_function.

Одним із важливих аспектів є роль IAM, яка визначає, які дії дозволено виконувати функції. Без належних дозволів створення або запуск Lambda не відбудеться. Тому інтеграція boto3 з IAM є ключовою частиною безпечної автоматизації. Після створення функції можна оновлювати її код або параметри без повного видалення. Це виконується через метод update_function_code, що дозволяє безперервно вдосконалювати застосунок у межах CI/CD-процесу. Крім того, boto3 дозволяє викликати функцію безпосередньо з Python-коду.

Таким чином, AWS SDK для Python стає універсальним інструментом як для розробки, так і для експлуатації. У підсумку, створення лямбда-функцій за допомогою AWS SDK для Python – це ефективний підхід до автоматизації хмарних процесів. Він об’єднує силу безсерверних технологій з простотою та гнучкістю мови Python. Завдяки boto3 можна швидко реалізувати масштабовані рішення, інтегровані з іншими сервісами AWS, такими як S3, DynamoDB чи CloudWatch. Такий підхід дозволяє не лише скоротити витрати на інфраструктуру, а й підвищити продуктивність команди, що працює над створенням сучасних безсерверних архітектур.

Завдання для виконання:

1. Підготовка середовища розробки у відповідності до рисунку 7.
2. Створення лямбда-функції get_all_products.
3. Налаштування методу GET REST API для виклику лямбда-функції.
4. Створення лямбда-функції create_report.
5. Налаштування методу POST REST API для виклику лямбда-функції
6. Тестування інтеграції за допомогою веб-сайту кафе.

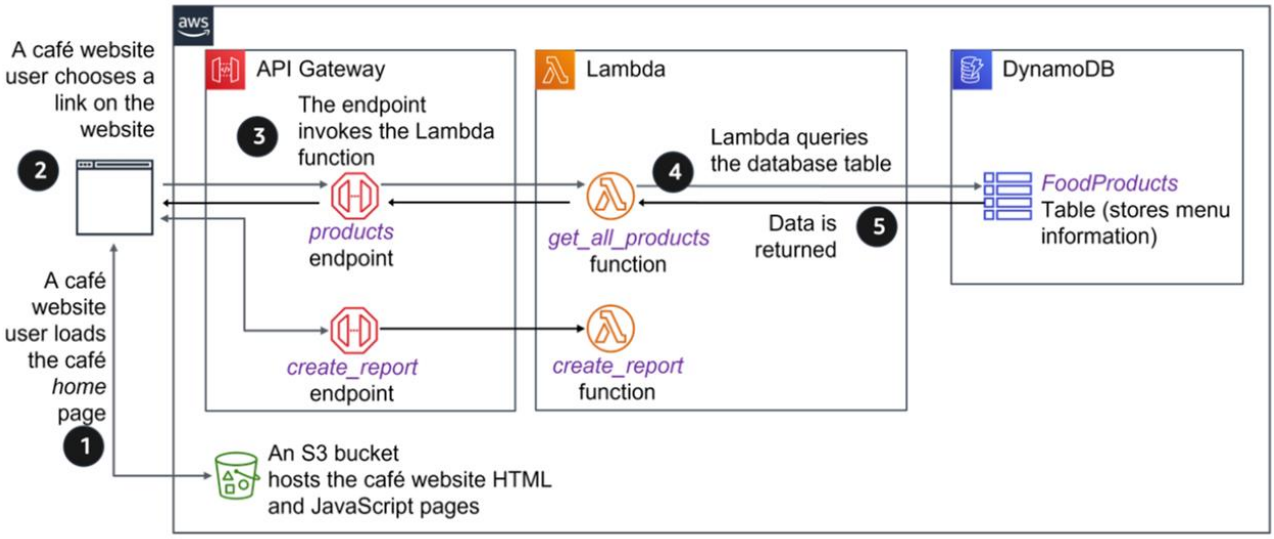

Рисунок 7 – AWS SDK для Python (boto3) для створення функцій AWS Lambda

Контрольні питання

1. Що таке AWS Lambda і для чого використовують лямбда-функції?
2. Який модуль AWS SDK для Python використовується для роботи з Lambda-функціями?
3. Опишіть кроки створення лямбда-функції за допомогою boto3.
4. Які основні параметри необхідно вказати при виклику методу create_function в boto3?
5. Як прикріпити роль IAM до Lambda-функції під час її створення через Python?
6. Як завантажити код лямбда-функції з локального файлу ZIP при створенні через boto3?
7. Що таке Handler у Lambda-функції і як його вказати при створенні функції в boto3?
8. Як перевірити статус створеної Lambda-функції через boto3 після виклику create_function?
9. Як оновити код існуючої Lambda-функції за допомогою Python SDK?
10. Які обмеження або важливі аспекти слід враховувати при створенні лямбда-функцій через AWS SDK для Python?

Лабораторна робота № 8

Тема. Міграція веб-застосунку до контейнерів Docker.

Мета : набути практичних навичок контейнеризації веб-застосунку шляхом його міграції до середовища Docker.

Література: [2, 5, 18, 21]

Теоретичні відомості

У сучасному світі розробки програмного забезпечення швидкість, масштабованість та надійність системи стають ключовими критеріями успіху. Традиційні підходи до розгортання веб-застосунків часто стикаються з проблемами сумісності, складного управління залежностями та високих витрат на інфраструктуру. Саме тому все більше компаній обирають контейнери як засіб модернізації своїх систем. Одним із найпопулярніших рішень у цій сфері є Docker, який дозволяє ізолювати застосунок разом із його середовищем, забезпечуючи портативність та відтворюваність.

Міграція веб-застосунку до Docker починається з аналізу його архітектури. Важливо визначити всі компоненти системи: веб-сервер, базу даних, кеш, служби аутентифікації та інші залежності. Традиційні застосунки, що працюють на фізичних серверах або віртуальних машинах, часто мають складну взаємозалежність бібліотек і системних пакетів, що ускладнює перенесення на нову платформу. Docker дозволяє створити ізольоване середовище, у якому всі залежності пакуються разом із застосунком у єдиний контейнер. Це значно зменшує ризик конфліктів версій та несподіваних збоїв при розгортанні.

Наступним етапом є створення Dockerfile – конфігураційного файлу, що описує процес побудови образу контейнера. У Dockerfile визначаються базовий образ операційної системи, встановлення необхідних бібліотек та запуск веб-застосунку. Наприклад, для застосунку на Python часто обирають образ python:3.11, встановлюють залежності через pip і копіюють код у контейнер. Dockerfile забезпечує повторюваність та контроль версій середовища, що особливо цінно для команд розробки та DevOps.

Після створення образу веб-застосунок можна запускати у контейнері локально для тестування. На цьому етапі перевіряють, чи працює застосунок коректно, чи всі сервіси взаємодіють один з одним, та чи забезпечено безпеку даних. Docker також підтримує мультиконтейнерні середовища за допомогою Docker Compose, що дозволяє одночасно запускати базу даних, кеш та веб-сервер у взаємопов’язаних контейнерах.

Міграція до контейнерів відкриває додаткові переваги для масштабування та управління інфраструктурою. Використовуючи оркестратори, такі як Kubernetes, можна автоматично масштабувати веб-застосунок залежно від навантаження, забезпечувати високу доступність та балансування трафіку. Крім того, контейнери полегшують інтеграцію з CI/CD-пайплайнами, що дозволяє швидко доставляти нові версії програмного забезпечення користувачам.

Водночас, міграція до Docker потребує уважного підходу до безпеки та моніторингу. Контейнери вимагають регулярного оновлення базових образів, обмеження доступу до внутрішніх сервісів і контролю за ресурсами. Без

належного спостереження існує ризик витоків даних або перевантаження системи. Тому важливо інтегрувати інструменти логування та моніторингу, такі як Prometheus або ELK Stack, у контейнеризоване середовище.

Таким чином, міграція веб-застосунку до Docker дозволяє значно підвищити гнучкість, масштабованість та надійність системи. Контейнери ізолюють середовище застосунку, забезпечують відтворюваність і полегшують інтеграцію з сучасними DevOps-практиками. Хоча процес міграції потребує ретельного планування та уваги до безпеки, його вигоди роблять Docker ключовим інструментом для сучасної розробки та підтримки веб-застосунків.

Завдання для виконання:

1. Підготовка середовища розробки у відповідності до рисунку 8.
2. Аналіз існуючої інфраструктури програми
3. Міграція програми до контейнера Docker
4. Міграція бази даних MySQL до контейнера Docker
5. Тестування контейнера MySQL за допомогою програми node
6. Додавання образів Docker до реєстру Amazon Elastic Container (Amazon ECR)

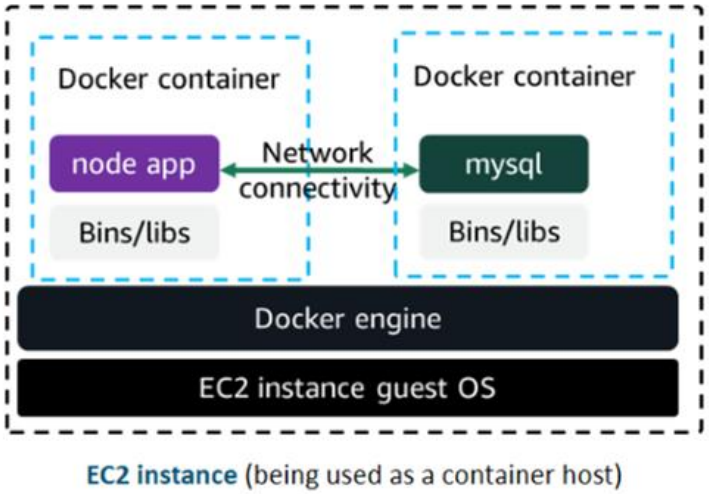

Рисунок 8 – Схема міграції

Контрольні питання

1. Що таке Docker і які переваги його використання для веб-застосунків?
2. Які основні компоненти Docker необхідно знати для міграції веб-застосунку (Dockerfile, образи, контейнери, Docker Compose)?
3. Як створюється Dockerfile для веб-застосунку і які базові інструкції в ньому використовуються?
4. Що таке Docker Compose і як він допомагає у міграції багатокомпонентних веб-застосунків?
5. Як забезпечити збереження даних веб-застосунку при роботі з контейнерами Docker?
6. Як здійснюється налаштування мережі між контейнерами у випадку веб-застосунків із базою даних?
7. Наведіть підходи для масштабування веб-застосунку у контейнерах Docker.
8. Вкажіть як відстежувати та логувати роботу веб-застосунку в Docker-контейнерах.

Лабораторна робота № 9

Тема. Запуск контейнерів у керованому сервісі.

Мета : ознайомитися з принципами роботи керованих сервісів контейнеризації (Managed Container Services), навчитися розгортати та керувати контейнерами у хмарному середовищі, забезпечувати масштабування та моніторинг контейнеризованих застосунків.

Література: [5, 7, 19, 22]

Теоретичні відомості

Сучасна розробка програмного забезпечення все частіше використовує контейнеризацію як ключову технологію для забезпечення портативності, масштабованості та ефективного управління додатками. Контейнери дозволяють ізолювати програми та їхні залежності, що забезпечує стабільну роботу незалежно від середовища розгортання. Однак, у процесі масштабування та управління великою кількістю контейнерів виникає потреба в автоматизації та оптимізації цих процесів. Саме тут на допомогу приходять керовані сервіси для контейнерів, які дозволяють значно спростити запуск, моніторинг та масштабування контейнеризованих додатків.

Керовані сервіси контейнеризації, такі як Amazon Elastic Kubernetes Service (EKS), Google Kubernetes Engine (GKE) або Azure Kubernetes Service (AKS), надають розробникам можливість запускати контейнери без необхідності самостійно налаштовувати та підтримувати інфраструктуру. Вони автоматизують створення кластерів, балансування навантаження, оновлення та масштабування контейнерів, що дозволяє командам зосередитися на розробці функціоналу, а не на операційних завданнях. Важливо, що такі сервіси забезпечують інтеграцію з іншими хмарними компонентами, такими як бази даних, системи зберігання та сервіси безпеки, що робить розгортання комплексних додатків більш зручним та надійним.

Процес запуску контейнерів у керованому сервісі зазвичай починається зі створення образу контейнера та його публікації у реєстрі контейнерів, наприклад Docker Hub або приватному хмарному реєстрі. Далі, розробник описує конфігурацію розгортання, включно з кількістю реплік, обмеженнями ресурсів та політиками масштабування. Керований сервіс автоматично обробляє ці параметри, забезпечуючи запуск контейнерів на доступних вузлах кластера та відновлюючи їх у разі збою. Це значно підвищує надійність та стійкість додатків.

Однією з ключових переваг керованих сервісів є їх здатність до автоматичного масштабування. Наприклад, у періоди високого навантаження сервіс може динамічно додавати нові екземпляри контейнерів, а у періоди низького навантаження – зменшувати їх кількість, тим самим оптимізуючи витрати на ресурси. Крім того, керовані сервіси надають вбудовані механізми моніторингу та логування, що дозволяє відстежувати стан контейнерів, виявляти проблеми та швидко реагувати на інциденти.

Безпека також є важливим аспектом. Керовані сервіси забезпечують автоматичне оновлення вузлів кластера, шифрування даних та контроль

доступу на основі ролей. Це дозволяє мінімізувати ризики експлуатаційних помилок та кібератак, що особливо важливо для корпоративних середовищ.

Підсумовуючи, запуск контейнерів у керованому сервісі надає розробникам потужний інструмент для ефективного управління додатками, скорочення часу на операційні завдання та підвищення надійності та масштабованості сервісів. Використання таких платформ стає все більш популярним у сучасних DevOps-практиках, оскільки вони забезпечують баланс між контролем та автоматизацією, дозволяючи командам зосередитися на створенні цінності для кінцевого користувача.

Завдання для виконання:

1. Налаштування підмереж для використання Amazon RDS та Elastic Beanstalk (рис. 9).
2. Налаштування бази даних Aurora Serverless.
3. Перегляд образу контейнера.
4. Налаштування зв’язку між контейнером та базою даних.
5. Створення об’єктів бази даних програми.
6. Заповнення бази даних даними постачальника.
7. Перегляд політики IAM та ролі для Elastic Beanstalk.
8. Створення програми Elastic Beanstalk.
9. Налаштування проксі-сервера Amazon API Gateway.

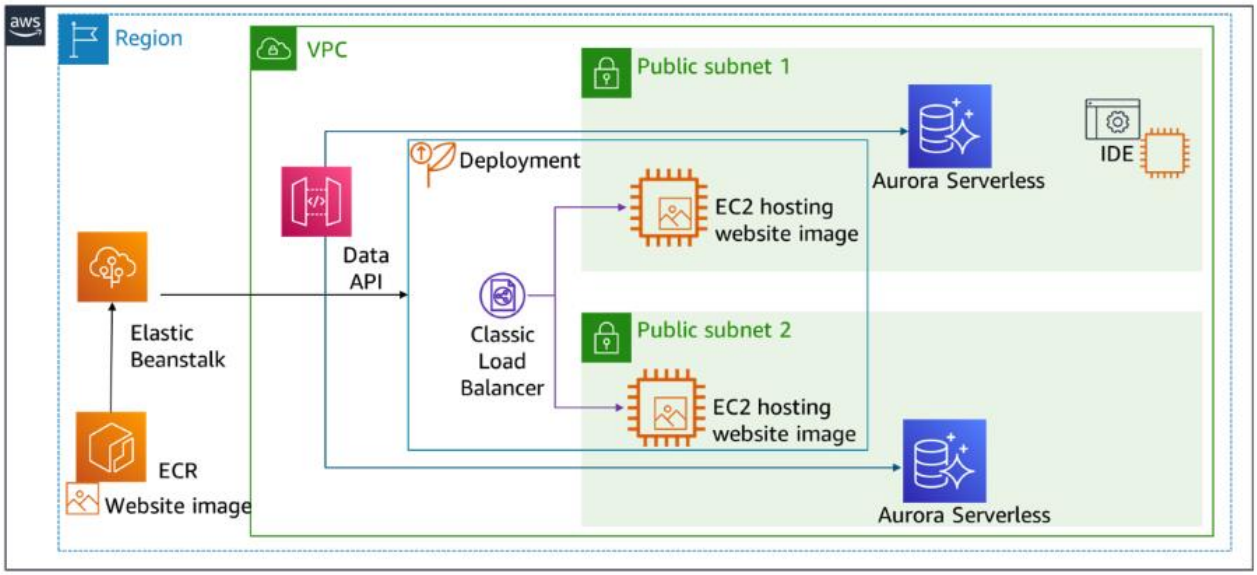

Рисунок 9 – Механізм взаємодії Amazon RDS та Elastic Beanstalk

Контрольні питання

1. Назвіть популярні хмарні керовані сервіси для запуску контейнерів.
2. Які переваги використання керованого сервісу для запуску контейнерів?
3. Що таке кластер у контексті керованого контейнерного сервісу?
4. Поясніть різницю між завданням (task) та сервісом (service) у керованих контейнерних платформах.
5. Як забезпечується масштабування контейнерів у керованому сервісі?
6. Що таке container definitions і як вони впливають на запуск?
7. Як організовується мережеве підключення контейнерів у керованих сервісах?

Лабораторна робота № 10

Тема. Кешування даних застосунку за допомогою ElastiCache.

Мета : ознайомитися з принципами кешування даних у сучасних веб-застосунках та навчитися впроваджувати механізм кешування за допомогою сервісу Amazon ElastiCache.

Література: [3, 8, 14, 20]

Теоретичні відомості

У сучасних веб- та мобільних застосунках швидкість доступу до даних є критичною для забезпечення комфортного користувацького досвіду та масштабованості системи. Класичний підхід до зберігання даних у реляційних або NoSQL базах даних не завжди забезпечує необхідну швидкість відповіді, особливо при високих навантаженнях. Саме тому кешування даних стало невід'ємною частиною архітектури багатьох сучасних застосунків. Одним із потужних інструментів для цього є сервіс Amazon ElastiCache, який надає керовані розподілені кеші на основі Redis або Memcached.

ElastiCache дозволяє зберігати часто використовувані дані у швидкій пам'яті, значно скорочуючи час доступу у порівнянні із запитами до бази даних. Наприклад, інформація про сеанси користувачів, результати складних запитів або статистичні дані можуть кешуватися у Redis, що дозволяє застосунку обробляти тисячі запитів на секунду без перевантаження бази даних. Крім того, кешування знижує латентність, забезпечує стабільну продуктивність та дозволяє ефективно масштабувати систему у хмарному середовищі.

Однією з ключових переваг ElastiCache є його інтеграція з екосистемою AWS. Користувачі можуть швидко створювати та масштабувати кеші, налаштовувати реплікацію для забезпечення високої доступності, а також автоматично відновлювати дані після збою. Сервіс підтримує горизонтальне масштабування та шардінг, що дозволяє обробляти великі об'єми даних без зниження продуктивності. Водночас Redis у ElastiCache надає додаткові можливості, такі як підтримка структур даних (списки, множини, геші), TTL (час життя ключів) та транзакцій, що робить кешування гнучким і адаптованим до різних сценаріїв.

Ефективне кешування вимагає правильного вибору стратегії. Найпопулярнішими є write-through, write-back та lazy-loading. У першому випадку дані записуються одночасно у базу та кеш, забезпечуючи синхронність. Write-back дозволяє зменшити навантаження на базу, записуючи дані у кеш з подальшим асинхронним оновленням бази. Lazy-loading передбачає завантаження даних у кеш лише при запиті, що оптимально для великих наборів рідко змінюваних даних.

Проте кешування має й свої ризики. Дані у кеші можуть застарівати, тому необхідно враховувати механізми інвалідизації та TTL. Неправильне налаштування шардінгу чи відсутність моніторингу може призвести до перевантаження кешу або втрати продуктивності. В цьому контексті ElastiCache надає інструменти моніторингу та метрики через CloudWatch, що дозволяє вчасно реагувати на проблеми та оптимізувати використання ресурсів.

Отже, використання ElastiCache для кешування даних у сучасних застосунках забезпечує швидкий доступ до інформації, знижує навантаження на базу даних та сприяє масштабованості системи. Завдяки інтеграції з AWS та підтримці Redis і Memcached, розробники отримують потужний інструмент для побудови високопродуктивних, надійних та гнучких рішень, що відповідають вимогам сучасного цифрового світу.

Завдання для виконання:

1. Підготовка середовища розробки у відповідності до рисунку 10.
2. Налаштування підмереж для використання ElastiCache
3. Створення кластера ElastiCache
4. Завантаження записів у кеш з бази даних
5. Оновлення даних
6. Створення нового запису
7. Видалення запису
8. Оновлення контейнера програми

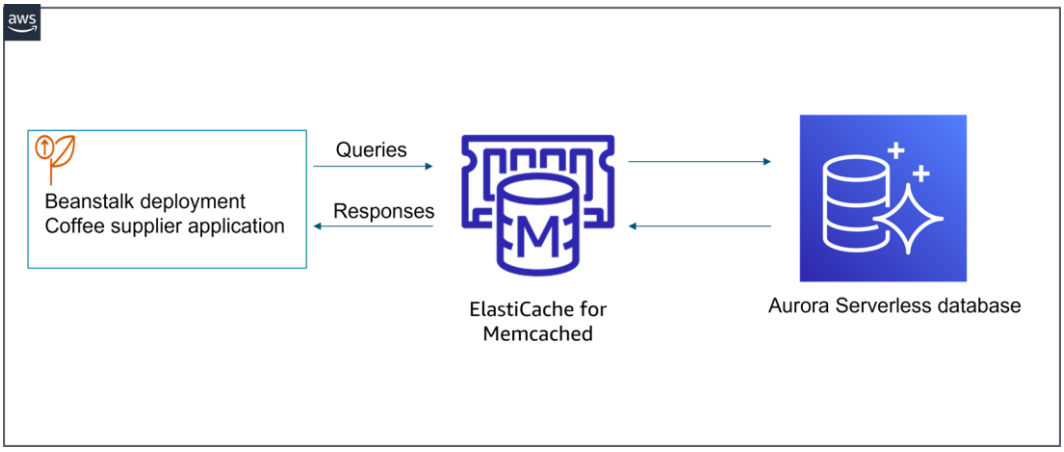

Рисунок 10 – Механізм кешування Amazon ElastiCache

Контрольні питання

1. Що таке Amazon ElastiCache і для чого він використовується у застосунках?
2. Які два основні типи ElastiCache підтримуються і в чому їхня різниця?
3. Як кешування за допомогою ElastiCache може вплинути на продуктивність застосунку?
4. Що таке TTL (Time-to-Live) у контексті кешування і як він налаштовується в ElastiCache?
5. Які сценарії використання Redis у ElastiCache є найпоширенішими?
6. Як забезпечити високу доступність даних при використанні ElastiCache?
7. Що таке шардінг (sharding) у ElastiCache і коли його доцільно застосовувати?
8. Які ризики або обмеження пов’язані з кешуванням даних у ElastiCache?
9. Як інтегрувати ElastiCache з існуючим застосунком на прикладі Node.js або Python?
10. Вкажіть відмінності між кешуванням даних на стороні клієнта від кешування за допомогою ElastiCache

Лабораторна робота № 11

Тема. Впровадження CloudFront для кешування та безпеки застосунку.

Мета : ознайомитися з можливостями Amazon CloudFront для підвищення продуктивності та безпеки веб-застосунків шляхом впровадження CDN, реалізації кешування контенту, налаштування правил доступу та захисту від атак.

Література: [1, 4, 13, 21]

Теоретичні відомості

Сучасні веб-застосунки вимагають високої продуктивності та надійного захисту від кіберзагроз, особливо в умовах глобальної аудиторії. Одним із ефективних рішень для досягнення цих цілей є використання Amazon CloudFront – глобальної мережі доставки контенту (CDN), яка дозволяє оптимізувати швидкість завантаження ресурсів і підвищити безпеку веб-застосунку.

Основною перевагою впровадження CloudFront є його здатність до кешування контенту на мережі вузлів (Edge Locations), розташованих по всьому світу. Завдяки цьому користувачі отримують дані з найближчого географічного вузла, що значно знижує затримки та час завантаження сторінок. Наприклад, статичні ресурси, такі як зображення, скрипти та стилі, можуть зберігатися на серверах CloudFront, дозволяючи браузеру завантажувати їх безпосередньо з CDN, а не з основного сервера. Це не лише підвищує продуктивність, але й зменшує навантаження на сервер застосунку, що критично для високонавантажених систем.

Крім кешування, CloudFront надає потужні інструменти для забезпечення безпеки веб-застосунку. Використання інтегрованого AWS Web Application Firewall (WAF) дозволяє створювати правила для блокування шкідливих запитів, таких як SQL-ін’єкції, XSS-атаки або інші типові загрози веб-застосункам. Крім того, CloudFront підтримує шифрування трафіку через HTTPS та інтегрується з AWS Shield для захисту від DDoS-атак. Це дозволяє значно підвищити стійкість системи до зовнішніх атак, зберігаючи при цьому швидкодію і доступність для легітимних користувачів.

Впровадження CloudFront також забезпечує гнучкість у налаштуванні кешування. Можна визначати різні політики кешування для статичного та динамічного контенту, встановлювати час життя кешу (TTL) та використовувати версіонування ресурсів для автоматичного оновлення контенту у вузлах CDN. Такі можливості дозволяють оптимально балансувати між швидкістю завантаження та актуальністю даних, що особливо важливо для застосунків із частими оновленнями.

В результаті, інтеграція CloudFront у архітектуру веб-застосунку приносить одночасно дві ключові вигоди: прискорення доставки контенту та підвищення рівня безпеки. Вона зменшує час відгуку для користувачів по всьому світу, знижує навантаження на основні сервери та захищає систему від типових кіберзагроз. Використання CloudFront стає особливо актуальним для компаній із глобальною аудиторією та високими вимогами до доступності та безпеки даних.

Отже, впровадження Amazon CloudFront є стратегічно важливим кроком для сучасних веб-застосунків, що прагнуть поєднати продуктивність і надійний захист, забезпечуючи користувачам швидкий та безпечний доступ до сервісів незалежно від їхнього місцезнаходження.

Завдання для виконання:

1. Підготовка середовища розробки у відповідності до рисунку 11.
2. Налаштування дистрибутива Amazon CloudFront для статичного контенту веб-сайту.
3. Захист мережевого доступу до дистрибутива CloudFront за допомогою AWS WAF.
4. Захист кінцевої точки REST API за допомогою AWS WAF.
5. Налаштування функції CloudFront на веб-сайті.
6. Налаштування тривалості кешу.

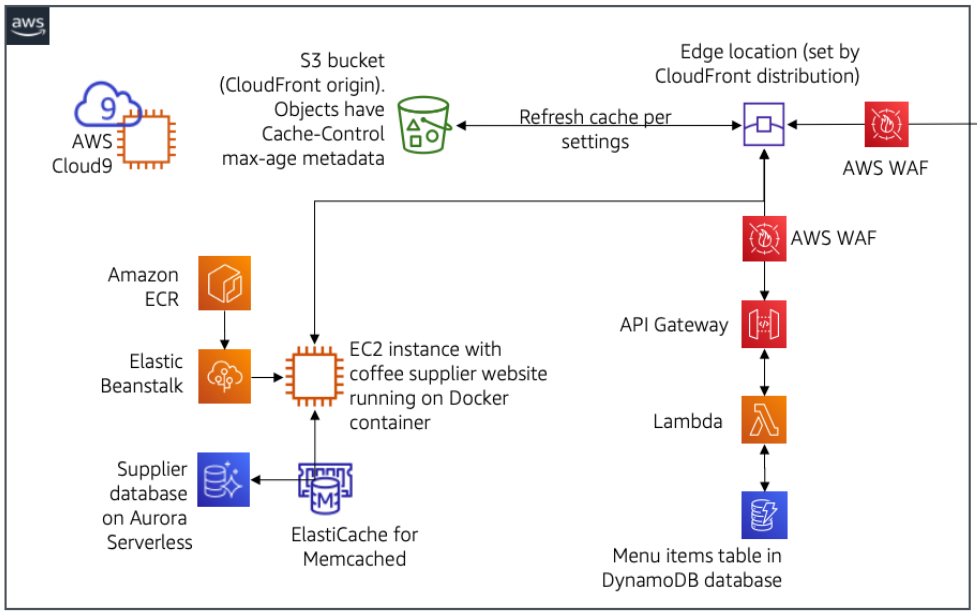

Рисунок 11 – Діаграма розгорнутого веб-сайту

Контрольні питання

1. Що таке Amazon CloudFront і які його основні функції?
2. Як CloudFront впливає на продуктивність застосунку через кешування контенту?
3. Які типи кешування підтримує CloudFront і як вони працюють?
4. Як налаштувати TTL (Time-to-Live) для об’єктів у CloudFront?
5. Які механізми безпеки надає CloudFront для захисту від DDoS-атак?
6. Що таке AWS WAF і як його інтегрувати з CloudFront для підвищення безпеки?
7. Як CloudFront допомагає забезпечити шифрування між користувачем і сервером?
8. Що таке origin і origin access identity (OAI) у CloudFront?
9. Як відбувається інвалідація кешу в CloudFront і в яких випадках це необхідно робити?
10. Які метрики та логи CloudFront слід моніторити для оцінки ефективності кешування та безпеки?

Лабораторна робота № 12

Тема. Організація системи повідомлень на основі Amazon SNS і SQS.

Мета : навчитися організовувати публікацію, підписку та обробку повідомлень у хмарній інфраструктурі AWS для забезпечення надійного та масштабованого обміну даними між компонентами розподілених застосунків.

Література: [2, 3, 15, 22]

Теоретичні відомості

У сучасних інформаційних системах, де обробка великих обсягів даних відбувається в режимі реального часу, ефективна організація системи повідомлень стає критичною. Одним із найпопулярніших рішень для побудови таких систем у хмарі є комбінація сервісів Amazon Simple Notification Service (SNS) і Amazon Simple Queue Service (SQS). Ці сервіси забезпечують надійний, масштабований та гнучкий обмін повідомленнями між різними компонентами системи, дозволяючи реалізовувати архітектури на основі подій.

Amazon SNS є сервісом публікації-підписки (publish-subscribe), який дозволяє надсилати повідомлення одночасно багатьом підписникам. Підписниками можуть бути як інші сервіси AWS, так і зовнішні системи: HTTP/S endpoint, email, SMS або навіть мобільні push-повідомлення. Основна перевага SNS полягає в здатності розсилати повідомлення миттєво та паралельно на багато кінцевих точок, що робить його ідеальним для оповіщень, тригерів подій і інтеграцій між мікросервісами.

Amazon SQS, навпаки, є сервісом чергування повідомлень, що забезпечує надійну асинхронну доставку. SQS дозволяє системам передавати повідомлення у чергу, де вони зберігаються до моменту, поки одержувач їх не обробить. Це гарантує, що жодне повідомлення не буде втрачено, навіть у разі тимчасової недоступності обробника. SQS підтримує як стандартні черги з високою пропускною здатністю, так і FIFO-черги, які зберігають порядок повідомлень і гарантують доставку без дублювання.

Поєднання SNS і SQS дозволяє створити архітектуру, яка об’єднує переваги обох сервісів. Наприклад, у сценарії обробки замовлень електронної комерції, коли нове замовлення створюється, SNS може відразу розіслати повідомлення про нього до кількох підсистем: аналітики, складу, служби доставки та системи повідомлень користувачам. Кожна з цих систем може мати власну SQS-чергу, яка забезпечує асинхронне та надійне отримання повідомлень. Це дозволяє компонентам працювати незалежно та масштабуватися відповідно до навантаження.

Крім того, така інтеграція спрощує управління помилками та повторну обробку повідомлень. Якщо обробник тимчасово недоступний, повідомлення залишаються у черзі SQS і будуть оброблені пізніше, що підвищує надійність системи. Додатково, використання SNS разом із SQS дозволяє організувати багаторівневу маршрутизацію повідомлень і легко додавати нові підписники без змін у коді відправника.

Таким чином, організація системи повідомлень на основі Amazon SNS і SQS забезпечує гнучкість, масштабованість та надійність обміну даними між компонентами сучасних інформаційних систем. Такий підхід дозволяє

реалізувати архітектуру на основі подій, що особливо важливо для високонавантажених сервісів, де час реакції і збереження даних критично важливі.

Завдання для виконання:

1. Підготовка середовища розробки у відповідності до рисунку 12.
2. Налаштування черги недоставлених листів Amazon SQS.
3. Налаштування черги Amazon SQS.
4. Тема налаштування Amazon SNS.
5. Зв’язування Amazon SQS та Amazon SNS.
6. Тестування публікації повідомлень.
7. Налаштування програми для опитування черги.

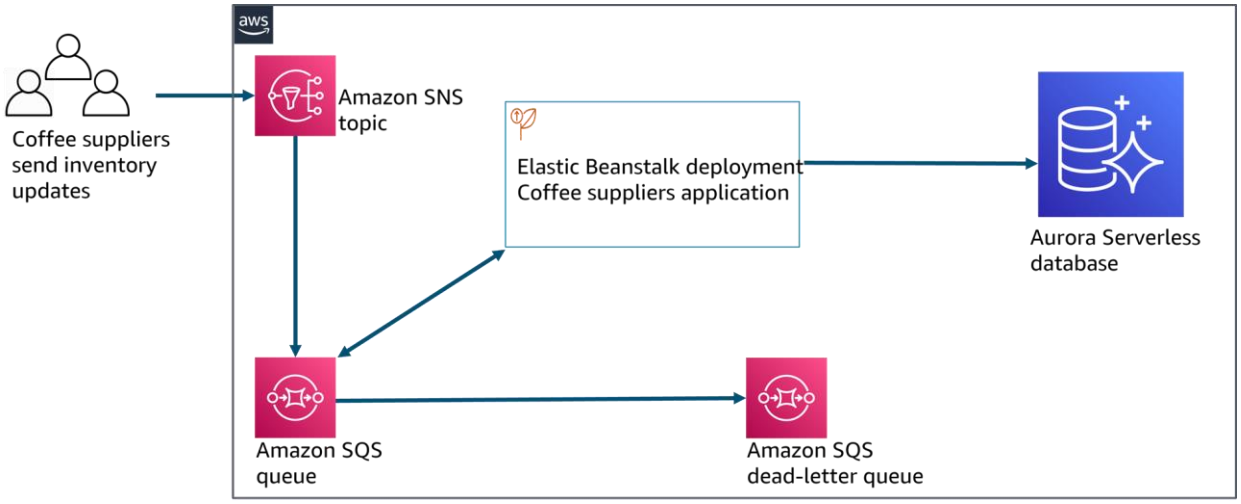

Рисунок 12 – Архітектура оновлення інвентаризації

Контрольні питання

1. Що таке Amazon SNS та які основні функції цієї служби?
2. Які типи підписників можна додавати до SNS-теми?
3. Що таке Amazon SQS і які основні сценарії його використання?
4. Поясніть різницю між стандартною чергою (Standard Queue) та чергою FIFO (FIFO Queue) в Amazon SQS.
5. Як налаштувати інтеграцію SNS і SQS для забезпечення надійної доставки повідомлень?
6. Що таке «dead-letter queue» в SQS і як вона використовується?
7. Які механізми контролю повторної доставки повідомлень у SQS доступні розробникам?
8. Яким чином можна забезпечити масштабування системи повідомлень на основі SNS і SQS?
9. Опишіть процес обробки повідомлень від моменту публікації в SNS до отримання повідомлення в SQS і його обробки споживачем.

Лабораторна робота № 13

Тема. Оркестрація безсерверних функцій за допомогою покрокових функцій.
Мета : Ознайомитися з принципами оркестрації безсерверних (serverless) функцій за допомогою покрокових функцій (step functions), навчитися створювати та керувати послідовністю виконання функцій, забезпечувати взаємодію між ними.

Література: [3, 6, 12, 20]

Теоретичні відомості

У сучасному світі програмування безсерверні технології набувають все більшої популярності завдяки своїй масштабованості, економічності та гнучкості. Вони дозволяють розробникам створювати додатки без необхідності управління інфраструктурою, фокусуючись виключно на бізнес-логіці. Однак із збільшенням складності безсерверних додатків виникає потреба в ефективній оркестрації функцій — способі координувати взаємодію кількох функцій для досягнення цілісного результату. Саме тут на допомогу приходять покрокові функції (step functions).

Покрокові функції є потужним інструментом для організації безсерверних процесів. Вони дозволяють описати робочий процес як послідовність кроків, кожен з яких може виконуватися окремо, незалежно від інших. Кожен крок у такій функції представляє собою окрему безсерверну функцію, яка може повертати результат, обробляти помилки або викликати інші функції. Завдяки цьому розробники отримують можливість детально контролювати логіку додатка, а також гнучко реагувати на непередбачувані ситуації.

Однією з ключових переваг оркестрації через покрокові функції є підтримка умовної логіки та повторних спроб виконання. Наприклад, якщо певний крок викликає помилку через тимчасову проблему з мережею чи базою даних, функція може автоматично повторити спробу або виконати альтернативний сценарій. Такий підхід значно підвищує надійність безсерверних додатків, зменшуючи ризик збоїв у критичних процесах. Крім того, покрокові функції дозволяють легко інтегрувати різні сервіси та API, створюючи єдину робочу ланку для складних бізнес-процесів.

Іншим важливим аспектом є прозорість і відстежуваність процесів. Використання покрокових функцій забезпечує візуальне уявлення робочого процесу та запис логів для кожного кроку. Це спрощує діагностику помилок і оптимізацію додатка. Розробники можуть швидко зрозуміти, на якому етапі виникла проблема, та внести необхідні корективи без зупинки всього процесу.

Завдяки покроковим функціям, оркестрація безсерверних функцій стає не лише ефективною, але й гнучкою. Вони дозволяють створювати модульні та масштабовані рішення, які легко адаптувати під нові вимоги бізнесу. У поєднанні з іншими сервісами хмарної інфраструктури, такими як бази даних, черги повідомлень чи сховища файлів, покрокові функції стають фундаментом для побудови сучасних, стійких до навантажень і легко керованих додатків.

Отже, покрокові функції відкривають нові горизонти для безсерверного програмування. Вони забезпечують ефективну оркестрацію функцій,

дозволяючи будувати складні робочі процеси, контролювати помилки та інтегрувати різноманітні сервіси. У світі, де швидкість розробки та надійність додатків мають вирішальне значення, такі підходи стають невід'ємною частиною сучасної архітектури програмного забезпечення.

Завдання для виконання:

1. Підготовка середовища розробки у відповідності до рисунку 13.
2. Налаштування теми SNS та підписка на неї.
3. Створення кінцевого автомата Step Functions.
4. Створення лямбда-функції для генерації попередньо підписаної URL-адреси.
5. Налаштування REST API для виклику кінцевого автомата.
6. Налаштування лямбда-функції для генерації HTML-звіту.
7. Додавання функції GenerateHTML до кінцевого автомата.
8. Створення лямбда-функції для отримання записів постачальників.
9. Додавання функції generateReportData до кінцевого автомата.

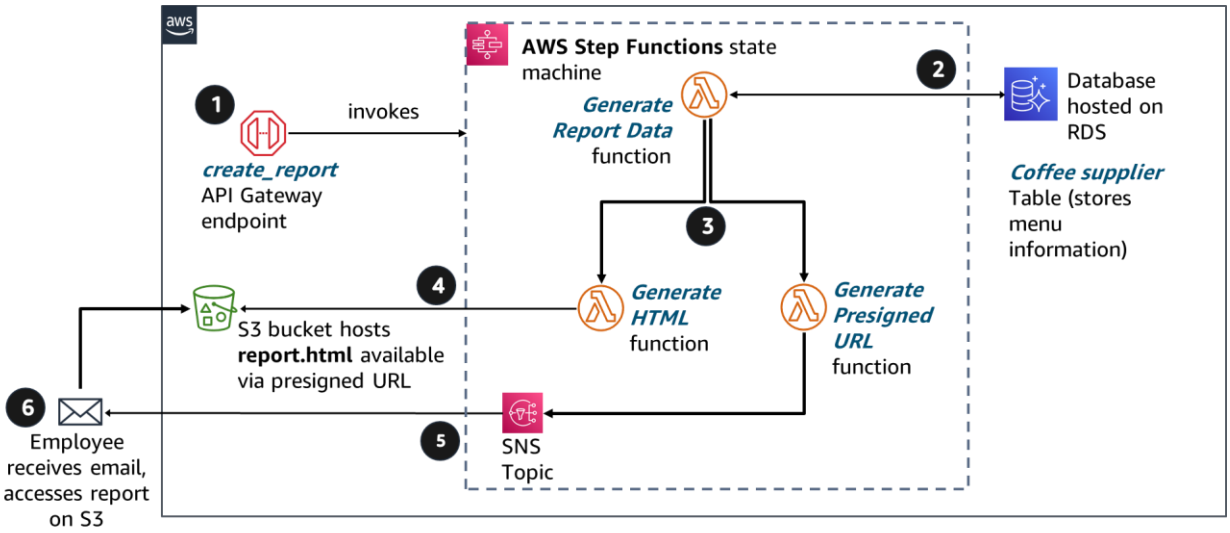

Рисунок 13 – Механізм оркестрації безсерверних функцій

Контрольні питання

1. Що таке покрокові функції (Step Functions) у контексті безсерверних обчислень?
2. Які основні компоненти визначають роботу покрокової функції?
3. Чим відрізняється оркестрація безсерверних функцій від простої послідовної обробки функцій?
4. Які типи станів існують у Step Functions і яка їхня роль?
5. Поясніть, як реалізується умовна логіка (if/else) у покрокових функціях.
6. Як покрокові функції інтегруються з іншими безсерверними сервісами, наприклад, AWS Lambda або S3?
7. В яких випадках доцільно використовувати паралельні потоки (Parallel state) у покрокових функціях?
8. Які обмеження існують щодо тривалості виконання покрокових функцій?
9. Наведіть приклад практичного сценарію використання покрокових функцій для автоматизації бізнес-процесу.

Лабораторна робота № 14

Тема. Впровадження автентифікації застосунку за допомогою Amazon Cognito.
Мета : вивчити принципи роботи Amazon Cognito та набуття навичок інтеграції автентифікації користувачів у застосунок для забезпечення безпечного доступу до ресурсів.

Література: [1, 4, 10, 21]

Теоретичні відомості

У сучасному світі безпека цифрових продуктів набуває особливого значення. Користувачі очікують не лише зручності у використанні застосунків, а й захисту своїх даних. Одним із ефективних рішень для реалізації автентифікації та управління користувачами є сервіс Amazon Cognito, що належить до екосистеми AWS. Ця платформа дозволяє розробникам швидко інтегрувати аутентифікацію, управління обліковими записами та безпечний доступ до ресурсів, мінімізуючи витрати часу на реалізацію власної системи безпеки.

Amazon Cognito забезпечує два основні компоненти: User Pools і Identity Pools. User Pools відповідають за управління обліковими записами користувачів: реєстрацію, автентифікацію, відновлення пароля та багатофакторну автентифікацію (MFA). Identity Pools, у свою чергу, дозволяють надавати користувачам тимчасові облікові дані AWS для доступу до інших сервісів, таких як S3 чи DynamoDB. Така архітектура дозволяє легко розмежовувати логіку автентифікації та авторизації, підвищуючи безпеку застосунку.

Процес впровадження автентифікації з Amazon Cognito починається зі створення User Pool, де визначаються політики безпеки, атрибути користувача та методи входу (електронна пошта, телефон, соціальні мережі). Далі розробник інтегрує Cognito SDK або використовує бібліотеки, такі як Amplify, для взаємодії застосунку з сервісом. Це дозволяє реалізувати реєстрацію користувачів, вхід у систему, підтвердження електронної пошти та відновлення пароля без необхідності створювати власний бекенд для обробки цих функцій. Крім того, Amazon Cognito підтримує інтеграцію з популярними провайдерами ідентифікації, такими як Google, Facebook або Apple, що забезпечує гнучкість і зручність для користувачів.

Однією з ключових переваг використання Cognito є безпечне управління токенами доступу. Після автентифікації користувач отримує JWT-токени (ID token, Access token та Refresh token), які можна використовувати для підтвердження особи при зверненні до REST API або GraphQL. Токени автоматично керуються сервісом, включно з оновленням і перевіркою строку дії, що зменшує ризик витоку даних та спрощує підтримку безпечного доступу.

Впровадження Amazon Cognito також сприяє масштабованості застосунку. Сервіс здатний обробляти мільйони користувачів без додаткового налаштування серверної інфраструктури, що особливо важливо для стартапів та великих проектів. Крім того, інтеграція з іншими сервісами AWS, такими як Lambda чи API Gateway, дозволяє реалізувати складні сценарії авторизації та управління доступом.

Отже, використання Amazon Cognito для автентифікації застосунків є ефективним, безпечним та масштабованим рішенням. Сервіс надає розробникам готовий інструментарій для управління користувачами та токенами, інтеграцію з провайдерами ідентифікації і підтримку сучасних практик безпеки. Це дозволяє сфокусуватися на розвитку функціоналу застосунку, одночасно забезпечуючи високий рівень захисту даних користувачів. Використання Cognito значно спрощує процес автентифікації і стає оптимальним вибором для сучасних веб- та мобільних застосунків.

Завдання для виконання:

1. Підготовка середовища розробки у відповідності до рисунку 14.
2. Налаштування пулу користувачів Cognito та клієнта програми.
3. Налаштування клієнтської програми.
4. Інтеграція розміщеного URI Amazon Cognito у веб-сайт.
5. Спостереження за деталями кінцевої точки REST API та тестування.
6. Створення користувача для пулу користувачів Amazon Cognito.
7. Налаштування авторизатора API Gateway.
8. Тестування процесу запиту з веб-сайту.

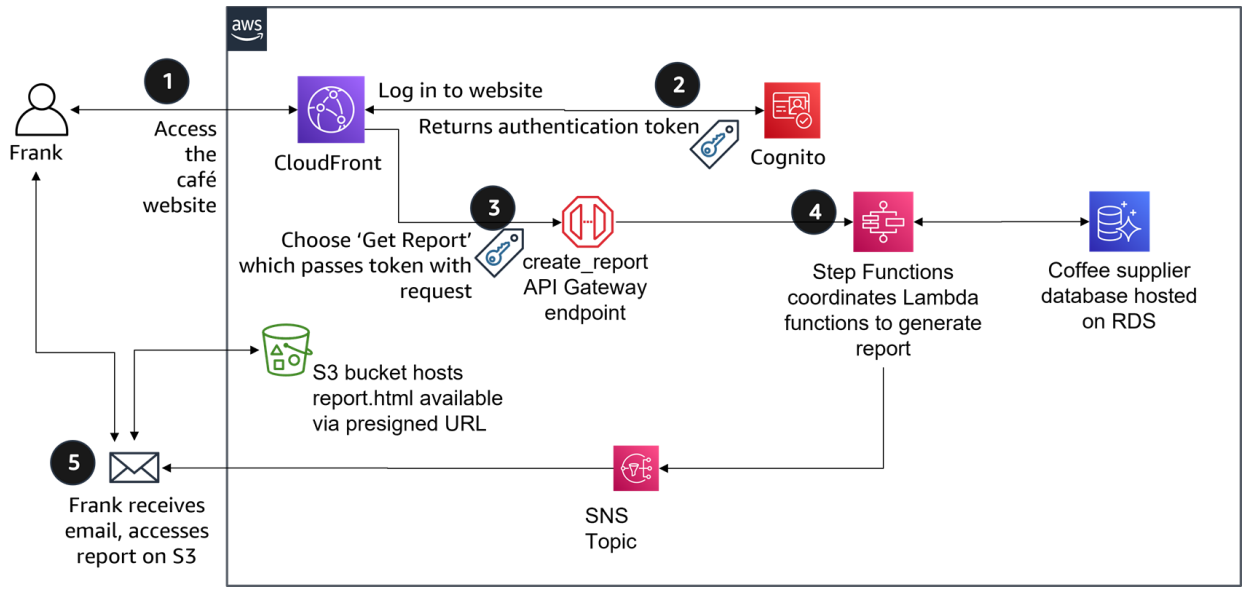

Рисунок 14 – Схема автентифікації застосунку за допомогою Amazon Cognito

Контрольні питання

1. Що таке Amazon Cognito і які основні проблеми автентифікації він вирішує?
2. Як налаштувати User Pool у Amazon Cognito для реєстрації та входу користувачів?
3. Які методи автентифікації підтримує Cognito (пароль, соціальні провайдери, SAML тощо)?
4. Що таке токени в Amazon Cognito (ID token, access token, refresh token) і для чого вони використовуються?
5. Як здійснюється інтеграція Cognito із застосунком (SDK, Hosted UI)?
6. Що таке Multi-Factor Authentication (MFA) в Cognito і як її налаштувати?
7. Як реалізувати обмеження доступу до ресурсів AWS на основі ролей, пов'язаних із Cognito (Cognito Identity Pool + IAM roles)?

Лабораторна робота № 15

Тема. Автоматизація розгортання застосунку за допомогою конвеєра CI/CD.

Мета : вивчити та застосувати методи автоматизації розгортання застосунку за допомогою конвеєра CI/CD для підвищення швидкості та надійності релізів.

Література: [6, 9, 18, 21]

Теоретичні відомості

Автоматизація розгортання застосунків стала ключовим аспектом сучасної розробки програмного забезпечення, особливо у світі DevOps, де швидкість і надійність випуску нових версій визначають конкурентоспроможність компанії. Одним із найефективніших підходів до цього процесу є використання конвеєра CI/CD – системи безперервної інтеграції та безперервного розгортання. CI/CD дозволяє командам автоматизувати не лише збірку та тестування коду, а й його розгортання в різних середовищах, що значно підвищує якість та швидкість доставки продукту.

Процес CI (Continuous Integration) передбачає регулярне інтегрування змін у спільний репозиторій коду. Кожна зміна автоматично перевіряється за допомогою тестів та інших механізмів контролю якості. Це дозволяє виявляти помилки на ранніх етапах, зменшуючи ризик виникнення серйозних проблем у продакшн-середовищі. Автоматизація цього процесу забезпечується інструментами, такими як Jenkins, GitLab CI/CD, GitHub Actions або CircleCI, які можуть запускати збірку, тестування та статичний аналіз коду після кожного коміту.

CD (Continuous Delivery або Continuous Deployment) відповідає за безперервну доставку або розгортання застосунку. У випадку Continuous Delivery зміни коду після успішного проходження всіх тестів готуються для релізу, але розгортання на продуктивне середовище здійснюється вручну або після додаткового погодження. Continuous Deployment, у свою чергу, автоматизує весь процес, включаючи доставку змін на продакшн, що дозволяє випускати нові функції і виправлення максимально швидко та без людського втручання. Цей підхід забезпечує швидкий зворотний зв’язок і дозволяє бізнесу адаптуватися до змін ринку у реальному часі.

Одним із ключових елементів CI/CD є використання контейнеризації та оркестрації, наприклад Docker і Kubernetes. Контейнери забезпечують однакове середовище для розробки, тестування та продакшн, що мінімізує проблеми сумісності та конфігурацій. Оркестратори автоматично масштабують застосунки, забезпечують балансування навантаження та підтримують високу доступність сервісів. Завдяки цьому розгортання стає передбачуваним і стабільним, а ризики помилок у продакшн зводяться до мінімуму.

Крім технічних переваг, автоматизація CI/CD має великий вплив на командну роботу та культуру розробки. Вона стимулює практику частих комітів, покращує прозорість процесів і дозволяє розробникам зосередитися на створенні цінності для користувача, а не на рутинних завданнях розгортання. Команди отримують можливість швидко реагувати на зворотний зв’язок, впроваджувати нові функції та оперативно виправляти помилки.

Отже, автоматизація розгортання застосунку за допомогою конвеєра CI/CD є критично важливим інструментом для сучасної розробки програмного забезпечення. Вона забезпечує надійність, швидкість і гнучкість процесів, дозволяє ефективно управляти змінами та підтримувати високу якість продукту. Впровадження CI/CD є не лише технічним рішенням, але й важливим кроком у формуванні культури безперервного вдосконалення в команді.

Завдання для виконання:

1. Підготовка середовища розробки у відповідності до рисунку 15.
2. Створення репозиторію коду.
3. Створення конвеєра для автоматизації оновлень веб-сайту.
4. Клонування репозиторію.
5. Вивчення інтеграції Git з IDE.
6. Завантаження коду веб-сайту кафе до репозиторію коду.

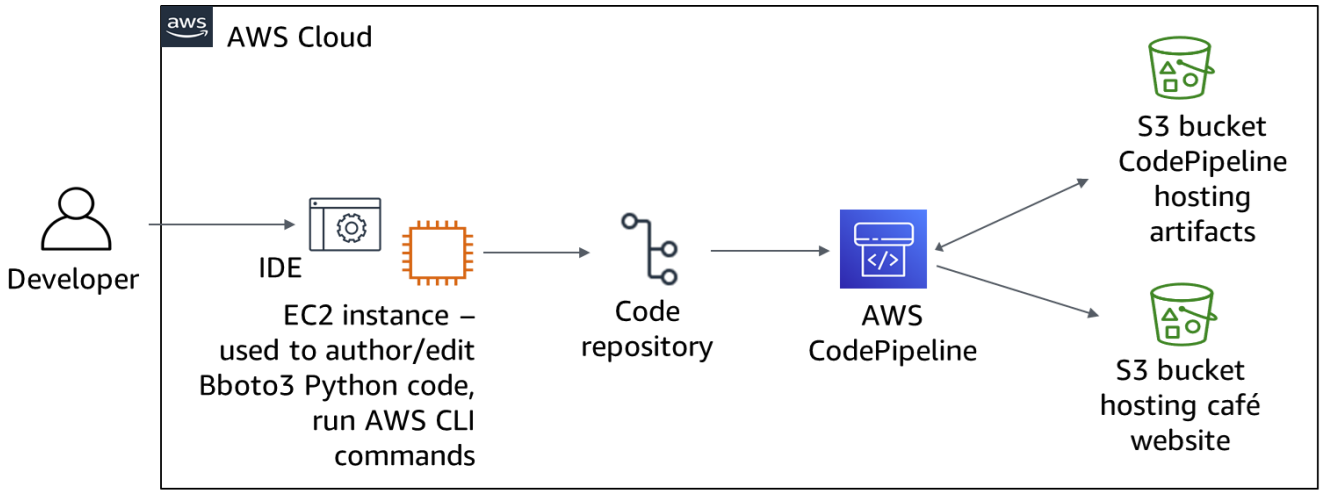

Рисунок 15 – Архітектура розгортання застосунку з конвеєра CI/CD

Контрольні питання

1. Що таке конвеєр CI/CD і які основні етапи він включає?
2. Яка різниця між Continuous Integration (CI) та Continuous Deployment/Delivery (CD)?
3. Назвіть популярні інструменти для організації CI/CD-процесів.
4. Яку роль відіграє автоматичне тестування у конвеєрі CI/CD?
5. Як можна автоматизувати збірку та деплой застосунку за допомогою скриптів або плагінів CI/CD?
6. Що таке pipeline as code і які його переваги?
7. Як здійснюється інтеграція систем контролю версій (Git, GitLab, GitHub) у CI/CD-процес?
8. Які кроки можна включити у конвеєр для забезпечення безпеки розгортання застосунку?
9. Як моніторинг і логування допомагають у процесі CI/CD?
10. Наведіть приклад ситуації, коли автоматизація розгортання через CI/CD економить час і знижує ризики помилок.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Andrushchak I., Koshelyuk V., Kominko V., Shepeliuk S., Levchuk M. SEO analysis of modern and current software products. The 9th International scientific and practical conference «Scientists and existing problems of human development». Zagreb, Croatia. International Science Group. 2023. pp. 348-352.
2. Dennis, A., Wixom, B.H., Tegarden, D. Systems Analysis and Design: An Object-Oriented Approach with UML. 8th Edition: Wiley, 2024. 544 p.
3. INCOSE. Systems Engineering Handbook. 5th Edition: Wiley, 2023. 211 p.
4. Issa, T., et al. Management Information Systems: Managing the Digital World: Springer, 2024. 57 p.
5. Kendall, K.E., Kendall, J.E. Systems Analysis and Design. 11th Edition: Pearson, 2023. 609 p.
6. Kleppmann, M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. 2nd Edition: O'Reilly Media, 2024. 590 p.
7. Martin, R.C. Clean Architecture: A Craftsman's Guide to Software Structure and Design (Updated Edition). Prentice Hall, 2023. 420 p.
8. Martsenyuk V.P., Sverstyuk A.S., Andrushchak I.Ye., Kosheliuk V.A., Matviiv Yu.Ya. Information modeling of the calculation of composite bodies with cracks by the joint action of mechanical and thermal loads: Monographю Lutsk: RVV Lutsk NTU, 2021. 208 p.
9. Peters, M. DevOps: The Complete Guide to Software Development Lifecycle. Packt Publishing, 2024. 182 p.
10. Shouhong Wang, Hai Wang. Information Systems Analysis and Design (2nd Edition). Systems Acquisition Approach. Universal-Publishers, 2022. 190 p.
11. Tilley, S. Systems Analysis and Design. 13th Edition: Cengage Learning, 2024. 576 p.
12. van Steen, M., Tanenbaum, A.S. Distributed Systems. 4th Edition: Distributed Systems, 2023. 683 p.
13. Xu, A., Lam, S. System Design Interview. An Insider's Guide: Volume 1 & 2. ByteByteGo, 2023. 434 p.
14. Андрущак І. Є., Кошелюк В. А. Особливості захисту хмарного середовища на основі BlockChain. Proceedings of the 10th International scientific and practical conference. CPN Publishing Group. Tokyo, Japan. 2022. pp. 161-168.
15. Андрущак І., Кошелюк В., Ясашний Д. Аудит безпеки оркестрації легких контейнерних платформ. Modern Trends in the Development of Economy, Technology and Industry: Collection of Scientific Papers «International Scientific Unity» with Proceedings of the 3rd International Scientific and Practical Conference. April 9-11, 2025. Toronto, Canada. pp. 169-174.

16. Андрущак І., Кошелюк В., Ясашний Д. Про ефективність використання легких контейнерних платформ. The 15th International scientific and practical conference «Scientific research: integration of science and practice for effective development». Florence, Italy. International Science Group. 2025. pp. 47-52.

17. Гломозда Д.К. Проектування, системний аналіз і розробка корпоративних інформаційних систем: навч. посібник Нац. ун-т «Києво-Могилян. акад.». Київ: НаУКМА, 2023. 296 с.

18. Зінов'єва О.Г. Проектування інформаційних систем з використанням CASE-засобів. Українські студії в європейському контексті. 2023. №7. с. 220-227.

19. Ізмайлова О.В. Проектування інформаційних систем: навчальний посібник. Київ: КНУБА, 2022. 87 с.

20. Литвин В.В., Пасічник В.В., Шаховська Н.Б. Проектування інформаційних систем : навчальний посібник. Львів: Видавництво «Магнолія 2006», 2023. 380 с.

21. Марченко А.В. Проектування інформаційних систем: Посібник для студентів спеціальності 126 Інформаційні системи та технології. Київ: ДУІКТ, 2023. 166 с.

22. Трегуб В.Г. Проектування систем автоматизації. Київ : Видавництво «Ліра-К», 2023. 484 с.

ДЛЯ НОТАТОК

Проектування інформаційних систем: методичні вказівки до лабораторних робіт для здобувачів першого (бакалаврського) рівня вищої освіти освітньої програми «Комп'ютерні науки» галузі знань 12 (F) Інформаційні технології спеціальності 122 (F3) Комп'ютерні науки денної та заочної форм навчання / уклад. В.А. Кошелюк – Луцьк: ЛНТУ, 2025. 40 с.

Видання містить рекомендації до виконання лабораторних робіт з дисципліни «Проектування інформаційних систем».

Призначене для здобувачів першого (бакалаврського) рівня вищої освіти спеціальності 122 (F3) Комп'ютерні науки.

Комп'ютерний набір та верстка: В.А. Кошелюк

Редактор: В.А. Кошелюк

Підп. до друку «_____»_____2025 р.
Формат 60×84/16. Папір офс. Гарн. Таймс.
Ум. друк. арк. ____ Тираж ___ прим. Зам. ___

Відділ іміджу та промоції
Луцького національного технічного університету
43018, м. Луцьк, вул. Львівська, 75