Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Факультет комп'ютерних наук
Кафедра теоретичної та прикладної системотехніки

«Затверджую»
Зав. кафедри теоретичної та
прикладної системотехніки
_____ д.т.н., проф. С. І. Шматков
«___» _____ 2023 р

Пояснювальна записка
до кваліфікаційної роботи
бакалавра

на тему: «Комп'ютерна модель організації створення системного програмного забезпечення з використанням технологій DevOps»

Захищено на засіданні
Атестаційної комісії № 40
протокол № __ від __.06.2023 р.
Оцінка _____ / _____
Голова Атестаційної комісії
_____ Скоб Ю. О.
(підпис) (прізвище та ініціали)

Виконав:
студент 4 курсу, групи КІ– 41
Галузь знань: 12 – Інформаційні технології
Спеціальність: 123 – «Комп'ютерна інженерія»
Чистяков Богдан Юрійович
_____
(підпис)

Керівник:
старший викладач
Мороз Ольга Юріївна
_____
(підпис)

Рецензент:
Завідувач кафедри електроніки та управляючих систем к.ф.-м.н., доцент
_____
(підпис)

Харків – 2023

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел і двох додатків. Загальний обсяг роботи складає 67 сторінок, із яких 45 сторінок основної частини з 11 рисунками, 18 найменуваннями списку використаних джерел та чотирма додатками.

Метою кваліфікаційної роботи є скорочення часу сучасного робочого процесу створення системного програмного забезпечення з використанням технологій DevOps.

Об'єкт дослідження – процес створення системного програмного забезпечення з використанням технологій DevOps.

Предмет дослідження – методи та інструменти створення системного програмного забезпечення.

Проблема, яка вирішується в кваліфікаційній роботі полягає в тому, щоб скориставшись існуючими програмними засобами та Cloud CI/CD інструментами, розробити сервіс, що полегшує і пришвидшує створення системного програмного забезпечення.

Область застосування – розроблений програмний продукт може широко використовуватися в сфері доставки системного програмного забезпечення.

Ключові слова: реалізація сервісу, AWS, DevOps, cloud CI/CD, VCS, Scala, Maven, реліз, оновлення версій.

ABSTRACT

Explanatory note for the bachelor's thesis consists of an introduction, three chapters, conclusions, a list of references, and two appendices. The total volume of the work is 67 pages, of which 45 pages are the main part with 11 figures, 18 references in the list of sources used, and four appendices.

The purpose of the bachelor's thesis is to reduce the time of the modern workflow for creating system software using DevOps technologies.

The object of research is the process of creating system software using DevOps technologies.

The subject of research is the methods and tools for creating system software.

The problem that is solved in the bachelor's thesis is to use existing software tools and Cloud CI/CD tools to develop a service that simplifies and speeds up the creation of system software.

The scope of application is that the developed software product can be widely used in the field of delivering system software.

Keywords: service implementation, AWS, cloud CI/CD, VCS, Scala, Maven, release, version updates.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ І УМОВНИХ ПОЗНАЧЕНЬ

AWS – Amazon Web Services

CI – code integration

CD – code deployment

SQS – simple queue service

SNS – simple notification service

ECS – elastic container service

SDK – Software Development Kit

VCS – version control system

DevOps – Development and Operations

ВСТУП

Уявіть собі світ, у якому власники продукту, розробники, тестувальники, співробітники IT-експлуатації та фахівці з інформаційної безпеки діють спільно, як допомагаючи одне одному, а й забезпечуючи майбутній успіх організації. Працюючи заради спільної мети, вони забезпечують швидке впровадження планових результатів у виробництво (виконуючи в день десятки, сотні або навіть тисячі розгортань коду) і досягають при цьому високого рівня стабільності, стійкості, доступності та безпеки.

У такому світі крос-функціональні групи, безперечно, виконують перевірку своїх припущень, які саме функції особливо потішать користувачів і слугують досягненню цілей організації. Вони не просто дбають про реалізацію функцій, потрібних користувачам, але також активно забезпечують безперебійну роботу та перевірку ланцюжка створення цінностей, не викликаючи при цьому хаосу в управлінні IT-експлуатацією та збоїв у будь-яких внутрішніх та зовнішніх клієнтів.

У той же час тестувальники, співробітники експлуатації та фахівці з інформаційної безпеки постійно намагаються зменшити взаємні тертя всередині команди розробників, створюючи системи, що дають змогу діяти продуктивніше та результативніше. Коли компетентність спеціалістів з якості, співробітників IT-експлуатації та спеціалістів з інформаційної безпеки стає доступною командам, що займаються поставками, автоматизованими інструментами та платформами самообслуговування, вони можуть використовувати все це у повсякденній роботі та перестати залежати від інших команд.

Актуальність роботи. Такий підхід дає організаціям можливість створити надійну систему: невеликі команди швидко та автономно розробляють, тестують та розгортають код, до того ж роблять це надійно та безпечно. Це дозволяє організаціям максимізувати продуктивність розробників, організувати навчання

всередині організації, забезпечити високу задоволеність виконавців та стати переможцем у конкурентній ринковій боротьбі.

Метою дослідження є скорочення часу сучасного робочого процесу створення системного програмного забезпечення з використанням технологій DevOps.

Об'єкт дослідження – процес створення системного програмного забезпечення з використанням технології DevOps.

Предмет дослідження – методи та інструменти створення системного програмного забезпечення.

Методи дослідження:

Завдання дослідження

1. Розглянути та проаналізувати сучасні технології розробки системного програмного забезпечення с використанням технологій DevOps.
2. Дослідити сучасні CI/CD інструменти для побудови комп'ютерної моделі.
3. Розробити комп'ютерну модель за допомогою хмарних CI/CD інструментів і провести її тестування.

РОЗДІЛ 1
АНАЛІЗ СУЧАСНИХ ТЕХНОЛОГІЙ РОЗРОБКИ СИСТЕМНОГО ЗАБЕЗПЕЧЕННЯ С ВИКОРИСТАННЯМ ТЕХНОЛОГІЙ DEVOPS

1.1 Поняття DevOps та його переваги

DevOps – це методологія розробки програмного забезпечення, яка поєднує розробку програмного забезпечення (Dev) та операції інформаційних технологій (Ops), щоб скоротити життєвий цикл розробки систем, одночасно постійно вдосконалюючи функціонал, виправлення та оновлення з врахуванням бізнес-цілей. Підхід DevOps підкреслює комунікацію, співпрацю та інтеграцію між розробниками програмного забезпечення та професіоналами з операційної інформаційної технології.

Традиційний процес розробки програмного забезпечення часто був повільним та незручним, з довгими циклами розробки та тривалими часами розгортання. Цей підхід часто призводив до затримок та помилок, які можуть бути коштовними та часомірними для виправлення. DevOps був розроблений у відповідь на ці виклики з метою оптимізації процесу розробки програмного забезпечення та покращення якості та надійності програмного забезпечення, яке виробляється.

Переваги використання підходу DevOps безліч. По-перше, він дозволяє командам створювати програмне забезпечення вищої якості з меншою кількістю дефектів та помилок, що призводить до збільшення задоволення клієнтів. За допомогою співпраці та комунікації між розробниками та професіоналами з операційної інформаційної технології, DevOps допомагає забезпечити те, що програмне забезпечення ретельно перевірене та протестоване, перш ніж воно буде випущене клієнтам. Це може допомогти у запобіганні таких проблем, як відмови, втрата даних або порушення безпеки.

По-друге, DevOps дозволяє прискорити час введення на ринок, дозволяючи організаціям швидко реагувати на змінні ринкові умови та потреби клієнтів. Автоматизуючи ключові етапи процесу розробки програмного забезпечення,

такі як тестування та розгортання, DevOps може допомогти скоротити час, необхідний для впровадження нових функцій або оновлень. Це може бути особливо важливим в швидкорухливих галузях, таких як технології, де компанії повинні мати можливість швидко реагувати на змінні вимоги клієнтів та ринкові тенденції.

По-третє, DevOps сприяє культурі постійного вдосконалення, де команди стимулюються до експериментів, навчання та інновацій. За допомогою співпраці та комунікації, DevOps допомагає розбити бар'єри між різними командами та відділами в організації. Це може сприяти більшій креативності та інноваціям, а також більш ефективному вирішенню проблем.

По-четверте, DevOps сприяє більшій співпраці та комунікації між розробниками, операторами та іншими зацікавленими сторонами, що призводить до збільшення ефективності та продуктивності. Розбиваючи бар'єри між різними командами та відділами, DevOps допомагає забезпечити, що всі працюють на спільні цілі. Це може допомогти зменшити ризик неправильного спілкування або непорозумінь, які можуть призвести до затримок або помилок.

Нарешті, DevOps допомагає організаціям досягати більшої масштабованості та надійності, дозволяючи їм обробляти збільшений трафік та попит, не жертвуючи продуктивністю чи стабільністю. Автоматизуючи ключові етапи процесу розробки програмного забезпечення, DevOps може допомогти забезпечити, що програмне забезпечення розгортається послідовно та надійно в різних середовищах. Це може бути особливо важливим для організацій, які повинні обробляти великі обсяги трафіку або працюють у складних розподілених середовищах.

1.2 Порівняння традиційної розробки програмного забезпечення та методології DevOps

Традиційна розробка програмного забезпечення та методологія DevOps – це дві різні методології розробки програмного забезпечення. Ось порівняння цих двох методологій:

Процес

Традиційна розробка програмного забезпечення дотримується лінійного процесу, де кожна фаза циклу розробки програмного забезпечення завершується, перш ніж перейти до наступної фази. Цей процес може бути повільним та неефективним, оскільки кожна фаза може займати багато часу для завершення. Натомість, DevOps дотримується ітераційного процесу, де програмне забезпечення розробляється та розгортається поступово, в невеликих інкрементах. Цей процес є швидшим та ефективнішим, оскільки дозволяє отримувати швидкий зворотний зв'язок та ітерацію.

Співпраця

Традиційна розробка програмного забезпечення часто включає в себе команди, які працюють в окремих силосах, де розробники програмного забезпечення та професіонали з операційної діяльності IT працюють незалежно один від одного. Це може призвести до проблем з комунікацією та співпрацею, що може уповільнити процес розробки програмного забезпечення. Натомість, DevOps наголошує на співпраці та комунікації між розробниками програмного забезпечення та професіоналами з операційної діяльності IT. Ця співпраця допомагає забезпечити, що програмне забезпечення розробляється та розгортається вчасно та ефективно.

Автоматизація

Традиційна розробка програмного забезпечення часто базується на ручних процесах, які можуть займати багато часу та містити помилки. Натомість, DevOps наголошує на автоматизації, де ключові частини процесу розробки програмного забезпечення автоматизовані. Ця автоматизація допомагає зменшити ризик помилок, покращити ефективність та забезпечити, що програмне забезпечення розробляється та розгортається в послідовний та надійний спосіб.

Тестування

Традиційна розробка програмного забезпечення часто передбачає тестування в кінці циклу розробки, що може призвести до затримок та проблем з якістю. Натомість, DevOps наголошує на постійному тестуванні, де тестування

інтегрується в кожну фазу циклу розробки програмного забезпечення. Це постійне тестування допомагає забезпечити, що програмне забезпечення розробляється та розгортається вчасно та ефективно, забезпечуючи високий рівень якості.

Розгортання

Традиційна розробка програмного забезпечення часто включає окрему фазу розгортання, де програмне забезпечення розгортається в продакшні після завершення розробки. Ця фаза розгортання може бути повільною та містити помилки, оскільки вона включає ручні процеси та складні конфігурації. Натомість, DevOps наголошує на постійному розгортанні, де програмне забезпечення розгортається в продакшні поступово, в невеликих інкрементах. Це постійне розгортання допомагає забезпечити, що програмне забезпечення розгортається швидко та ефективно, зменшуючи при цьому ризик помилок та простоїв.

Отже, традиційна розробка програмного забезпечення та DevOps - це дві різні методології розробки програмного забезпечення. Тоді як традиційна розробка програмного забезпечення дотримується лінійного процесу та базується на ручних процесах, DevOps дотримується ітераційного процесу та наголошує на співпраці, автоматизації, постійному тестуванні та постійному розгортанні. За допомогою DevOps організації можуть розробляти та розгортати програмне забезпечення швидше та ефективніше, забезпечуючи високий рівень якості та надійності.

Рисунок 1.1 – Основні відмінності розробки ПЗ із методологією DevOps і традиційної

1.3 Основні принципи та практики DevOps

DevOps – це методологія розробки програмного забезпечення, яка підкреслює співпрацю, комунікацію та інтеграцію між розробниками програмного забезпечення та професіоналами з операційної інформаційної технології. Хоча не існує однієї універсальної стратегії DevOps, є кілька ключових принципів та практик, які є важливими для його успіху.

Постійна інтеграція та постійна доставка

Постійна інтеграція (CI) та постійна доставка (CD) є двома ключовими практиками, які є центральними для підходу DevOps. Постійна інтеграція передбачає часту інтеграцію змін коду в спільний репозиторій з автоматичним тестуванням та перевіркою, щоб забезпечити функціональність та відповідність стандартам якості коду. Постійна доставка передбачає автоматизоване розгортання змін коду в середовища продукції з мінімальною ручною інтервенцією. CI/CD допомагає забезпечити швидку та надійну доставку програмного забезпечення з меншою кількістю дефектів та помилок.

Інфраструктура як код

Інфраструктура як код (IaC) є ще одною ключовою практикою в DevOps. IaC передбачає використання коду для автоматизації створення, налаштування та управління IT-інфраструктурою, такою як сервери, мережі та сховища. Шляхом розгляду інфраструктури як коду, організації можуть забезпечити, що їхня інфраструктура є послідовною, повторюваною та масштабованою. IaC також допомагає зменшити ризик помилок або неправильних налаштувань, які можуть призвести до перерв у роботі або інших проблем.

Гнучка розробка

Гнучка розробка - це методологія розробки програмного забезпечення, яка підкреслює гнучкість, співпрацю та зворотний зв'язок від клієнтів. Гнучка розробка передбачає часту доставку невеликих, інкрементальних змін до програмного забезпечення з регулярним зворотним зв'язком від клієнтів та зацікавлених сторін. Цей підхід допомагає забезпечити, що програмне забезпечення розробляється в тісному взаємозв'язку з бізнес-цілями та потребами клієнтів.

Автоматизація

Автоматизація є ключовим принципом в DevOps, оскільки вона допомагає зменшити ручну роботу та покращити ефективність. Автоматизація передбачає використання інструментів та скриптів для автоматизації ключових етапів процесу розробки програмного забезпечення, таких як тестування, розгортання та моніторинг. Шляхом автоматизації цих задач, організації можуть зменшити ризик помилок, покращити якість та доставляти програмне забезпечення швидше та надійніше.

Моніторинг та зворотній зв'язок

Моніторинг та зворотній зв'язок є невід'ємними складовими підходу DevOps. Моніторинг передбачає постійне спостереження за програмним забезпеченням та інфраструктурою для виявлення проблем або аномалій. Зворотній зв'язок передбачає збір та аналіз даних для виявлення областей для поліпшення. Шляхом моніторингу програмного забезпечення та інфраструктури, організації можуть виявляти проблеми на ранніх етапах та швидко реагувати,

щоб запобігти перервам у роботі або іншим проблемам. Шляхом збору та аналізу зворотного зв'язку, організації можуть виявляти області для поліпшення та приймати рішення на основі даних щодо оптимізації свого процесу розробки програмного забезпечення.

Співпраця та комунікація

Співпраця та комунікація є серцем підходу DevOps. DevOps підкреслює важливість розбиття стін між різними командами та відділами та підтримку співпраці та комунікації між розробниками, професіоналами з операційної інформаційної технології та іншими зацікавленими сторонами. Шляхом спільної роботи та обміну знаннями та експертизою, команди можуть доставляти програмне забезпечення високої якості швидше та ефективніше.

Отже, основні принципи та практики DevOps є важливими для його успіху. Шляхом прийняття постійної інтеграції та доставки, інфраструктури як коду, гнучкої розробки, автоматизації, моніторингу та зворотного зв'язку, а також співпраці та комунікації, організації можуть доставляти програмне забезпечення високої якості швидше та ефективніше, покращуючи задоволеність клієнтів та бізнес-результати.

1.4 Інструменти та технології, що використовуються в практиках DevOps.

DevOps – це методологія розробки програмного забезпечення, яка підкреслює співпрацю, комунікацію та інтеграцію між розробниками програмного забезпечення та професіоналами з операційної інформаційної технології. Для підтримки цих цілей DevOps використовує широкий спектр інструментів та технологій, які допомагають автоматизувати ключові етапи процесу розробки програмного забезпечення, покращувати ефективність та забезпечувати якість.

Системи контролю версій

Системи контролю версій (VCS) є важливим елементом DevOps, оскільки вони дозволяють командам керувати та відстежувати зміни в коді з часом. Інструменти VCS, такі як Git, SVN та Mercurial, дозволяють розробникам

співпрацювати щодо змін в коді, відстежувати зміни з часом та повертатися до попередніх версій, якщо це необхідно. Інструменти VCS також забезпечують централізований репозиторій для коду, що допомагає забезпечити, що всі працюють з останньою версією.

Інструменти постійної інтеграції та доставки

Інструменти постійної інтеграції (CI) та постійної доставки (CD) є центральними для підходу DevOps. Інструменти CI/CD, такі як Jenkins, Travis CI та CircleCI, автоматизують ключові етапи процесу розробки програмного забезпечення, включаючи збірку, тестування та розгортання змін в коді. Ці інструменти допомагають забезпечити, що зміни в коді будуть ретельно протестовані та перевірені перед тим, як їх буде розгорнуто в продуктивному середовищі.

Інструменти управління конфігурацією

Інструменти управління конфігурацією, такі як Ansible, Chef та Puppet, використовуються в DevOps для автоматизації налаштування та управління IT-інфраструктурою. Ці інструменти дозволяють організаціям розглядати інфраструктуру як код, що допомагає забезпечити її консистентність, повторюваність та масштабованість. Інструменти управління конфігурацією також допомагають зменшити ризик помилок або неправильного налаштування, які можуть призвести до перерв у роботі або інших проблем.

Інструменти контейнеризації

Інструменти контейнеризації, такі як Docker та Kubernetes, використовуються в DevOps для упакування та розгортання програмних додатків у легкому та портативному форматі. Контейнери дозволяють організаціям легко розгортати додатки на різних середовищах без необхідності в складних конфігураціях або налаштуваннях. Інструменти контейнеризації також допомагають забезпечити, що додатки будуть консистентними та надійними на різних середовищах.

Інструменти моніторингу та логування

Інструменти моніторингу та логування, такі як Nagios, Prometheus та ELK Stack, використовуються в DevOps для моніторингу програмного забезпечення та інфраструктури, а також для збору та аналізу даних про продуктивність та поведінку системи. Ці інструменти допомагають організаціям виявляти проблеми заздалегідь, швидко реагувати для запобігання перерв у роботі або інших проблемах, а також визначати області для покращення.

Хмарні платформи

Хмарні платформи, такі як Amazon Web Services (AWS), Microsoft Azure та Google Cloud Platform, стають все більш популярними в DevOps, оскільки вони забезпечують гнучку, масштабовану та ефективну інфраструктуру для розробки та розгортання програмного забезпечення. Хмарні платформи дозволяють організаціям легко надавати та керувати IT-ресурсами, такими як сервери, сховища даних та бази даних, без необхідності в складних налаштуваннях або обслуговуванні.

У підсумку, DevOps ґрунтується на широкому спектрі інструментів та технологій для автоматизації ключових етапів процесу розробки програмного забезпечення, покращення ефективності та забезпечення якості. Від систем контролю версій та інструментів постійної інтеграції та доставки до інструментів управління конфігурацією, інструментів контейнеризації, інструментів моніторингу та логування та хмарних платформ, ці інструменти допомагають організаціям доставляти високоякісне програмне забезпечення швидше та ефективніше, покращуючи задоволеність клієнтів та бізнес-результати.

1.5 Виклики та обмеження DevOps у розробці системного програмного забезпечення.

Хоча DevOps став все більш популярним у останні роки, все ще існують деякі виклики та обмеження, пов'язані з його впровадженням у розробку системного програмного забезпечення. Ось деякі з ключових викликів та обмежень DevOps у розробці системного програмного забезпечення:

Культурні виклики

Один з найбільших викликів впровадження DevOps у розробку системного програмного забезпечення полягає в культурному аспекті. DevOps потребує тісної співпраці та комунікації між розробниками програмного забезпечення та професіоналами з операційної інфраструктури, що може бути складним у організаціях з відділами-силосами або відсутністю довіри між командами. Щоб подолати ці культурні виклики, організації повинні створити культуру співпраці та відкритої комунікації, та забезпечити, що всі спрямовані на спільні цілі та завдання.

Технічні виклики

DevOps також створює деякі технічні виклики у розробці системного програмного забезпечення. Наприклад, деякі системи старого зразка можуть бути несумісними з інструментами та технологіями DevOps, що може ускладнити їх впровадження в таких середовищах. Аналогічно, деякі організації можуть не мати необхідної інфраструктури або ресурсів для підтримки DevOps, що може ускладнити досягнення бажаних результатів.

Виклики щодо безпеки та відповідності

DevOps також може створювати деякі виклики щодо безпеки та відповідності в розробці системного програмного забезпечення. Наприклад, DevOps потребує високого рівня автоматизації, що може ускладнити забезпечення виконання вимог щодо безпеки та відповідності. Крім того, DevOps може введення нових ризиків безпеки, таких як використання інструментів сторонніх постачальників або витік конфіденційної інформації в тестових або розробних середовищах. Щоб подолати ці виклики, організації повинні забезпечити, що безпека та відповідність вбудовані в процес DevOps від самого початку, та що всі зацікавлені сторони знають свої обов'язки.

Виклики управління персоналом

DevOps потребує високого рівня вмінь та експертизи, що може бути складним для знаходження в деяких організаціях. Крім того, впровадження DevOps може потребувати нових ролей або обов'язків, що може бути складним

для заповнення або інтеграції в існуючі команди. Щоб подолати ці виклики, організації повинні інвестувати в програми навчання та розвитку, та забезпечити, що вони мають необхідні ресурси для підтримки впровадження DevOps.

Виклики вартості

Нарешті, DevOps також може створювати деякі виклики щодо вартості в розробці системного програмного забезпечення. Наприклад, впровадження DevOps може потребувати нових інструментів та технологій, що можуть бути дорогими для придбання та підтримки. Крім того, автоматизація ключових частин процесу розробки програмного забезпечення може потребувати додаткових ресурсів або інфраструктури, що також може бути дорогим. Щоб подолати ці виклики, організації повинні ретельно оцінити витрати та користь від DevOps, та забезпечити, що вони мають чітке розуміння повернення інвестицій.

На завершення можна сказати, що, хоча DevOps має багато переваг у розробці системного програмного забезпечення, також існують деякі виклики та обмеження, які потрібно вирішувати. Розуміючи ці виклики та обмеження, організації можуть вживати заходів для їх подолання та забезпечення успішного впровадження DevOps.

1.6 Майбутні тенденції в DevOps та їх вплив на розробку системного програмного забезпечення

DevOps став все більш популярним у останні роки, і очікується, що його використання продовжить зростати у майбутньому. Ось деякі з ключових майбутніх тенденцій в DevOps та їх потенційний вплив на розробку системного програмного забезпечення:

Хмарно-орієнтований DevOps

Хмарно-орієнтований DevOps є тенденцією, яка передбачає використання хмарно-орієнтованих технологій та архітектур для підтримки підходу DevOps до розробки програмного забезпечення. Очікується, що ця тенденція буде продовжувати зростати у майбутньому, оскільки все більше організацій переносить свої системи та додатки до хмари. Хмарно-орієнтований DevOps

може допомогти покращити ефективність, масштабованість та гнучкість, а також зменшити витрати та покращити надійність.

Штучний інтелект та машинне навчання

Очікується, що штучний інтелект та машинне навчання будуть відігравати все більш важливу роль у DevOps у майбутньому. Ці технології можуть допомогти автоматизувати ключові частини процесу розробки програмного забезпечення, такі як тестування та розгортання, а також покращити точність та надійність цих процесів. Крім того, ШІ та МН можуть допомогти виявляти патерни та тенденції в даних про розробку програмного забезпечення, що може бути використано для покращення загальної якості та ефективності процесу розробки програмного забезпечення.

DevSecOps

DevSecOps є тенденцією, яка передбачає інтеграцію безпеки в процес DevOps. Очікується, що ця тенденція буде продовжувати зростати у майбутньому, оскільки організації стають все більш занепокоєні питаннями безпеки та відповідності. DevSecOps може допомогти забезпечити, що безпека та відповідність вбудовані в процес розробки програмного забезпечення від самого початку, а не додаватися як додатковий елемент пізніше. Це може допомогти зменшити ризик витоку даних та інших проблем, а також покращити загальну якість та надійність програмного забезпечення.

NoOps

NoOps є тенденцією, яка передбачає використання автоматизації для усунення потреби у професіоналах з операційної діяльності IT в процесі розробки програмного забезпечення. Ця тенденція все ще знаходиться на початкових стадіях, але вона має потенціал революціонізувати спосіб, яким розробляється та розгортається програмне забезпечення. NoOps може допомогти покращити ефективність, зменшити витрати та усунути потребу в ручному втручанні в процес розробки програмного забезпечення.

Low-Code/No-Code DevOps

Low-code/no-code DevOps є тенденцією, яка передбачає використання інструментів та платформ візуального програмування для підтримки підходу DevOps до розробки програмного забезпечення. Очікується, що ця тенденція буде продовжувати зростати у майбутньому, оскільки все більше організацій шукають способи скорочення часу та ресурсів, необхідних для розробки та розгортання програмного забезпечення. Low-code/no-code DevOps може допомогти покращити ефективність, зменшити витрати та покращити загальну якість та надійність програмного забезпечення.

На завершення можна сказати, що є багато майбутніх тенденцій в DevOps, які мають потенціал революціонізувати спосіб розробки системного програмного забезпечення. За допомогою прийняття цих тенденцій та технологій, організації можуть покращити ефективність, зменшити витрати та покращити загальну якість та надійність свого програмного забезпечення.

Висновки за розділом 1

Було розглянуто поняття DevOps та його переваги. Було проведено порівняння традиційної розробки програмного забезпечення та методології DevOps. Було розглянуто основні принципи та практики DevOps, а також інструменти та технології, що використовуються в практиках DevOps. Було досліджено виклики та обмеження DevOps у розробці системного програмного забезпечення та майбутні тенденції в DevOps та їх вплив на розробку системного програмного забезпечення.

РОЗДІЛ 2
АНАЛІЗ СУЧАСНИХ ХМАРНИХ CI/CD ІНСТРУМЕНТІВ ДЛЯ ПОБУДОВИ КОМП'ЮТЕРНОЇ МОДЕЛІ

2.1 Cloud CI/CD та його важливості в комп'ютерному моделюванні

Cloud CI/CD – це методологія розробки та розгортання програмного забезпечення, яка використовує ресурси хмарних обчислень та практики постійної інтеграції та постійної доставки (CI/CD). Ця методологія стала все більш популярною в останні роки, і її важливість в комп'ютерному моделюванні не може бути переоцінена.

Cloud CI/CD включає в себе використання ресурсів хмарних обчислень, таких як віртуальні машини та контейнери, для підтримки процесу розробки та розгортання програмного забезпечення. Ця методологія також включає в себе використання практик постійної інтеграції та постійної доставки, де програмне забезпечення розробляється та розгортається поступово, в невеликих інкрементах. Cloud CI/CD допомагає покращити ефективність, знизити витрати та покращити загальну якість та надійність програмного забезпечення.

Cloud CI/CD особливо важливий в комп'ютерному моделюванні, де розробка та розгортання програмного забезпечення може бути складним та часовим. Шляхом використання ресурсів хмарних обчислень та практик постійної інтеграції та постійної доставки, організації, які займаються комп'ютерним моделюванням, можуть розробляти та розгортати програмне забезпечення швидше та ефективніше, забезпечуючи високий рівень якості та надійності.

Cloud CI/CD також допомагає покращити співпрацю та комунікацію між розробниками програмного забезпечення та професіоналами з операційної діяльності IT, що є важливим в комп'ютерному моделюванні. Шляхом спільної роботи в гнучкий та ітеративний спосіб, ці команди можуть забезпечити, що програмне забезпечення розробляється та розгортається вчасно та ефективно,

забезпечуючи при цьому відповідність потребам організації та її зацікавлених сторін.

Нарешті, Cloud CI/CD може допомогти зменшити ризик помилок та простою в комп'ютерному моделюванні. Шляхом автоматизації ключових етапів процесу розробки та розгортання програмного забезпечення, організації можуть зменшити ризик людської помилки та забезпечити, що програмне забезпечення розгортається відповідно та надійно. Це може допомогти покращити загальну якість та надійність програмного забезпечення, зменшити ризик простою та інших проблем.

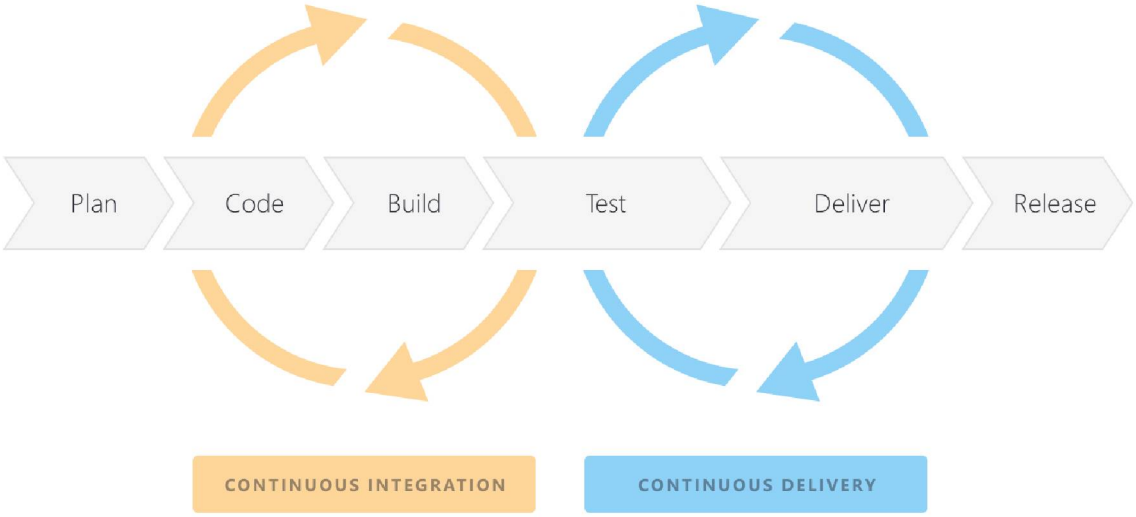


Рисунок 2.1 – Спростований CI/CD пайплайн

2.2 Основні фактори, які слід враховувати при виборі інструменту Cloud CI/CD для комп'ютерного моделювання

Вибір правильного інструменту Cloud CI/CD для комп'ютерного моделювання є важливим рішенням, яке може впливати на ефективність, надійність та якість процесу розробки та розгортання програмного забезпечення. Ось деякі ключові фактори, які слід враховувати при виборі інструменту Cloud CI/CD:

Інтеграція з існуючими інструментами та сервісами

При виборі інструменту Cloud CI/CD важливо враховувати, наскільки добре він інтегрується з існуючими інструментами та сервісами. Це включає платформи управління вихідним кодом, хмарні обчислювальні сервіси та інші інструменти, які використовуються в процесі розробки та розгортання програмного забезпечення. Інтеграція може допомогти спростити процес розробки та розгортання, зменшити кількість помилок та покращити загальну ефективність.

Масштабованість та гнучкість

Масштабованість та гнучкість також є важливими факторами, які слід враховувати при виборі інструменту Cloud CI/CD. Інструмент повинен мати можливість масштабуватися вгору або вниз, залежно від розміру та складності проекту. Він також повинен бути достатньо гнучким, щоб враховувати зміни в процесі розробки та розгортання, а також зміни в потребах та вимогах організації.

Безпека та відповідність

Безпека та відповідність є критичними факторами, які слід враховувати при виборі інструменту Cloud CI/CD. Інструмент повинен мати надійні функції безпеки, такі як шифрування, контроль доступу та моніторинг, щоб захистити конфіденційні дані та запобігти несанкціонованому доступу. Він також повинен відповідати відповідним регуляторним вимогам та стандартам, таким як GDPR, HIPAA та PCI DSS.

Модель вартості та ціноутворення

Модель вартості та ціноутворення інструменту Cloud CI/CD є також важливим фактором для розгляду. Деякі інструменти можуть мати вартість на користувача, на проект або на основі використання. Важливо розуміти модель ціноутворення та пов'язані з нею витрати, а також будь-які потенційні приховані витрати, такі як плата за додаткові функції або послуги.

Підтримка та документація

Нарешті, підтримка та документація є важливими факторами для розгляду при виборі інструменту Cloud CI/CD. Інструмент повинен мати комплексну

документацію, навчальні посібники та ресурси підтримки, такі як форуми або служба підтримки клієнтів, щоб допомогти користувачам у вирішенні проблем та отриманні максимальної користі з інструменту.

Заключно, вибір правильного інструменту Cloud CI/CD для комп'ютерного моделювання потребує ретельного розгляду кількох факторів, включаючи інтеграцію, масштабованість та гнучкість, безпеку та відповідність, модель вартості та ціноутворення та підтримку та документацію. Вибравши правильний інструмент, організації можуть покращити ефективність, надійність та якість процесу розробки та розгортання програмного забезпечення.

2.3 Огляд та порівняння сучасних інструментів Cloud CI/CD та їх функцій

Робочі процеси або стандартні кроки процесу CI-CD включають планування, написання коду, збірку та тестування. Цей набір операцій відомий як конвеєр. Кожен інструмент використовує ці етапи або додаткові кроки.

Нижче перераховані найкращі інструменти CI CD, які слід врахувати:

Buddy

Buddy – це творче рішення CI/CD для веб-розробників, яке допомагає полегшити перехід до DevOps. Він розробляє, тестує та розгортає програмне забезпечення за допомогою конвеєрів доставки. Він підтримує всі широко використовувані мови, фреймворки та інструменти управління завданнями.

Переваги Buddy:

- Конфігурація за 15 хвилин з простим та інформативним інтерфейсом користувача
- Розгортання, які виконуються миттєво в залежності від змін
- Збірки виконуються в ізольованих контейнерах з кешованими залежностями
- Підтримує всі широко використовувані мови, фреймворки та інструменти управління завданнями
- Присвячений список дій Docker/Kubernetes
- Підтримує конфігурацію YAML та паралелізм

Jenkins

Jenkins - це крос-платформний інструмент для безперервної інтеграції, який використовується для постійної збірки та тестування програмних додатків. Для розробки Jenkins використовують мову програмування Java. Він пропонує тестування та звітність в режимі реального часу. Його можна використовувати для виконання безперервних збірок, запуску тестів або виконання повторюваних завдань.

Розробники можуть розгорнути Jenkins у контейнері сервлетів. Завдяки його простій установці та конфігурації, це один з найкращих інструментів CI CD. Крім того, Jenkins можна розширювати за допомогою плагінів, які надають нескінченні можливості для безперервних розгортань.

Переваги Jenkins:

- Це відкритий інструмент зі сильним співтовариством.
- Його легко налаштувати.
- Він повністю безкоштовний.
- Оскільки він створений за допомогою Java, він може працювати на всіх основних платформах.

Buildbot

Один із найкращих інструментів CI CD – Buildbot, був створений на Python і працює з Twisted framework. Buildbot є відкритою платформою, яка автоматизує складні процеси тестування та розгортання додатків. Цей інструмент популярний тому, що він сприяє паралельному та розподіленому виконанню стратегій на кількох платформах.

Переваги Buildbot:

- Виконання збірок на кількох платформах-слейвах
- Необмежений процес збірки: підтримує проекти, написані на C, Python тощо
- Швидко знаходить проблему збірки та тестує її кожного разу, коли щось змінюється

Perforce Helix

Уніфікована, відкрита та адаптивна платформа CI Perforce Helix підтримує всі документовані API. Інструмент включає багато функцій, таких як управління додатками, планування та управління Agile, співпрацю розробників та управління відкритим кодом. Цей потужний двигун версіонування керує та захищає будь-який тип та розмір файлу. Він також забезпечує реплікацію для високопродуктивних розробок та збірок.

Perforce Helix може бути налаштований для конкретних команд розробників для робочих процесів, розширеної функціональності сервера та пакетних або злиттєвих шаблонів. Він також зберігає історію для кожної зміни.

Переваги Perforce Helix:

- Масштабний кодову базу
- Підтримує графічні або бінарні матеріали, які не є кодом
- Дозволяє залежності всередині та між компонентами коду

Bitbucket

Bitbucket Pipelines - це вбудований інструмент DevOps CI CD. Базуючись на файлі конфігурації в репозиторії, він дозволяє автоматично будувати, тестувати та навіть розгортати ваш код. Для кожної дії на Bitbucket генеруються хмарні контейнери. Ці контейнери можуть використовуватися для виконання команд з усіма перевагами нових конфігурацій системи, підходящих за вимогами.

Переваги Bitbucket:

- Інтелектуальний семантичний пошук
- Корисні функції вікі
- Надає міцну інтеграцію з Jira та Trello

TeamCity

TeamCity є одним з кращих інструментів CI CD, які доступні на ринку з безліччю можливостей. Архітектура та налаштування інструменту автоматично виконують завдання, такі як покриття коду, аналіз коду, уникнення дублювання коду та багато інших. TeamCity архівує або резервує всі зміни, помилки та збірки для подальшого використання.

Переваги TeamCity:

- Пропонує безліч способів повторного використання налаштувань батьківського проекту в дочірньому проекті
- Використовує коди джерел двох окремих систем контролю версій (VCS) для однієї збірки
- Може ідентифікувати неактивні збірки
- Надає методи для звернення уваги до певних збірок для подальшого використання

Circle CI

Інший адаптивний інструмент CI CD – Circle CI, який може бути в Docker-кластері, сервері Python API або середовищі мобільних додатків для кросплатформи. Цей інструмент допомагає зменшити накладні витрати на наявність присвяченого сервера, оскільки він базується на хмарі.

Circle CI використовує найкращі процеси збірки, тестування, налагодження та розгортання для процесу безперервної інтеграції. На кінці він генерує повідомлення.

Переваги Circle CI:

- Включає SSH до будь-якого завдання для вивчення проблем збірки
- Прискорює виконання завдань, дозволяючи паралельно виконувати їх у конфігураційному файлі config.yml
- Дозволяє повторно використовувати дані з попередніх операцій, налаштувавши кешування всього за два прості ключі
- Налаштуйте самохостед ранери для підтримки певної платформи.

AWS Pipeline

Для скорочення часу розробки та доставки продукту AWS пропонує повний набір інструментів DevOps CI CD. Кожного разу, коли змінюється код, AWS CodePipeline автоматизує процес збірки, тестування та розгортання на

основі встановленої моделі релізу. Це дозволяє швидко та ефективно надавати функції та оновлення.

Канали для коду можуть підключатися до інших сервісів. Це можуть бути сторонні товари, такі як GitHub або сервіси AWS, такі як Amazon Simple Storage Service. AWS CodePipeline може вирішувати різні випадки використання розробки та операцій.

Переваги AWS Pipeline:

- Підтримує збірку, тестування та компіляцію коду за допомогою AWS CodeBuild
- Контейнерні додатки надсилаються в хмару безперервно
- Збільшення продуктивності розробника

Travis CI

Travis CI - популярна відкрита платформа тестування. Підтримується безліч платформ, включаючи Linux, Mac та iOS. iOS буде корисним, якщо ви вже використовуєте матрицю збірки для тестування різних версій. У порівнянні з цим, система матриці на Linux та Mac розрізняє регістр і повертає записи каталогу в порядку, в якому вони існують. Крім того, Travis CI може бути синхронізований з GitHub для проведення тестування.

Переваги Travis CI:

- Створює артефакти та оцінює якість коду
- Просте розгортання хмарних сервісів
- Може виявляти як незначні, так і значні модифікації коду

GitLab

GitLab – це набір інструментів для керування різними етапами життєвого циклу розробки програмного забезпечення. Основна пропозиція - менеджер репозиторіїв Git для вебу з інструментами для відстеження проблем, аналітики та вікі.

З кожним комітом або push в GitLab ви маєте можливість запускати збірки, запускати тести та розгортати код. Завдання можуть бути створені на іншому сервері, у віртуальній машині або за допомогою контейнерів Docker.

Переваги GitLab:

- Простота налаштування
- Безпека вихідного коду
- Автоматизація каналів
- Планування розгортання

Нижче наведено компіляцію інструментів CI CD. Кожен з цих інструментів кращий у своєму роді. Ви можете вибрати інструмент, який найкраще відповідає вашим вимогам на основі ваших потреб.

- Buddy корисний для налаштування повідомлень, які повідомляють вас про неуспішні збірки або розгортання. Він також інтегрується з Shopify, WordPress, Google, Digital Ocean, AWS та більшими.
- У Jenkins кожна зміна вихідного коду будується та тестується перед тим, як бути збереженою. Тому розробники повинні лише зосередитися на конкретному коміті, а не на перегляді всього вихідного коду. Як результат, нове програмне забезпечення випускається часто.
- Buildbot базується на системі планування завдань, яка виконує роботу та створює звіт про тестування, якщо є доступні ресурси. Крім того, він має майстер-слейв-систему, де майстер відстежує зміни та інформацію для користувача або розробника.
- Perforce Helix швидко виявляє будь-яку загрозу, передбачає ризики та визначає наслідки цього ризику або небезпеки.
- Bitbucket повністю безкоштовний для максимум п'яти користувачів. Оскільки ви можете мати необмежену кількість приватних та публічних репозиторіїв, не дивно, що на платформі є кілька важливих проектів з відкритим кодом.
- CircleCI має розширену кешування, кешування шарів Docker та класи ресурсів для роботи на швидших серверах, які можуть бути налаштовані для запуску дуже складних каналів швидко.

- AWS pipeline збільшує продуктивність, звільняючи розробників від ручних дій та сприяючи звичкам, які допомагають зменшити кількість помилок та багів, які відправляються споживачам.
- Ручні спрацювання GoCD можуть розгорнути будь-яку версію програмного забезпечення та покращити надійність виробництва. Крім того, він генерує звіт про тестування та запускає його на різних мовах.
- Travis CI надає повну видимість користувачам під час тестування. Використовуючи його відмінні засоби командного рядка та API, він також виконує паралельні тестові запуски. Підтримуються запити на злиття та потік побудови гілки.
- GitLab дуже популярний серед людей, які працюють з контейнерами на основі Docker та мікросервісами. Якщо ви належите до цієї категорії, GitLab – це безумовна рекомендація. GitLab надає відмінний контроль пайплайну в цілому та фантастичні інтеграції.

Висновки за розділом 2

У другому розділі дипломної роботи було розглянуто Cloud CI/CD та його важливість в комп'ютерному моделюванні. Були описані основні фактори, які слід враховувати при виборі інструменту Cloud CI/CD для комп'ютерного моделювання, а також проведено огляд та порівняння сучасних інструментів Cloud CI/CD та їх функцій.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ КОМП'ЮТЕРНОЇ МОДЕЛІ ТА ЇЇ НАЛАШТУВАННЯ

3.1 Scala, Maven та AWS SDK: інструменти для розробки високоякісних додатків

Scala – потужна мова програмування, яка розроблена для того, щоб бути лаконічною, виразною та масштабованою. Це об'єктно-орієнтована мова, яка також є функціональною, що означає, що вона підтримує як імперативний, так і декларативний стилі програмування. Scala особливо добре підходить для розробки високопродуктивних розподілених додатків, які можуть масштабуватися для обробки великих обсягів трафіку та даних.

Однією з ключових переваг Scala є її здатність інтегруватися з іншими технологіями та фреймворками, такими як Apache Spark, Akka та Play. Ці інструменти можуть допомогти розробникам будувати складні розподілені системи, які можуть обробляти широкий спектр використання, від обробки даних в реальному часі до розробки веб-додатків.

Scala також пропонує ряд функцій, які можуть допомогти покращити якість та надійність програмного забезпечення, такі як виведення типів, зіставлення зразків та незмінність. Ці функції можуть допомогти зменшити ризик помилок та помилок, а також зробити його легше для розробників писати та підтримувати високоякісний код.

Scala пропонує ряд функцій та переваг, які роблять її популярним вибором для розробки високоякісних додатків. Деякі з основних функцій та переваг Scala включають:

- Лаконічність та виразність: Scala розроблена для того, щоб бути лаконічною та виразною, що означає, що розробники можуть писати код, який легше читати та розуміти. Scala також підтримує ряд функцій, таких як виведення типів, зіставлення зразків та функції вищих порядків, що можуть допомогти зменшити кількість заготовочного коду, який потрібен.

- Масштабованість та продуктивність: Scala розроблена для того, щоб бути масштабованою та продуктивною, що означає, що вона може обробляти великі обсяги даних та трафіку без втрати продуктивності. Scala також підтримує паралельне та розподілене програмування, що може допомогти покращити продуктивність додатків, які потребують обробки великих обсягів даних.
- Сумісність з Java: Scala розроблена для того, щоб бути сумісною з Java, що означає, що вона може використовувати великий набір бібліотек та інструментів, які доступні в екосистемі Java. Це може допомогти зменшити час та зусилля, необхідні для розробки додатків, оскільки розробники можуть повторно використовувати існуючий Java-код та бібліотеки.
- Функціональні функції програмування: Scala підтримує ряд функцій функціонального програмування, таких як незмінність, функції вищих порядків та лінива оцінка. Ці функції можуть допомогти зменшити ризик помилок та помилок, а також зробити його легше для розробників писати та підтримувати високоякісний код.
- Функції об'єктно-орієнтованого програмування: Scala також підтримує ряд функцій об'єктно-орієнтованого програмування, таких як успадкування, інкапсуляція та поліморфізм. Ці функції можуть допомогти покращити модульність та підтримуваність коду, а також зробити його легше для розуміння складних систем.

Scala - потужна мова програмування, яку можна використовувати для розробки високоякісних додатків, які є масштабованими, продуктивними та надійними. Використовуючи функції та переваги Scala, розробники можуть оптимізувати процес розробки, зменшити ризик помилок та помилок, і забезпечити, що програмне забезпечення повністю протестоване та перевірене, перш ніж воно буде випущене клієнтам. Незалежно від того, чи ви будуєте веб-додаток, оброблюєте дані чи створюєте модель машинного навчання, Scala може

допомогти вам досягти ваших цілей та забезпечити, що програмне забезпечення відповідає потребам ваших користувачів.

Maven – потужний інструмент автоматизації збірки, призначений для спрощення процесу створення та управління Java-проектами. Він надає ряд функцій, які можуть допомогти розробникам оптимізувати процес розробки, такі як управління залежностями, контроль версій та звіт проекту.

Однією з ключових переваг Maven є його здатність керувати залежностями між різними компонентами проекту. Це може допомогти забезпечити, що програмне забезпечення будується послідовно та надійно, навіть коли його розробляють кілька команд або в різних середовищах.

Maven також пропонує ряд плагінів та розширень, які можуть допомогти автоматизувати загальні завдання, такі як тестування, упакування та розгортання. Це може допомогти зменшити час та зусилля, необхідні для створення та випуску програмного забезпечення, а також покращити якість та надійність створеного програмного забезпечення.

Maven пропонує ряд функцій та переваг, які роблять її популярним вибором для створення високоякісних додатків. Деякі з основних функцій та переваг Maven включають:

- Управління залежностями: Maven надає потужну систему управління залежностями, яка може допомогти забезпечити послідовність та надійність побудови програмного забезпечення, навіть коли його розробляють кілька команд або в різних середовищах. Maven може автоматично завантажувати та керувати залежностями, а також вирішувати конфлікти між різними версіями однієї й тієї ж залежності.
- Автоматизація збірки: Maven може автоматизувати загальні завдання збірки, такі як компіляція коду, запуск тестів та упакування програмного забезпечення. Це може допомогти зменшити час та зусилля, необхідні для створення та випуску програмного забезпечення, а також покращити якість та надійність створеного програмного забезпечення.

- Звіт проекту: Maven може генерувати різноманітні звіти, які можуть допомогти розробникам зрозуміти стан та якість їх проектів. Ці звіти можуть містити інформацію про покриття коду, результати тестування та інші метрики, які можуть допомогти покращити якість та надійність програмного забезпечення.
- Сумісність з іншими інструментами: Maven розроблений для того, щоб бути сумісним з різноманітними інструментами та фреймворками, такими як Eclipse, IntelliJ та Spring. Це може допомогти зменшити час та зусилля, необхідні для створення та управління проектами, оскільки розробники можуть використовувати існуючі інструменти та фреймворки.
- Розширюваність: Maven дуже розширюваний, що означає, що розробники можуть налаштовувати та розширювати інструмент, щоб задовольнити свої потреби. Це може включати створення власних плагінів, додавання нових цілей або інтеграцію з іншими інструментами та фреймворками.

Maven – потужний інструмент автоматизації збірки, який може допомогти розробникам оптимізувати процес розробки та забезпечити, що програмне забезпечення повністю протестоване та перевірене, перш ніж воно буде випущене клієнтам. Використовуючи функції та переваги Maven, розробники можуть автоматизувати загальні завдання збірки, керувати залежностями та генерувати звіти, які можуть допомогти покращити якість та надійність програмного забезпечення. Незалежно від того, чи ви будуєте невелику бібліотеку чи велику корпоративну систему, Maven може допомогти вам досягти ваших цілей та забезпечити, що програмне забезпечення відповідає потребам ваших користувачів.

AWS SDK – це набір інструментів та бібліотек, призначених для допомоги розробникам у створенні додатків, які працюють на платформі хмарних обчислень Amazon Web Services (AWS). Він надає ряд функцій, які можуть

допомогти спростити процес створення та розгортання хмарних додатків, такі як аутентифікація, контроль доступу та зберігання даних.

Однією з ключових переваг AWS SDK є його здатність інтегруватися з іншими сервісами AWS, такими як Amazon S3, Amazon DynamoDB та Amazon EC2. Це може допомогти розробникам створювати потужні та масштабовані додатки, які можуть обробляти широкий спектр використання, від обробки даних до машинного навчання.

AWS SDK також пропонує ряд функцій, які можуть допомогти покращити безпеку та надійність хмарних додатків, такі як шифрування, моніторинг та стійкість до відмов. Ці функції можуть допомогти зменшити ризик витоку даних або простою, а також забезпечити, що програмне забезпечення є високодоступним та масштабованим.

3.2 Використання хмарних сервісів AWS Serverless для розробки та виконання додатків

AWS Serverless – це потужний набір хмарних сервісів, який може допомогти розробникам створювати та розгортати додатки без необхідності управління серверами або інфраструктурою. AWS Serverless надає ряд функцій та переваг, які можуть допомогти покращити ефективність та масштабованість розробки та виконання додатків.

AWS Serverless пропонує ряд функцій та переваг, які роблять його популярним вибором для створення та розгортання хмарних додатків. Деякі з основних функцій та переваг AWS Serverless включають:

- Економічність: AWS Serverless може допомогти знизити витрати на розробку та виконання додатків, оскільки розробники платять лише за ресурси, які фактично використовуються. Це може допомогти зменшити необхідність у дорогому обладнанні або інфраструктурі, а також покращити масштабованість та надійність додатків.
- Масштабованість та продуктивність: AWS Serverless розроблений для того, щоб бути високомасштабованим та продуктивним, що означає, що він може обробляти великі обсяги даних та трафіку без

погіршення продуктивності. AWS Serverless також підтримує паралельне та розподілене програмування, що може допомогти покращити продуктивність додатків, які потребують обробки великих обсягів даних.

- Простота розгортання: AWS Serverless може спростити процес розгортання, оскільки розробники не повинні керувати серверами або інфраструктурою. Це може допомогти зменшити час та зусилля, необхідні для розгортання додатків, а також покращити надійність та масштабованість розгорнутих додатків.
- Інтеграція з іншими сервісами AWS: AWS Serverless може інтегруватися з різноманітними сервісами AWS, такими як Amazon S3, Amazon DynamoDB та Amazon API Gateway. Це може допомогти спростити процес розробки, оскільки розробники можуть використовувати існуючі сервіси та інструменти AWS.
- Простота обслуговування: AWS Serverless може спростити процес обслуговування, оскільки розробники не повинні керувати серверами або інфраструктурою. Це може допомогти зменшити час та зусилля, необхідні для обслуговування додатків, а також покращити надійність та масштабованість обслуговуваних додатків.

AWS Serverless може бути використаний для розробки різноманітних хмарних додатків, від веб-додатків до обробки даних та моделей машинного навчання. Для використання AWS Serverless для розробки та виконання додатків, розробники повинні дотримуватися кількох кращих практик, таких як:

- Вибір правильного сервісу AWS Serverless: AWS надає широкий спектр сервісів Serverless, кожен зі своїми перевагами та недоліками, тому розробники повинні ретельно оцінювати свої варіанти перед вибором сервісу.
- Написання функцій: AWS Serverless додатки будуються за допомогою функцій, які є невеликими фрагментами коду, що виконують певну задачу. Розробники повинні писати функції, які оптимізовані для

продуктивності та масштабованості, і повинні дотримуватися кращих практик, таких як написання чистого та модульного коду.

- Використання AWS Lambda: AWS Lambda - це сервіс обчислення Serverless, який може допомогти спростити процес створення та розгортання Serverless додатків. Розробники повинні використовувати AWS Lambda для запуску своїх функцій, оскільки Lambda може автоматично масштабувати функції в залежності від попиту та допомагає знизити витрати на розробку та виконання додатків.
- Використання інших сервісів AWS Serverless: Розробники повинні використовувати інші сервіси AWS Serverless, такі як Amazon S3, Amazon DynamoDB та Amazon API Gateway, для створення та розгортання своїх додатків. Ці сервіси можуть допомогти спростити процес розробки, оскільки розробники можуть використовувати існуючі сервіси та інструменти AWS.
- Дотримання кращих практик: Нарешті, розробники повинні дотримуватися кращих практик, таких як написання чистого та модульного коду, використання систем контролю версій та документування коду. Дотримуючись цих кращих практик, розробники можуть забезпечити, що програмне забезпечення легко зрозуміти, обслуговувати та розширювати з часом.

3.3 Розробка та аналіз програмного сервісу

Під час розробки програмного забезпечення завжди настає момент коли потрібно донести оновлення до кінцевого користувача. Зазвичай цей процес називається релізом. У розробників прийнято тестові або проміжні версії додатків які знаходяться у активній розробці позначати як SNAPSHOT, а під час релізу цей постфікс прибирається і залишається лише чиста версія артефакту.

На великих проектах із декількома мільйонами рядків коду і десятками репозиторіїв в обслуговуванні, зазвичай велика кількість власних бібліотек і розширень, в яких час витрачений на оновлення версій може досягати нерозумних об’ємів.

Розроблений сервіс пропонує полегшити роботу команди у таких випадках, автоматизувавши оновлення версій самого артефакту і всіх його залежностей. Основна логіка підходить тільки для проектів які використовують збірник проектів Maven.

3.3.1 Побудова основної архітектури сервісу

Під час побудови архітектури були взяті до уваги наступні критерії:

- Вартість – так як релізи відбуваються не кожного дня і потребують зазвичай схвалення від старших менеджерів, цей процес повинен буди за вимогою, тобто немає необхідності мати постійно працюючий сервер із сервісом.
- Довготривалість – основною ціллю додатку є оновлення тестових версій на релізні і генерація нових артефактів. Це займає досить багато часу (в залежності від кількості модулів в проекті). Складається процес з наступних кроків:

1. По-перше для оновлення версії потрібно склонувати репозиторій із кодом локально.
2. Після зробити зміни в відповідних файлах.
3. Потім зафіксувати їх та надіслати на віддалений репозиторій.
4. Далі потрібно у певному порядку запустити інструмент збірки, тому що проект може мати величезну кількість залежностей, які повинні бути зібрані у першу чергу.

- Простота у використанні – сервісом повинні із легкістю користуватися розробники, тестувальники і бізнес команда.
- Обробка помилок та сповіщення – важливо мати можливість у потрібний момент виявити проблему, щоб вирішити її у найшвидший спосіб. Це тільки посилюється під час роботи із релізами, тому що вони проводяться у час коли система має найменше навантаження.

- Відмовостійкість і масштабування – сервіс не повинен працювати повільніше із поступовим збільшенням проектів для обробки або збільшенням кількості залежностей цих проектів.

Відмовостійкість і масштабування може бути вирішення із використанням aws serveless. Цей підхід дозволяє не перейматися про апаратне забезпечення і його розширення, усе це відбувається автоматично.

Інструмент збірки Maven використовується із java мовами, а вони переважно використовується для написання web додатків. Тому усі члени команди повинні мати гарний досвід роботи із HTTP/HTTPS відповідно. Через це вхідною точкою сервісу буде публічне захищене HTTP API, що приймає запити із конкретним тілом і методом.

Зважаючи на те, що постійно працюючий сервер який буде приймати HTTP запити це погане рішення, з причин економії, варто розглянути керовані подіями сервіси. AWS Lambda найпопулярніший з таких сервісів, але вона може працювати максимум 15хв і має лише максимум 10гб пам'яті, чого буде недостатньо у ситуаціях з великою кількістю репозиторіїв.

Єдиним рішенням у цьому випадку стає використання AWS ECS (Elastic Container Service) сервісу, а конкретніше task defifnition, що дозволить вирішити проблему пам'яті і часу, тому що програма матиме змогу працювати стільки, скільки потрібно. Task defifnition може бути запущена на кластері із AWS console мануально або використовуючи AWS SDK. Іншою проблемою є те, що потрібно якимось чином передавати вхідну інформацію у контейнер (наприклад назви проектів для релізу). Для цього доведеться використовувати Lambda, яке буде виступати у ролі посередника.

API Gateway сервіс буде приймати HTTP запити із вхідною інформацією і передавати її до Lambda, що в свою чергу буде записувати це у SQS чергу і запускати ECS task defifnition, яка потім після запуску буде зчитувати потрібні основній програмі данні.

Для того щоб добре зрозуміти у чому саме проблема потрібно мати детальне логування, тобто кожна логічно відокремлена частина коду повинна

писати інформацію про статус протікання процесу, яка в свою чергу буде направлена у AWS CloudWatch, який є сервісом для пошуку помилок у логах тощо. Такий підхід допомогає лише під час аналізу вже існуючої проблеми і змушує спостерігати за повним циклом роботи сервісу, тому щоб не витрачати на це багато часу потрібне сповіщення про успішне завершення або навпаки критичну зупинку. Це буде реалізовано за допомогою AWS SNS (Simple Notification Service) сервісу, використовуючи його можна відправляти повідомлення на електронну адресу і не тільки.

Беручи до уваги все вище описане можна створити діаграму основної архітектури комп'ютерної моделі, яка показана на рисунку 3.1.

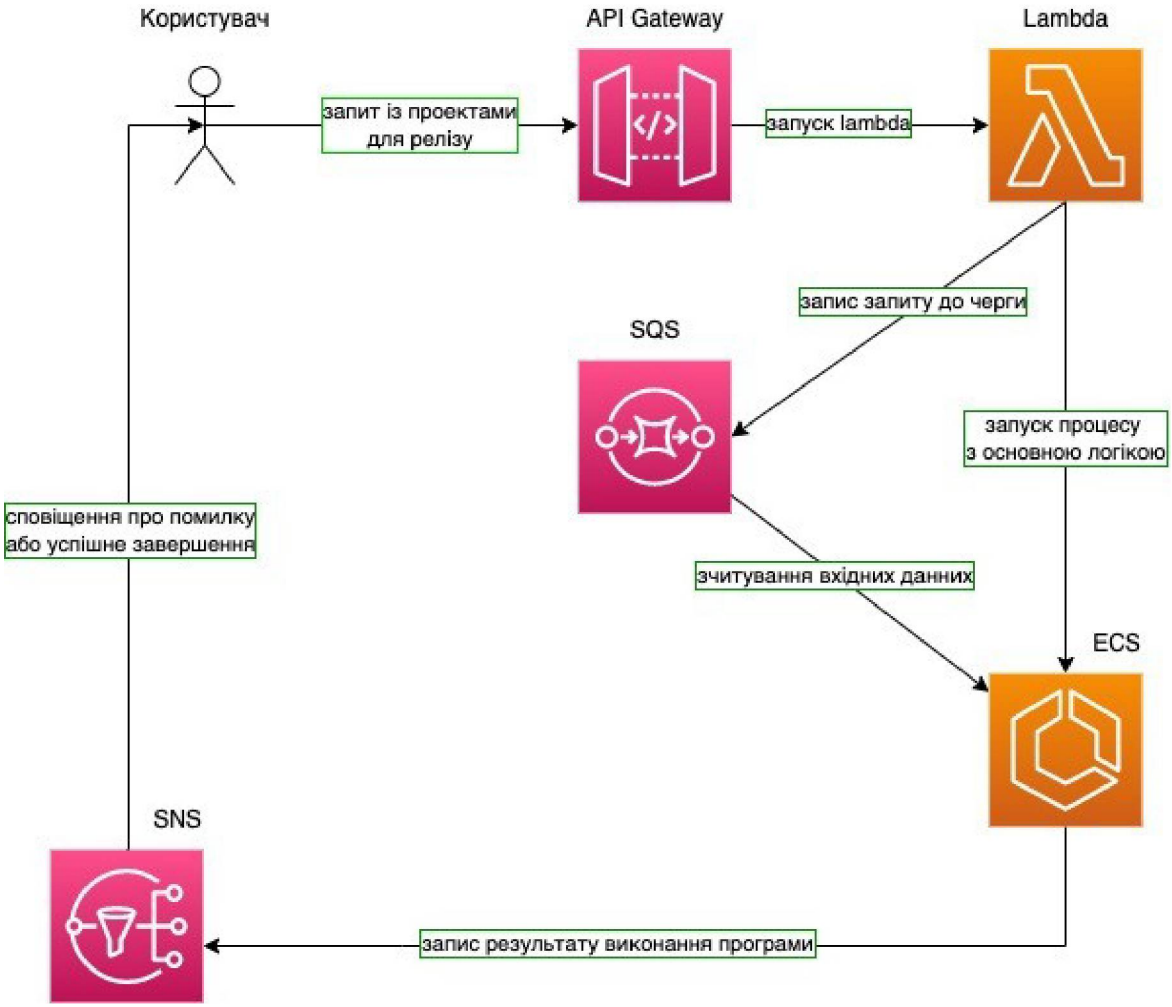


Рисунок. 3.1 – Основна архітектура комп'ютерної моделі організації створення системного програмного забезпечення з використанням технологій DevOps

3.3.2 Аналіз взаємодії користувача з сервісом

Щоб використовувати реалізовану модель, подтрібно розуміти спосіб взаємодії, у данному випадку це HTTP запит на відповідну кінцеву точку. Знаючи цю точку і контракт роботи із нею можна використовуючи різні застосунки запустити сервіс.

Зробити HTTP запит можна багатьма способами, найпопулярніші з них:

- Curl – це інструмент командного рядка для передачі даних між серверами та клієнтами. Він підтримує різні протоколи, включаючи HTTP, FTP, SMTP та багато інших. За допомогою curl ви можете надсилати запити на сервер та отримувати відповіді від нього. Curl часто використовується для тестування API, відлагодження проблем мережі та завантаження файлів з Інтернету. Отже виконавши наступну команду у терміналі запускаємо сервіс, як показано на рисунку 3.2.

```
→  ~ curl --header "Content-Type: application/json" \
  --request POST \
  --data '[
            "projectName1",
            "prjectName2",
           "projectName3"
         ]' \
  https://uwfemalgld.execute-api.us-east-1.amazonaws.com/v1/diplo
ma
"Service has been successfully started"%
→  ~
```

Рисунок 3.2 – Приклад використання curl

- Postman є популярною платформою для співпраці при розробці API. Він надає користувачам зручний інтерфейс для проектування, тестування та документування API. За допомогою Postman можна легко створювати запити, організовувати їх в колекції та ділитися ними зі своїми колегами.

Postman підтримує широкий спектр протоколів API, включаючи REST, SOAP та GraphQL. Він також надає потужний фреймворк для тестування, який дозволяє автоматизувати ваші тести API та запускати їх в хмарі. Ви

можете використовувати Postman для тестування API на різних етапах розробки, від локального тестування до виробничих середовищ. Запустити сервіс можна наступним чином, як показано на рисунку 3.3.

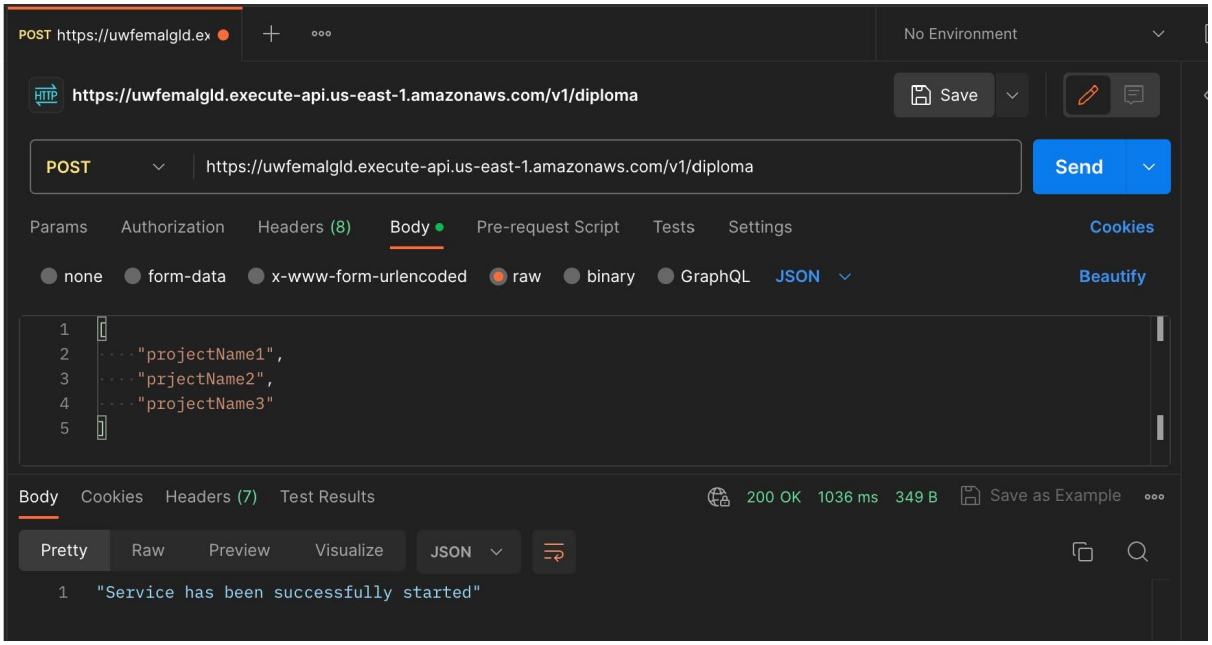


Рисунок 3.3 – Приклад використання Postman

Успішно виконавши запуск , користувач чекає на повідомлення про успішне завершення роботи програми або критичну помилку із подробицями, як показано на рисунку 3.4. і 3.5.

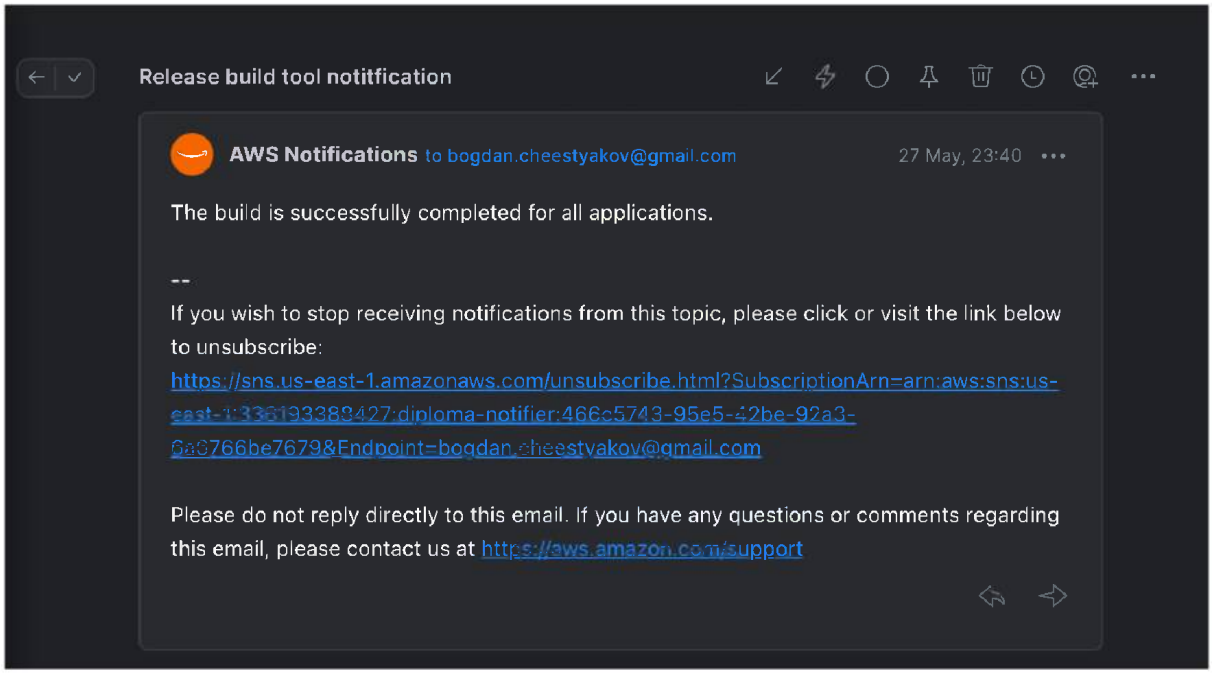


Рисунок 3.4 – Приклад повідомлення про успішне завершення роботи сервісу

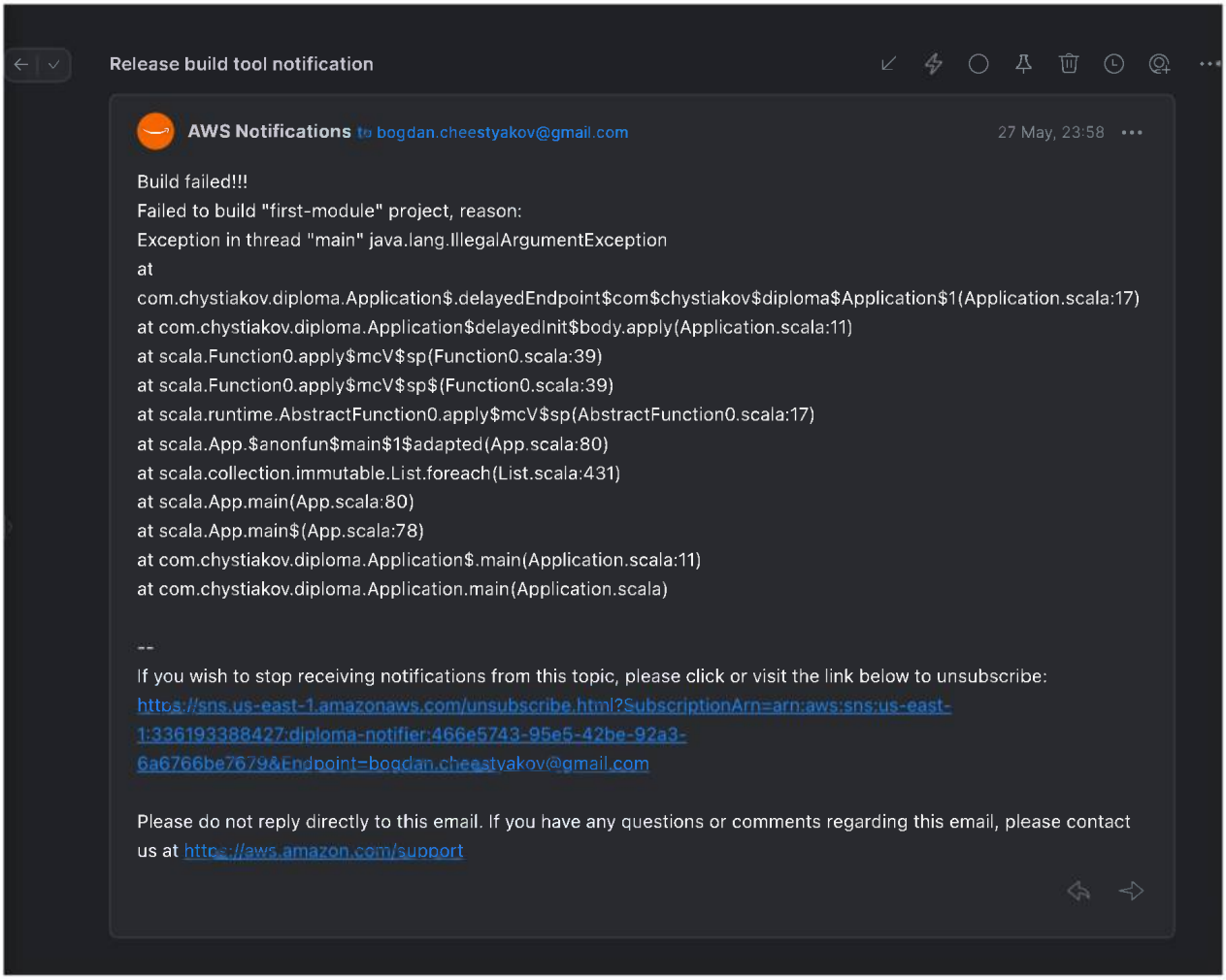


Рисунок 3.5 – Приклад повідомлення про помилку під час роботи сервісу

3.3.3 Інтерфейс та функціонал сервісу для адміністратора

Для того щоб краще розуміти адміністрування сервісу, потрібно розібратись у процесі побудови артефактів із вихідного коду.

У сучасній розробці програмного забезпечення завжди використовується VCS, на додаток до цього завжди повинен бути у наявності і віддалений репозиторій який буде джерелом істини. У даній реалізації віддаленим репозиторієм виступає AWS CodeCommit – це повністю керований сервіс контролю версій, який дозволяє компаніям легко розміщувати безпечні та високомасштабовані приватні репозиторії Git. За допомогою CodeCommit розробники можуть зберігати та керувати своїм кодом, співпрацювати з іншими та автоматизувати робочі процеси розробки програмного забезпечення.

CodeCommit зберігає лише вихідний код, який у свою чергу повинен бути оброблений певним чином для того, щоб з рештою мати змогу запускати додатки. Це і називається процесом збору, головною метою якого і є використання Maven.

Збірка коду локально – погана ідея, адже у кожного в команді абсолютно різне програмне і апаратне забезпечення, більше того, збірка може залежати від певної інфраструктури (модульні тести), наприклад баз даних тощо, до яких зазвичай немає доступу із локального середовища. Щоб уникнути проблем використовується AWS CodeBuild – це повністю керований сервіс збірки, який компілює вихідний код, запускає тести та генерує програмні пакети, які готові до розгортання. За допомогою CodeBuild, ми можемо легко збирати та тестувати свій код, не керуючи серверами або інфраструктурою збірки.

Результати роботи CodeBuild потрібно десь зберігати. Для цього а добре підходить AWS CodeArtifact – це повністю керований сервіс зберігання артефактів, який дозволяє організаціям легко та безпечно зберігати та ділитися програмними пакетами. За допомогою CodeArtifact, можна зберігати та керувати своїми пакетами, контролювати доступ до них та легко отримувати їх, коли це необхідно.

Пайплайн з основною логікою показано на рисунку 3.6.

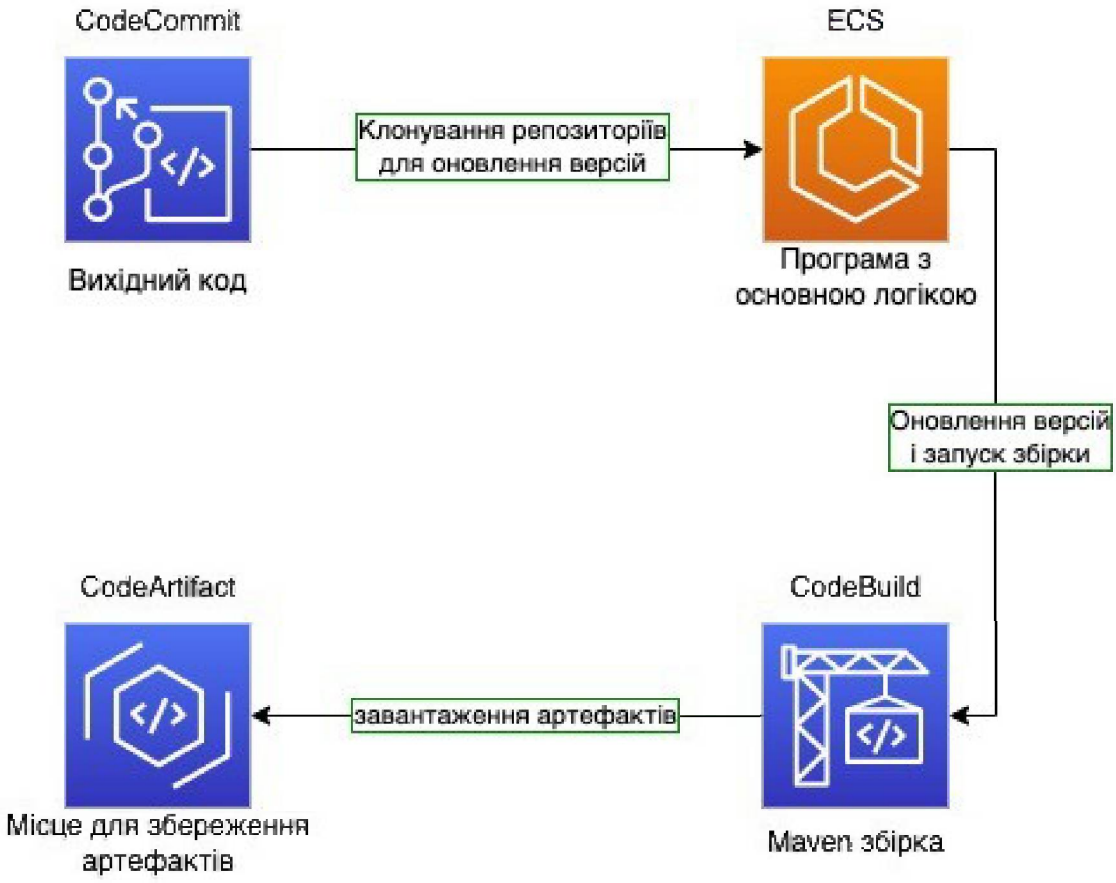


Рисунок 3.6 – Взаємодія програми з основною логікою із cloud CI/CD інструментами

Окремої уваги заслуговує моніторинг статусу і результатів виконання процесу. AWS надає можливість спостерігати за цим на кожному етапі.

Наприклад надіславши HTTP запит на кінцеву точку, варто подивитися логи у AWS CloudWatch Logs – це керований сервіс, який дозволяє легко централізувати та моніторити журнали з ресурсів та додатків AWS. За допомогою CloudWatch Logs, можна збирати, переглядати та аналізувати журнали з різних джерел в режимі реального часу, використовуючи ці дані для пошуку проблем, моніторингу продуктивності додатків та покращення безпеки AWS-середовища.

Спочатку там буде тільки інформація про запуск сервісу, переважно це логи Lambda, приклади можна побачити на рисунку 3.7.

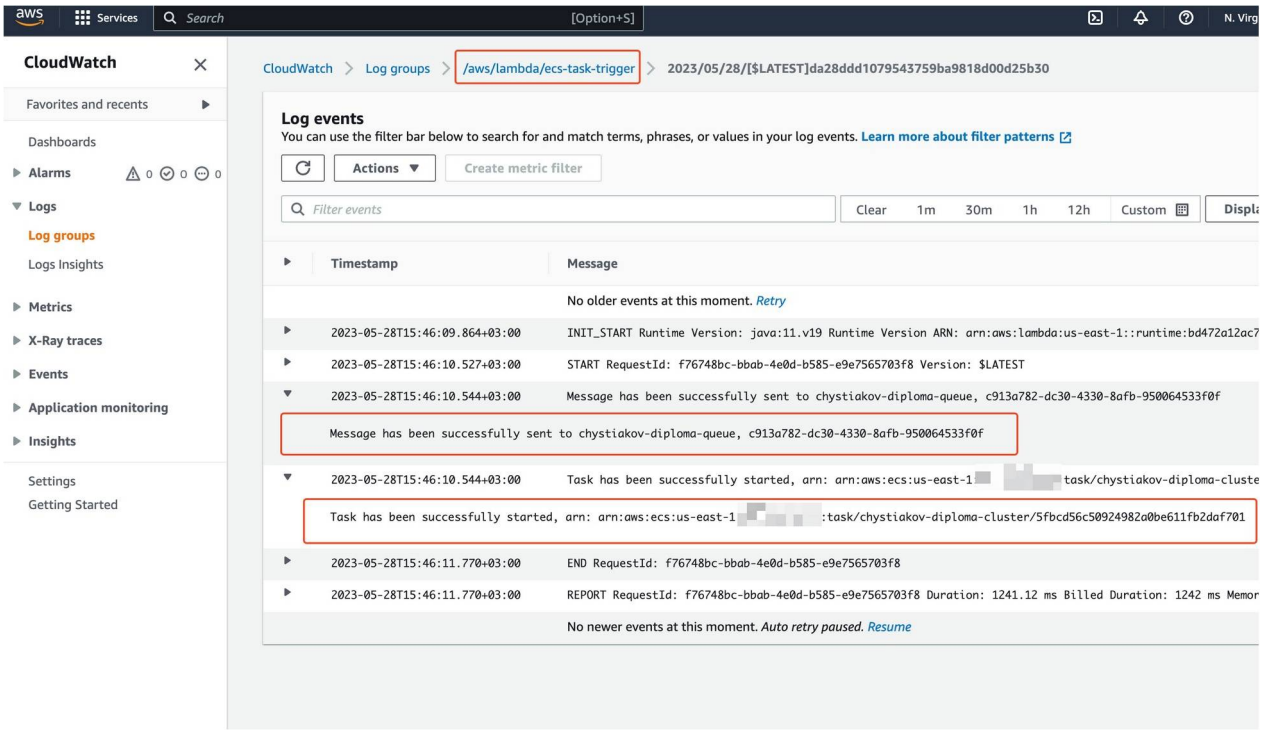


Рисунок 3.7 – Логи Lambda

Після того, як ECS task defifnition почне роботу, можна побачити статус його роботи на рисунку 3.8.

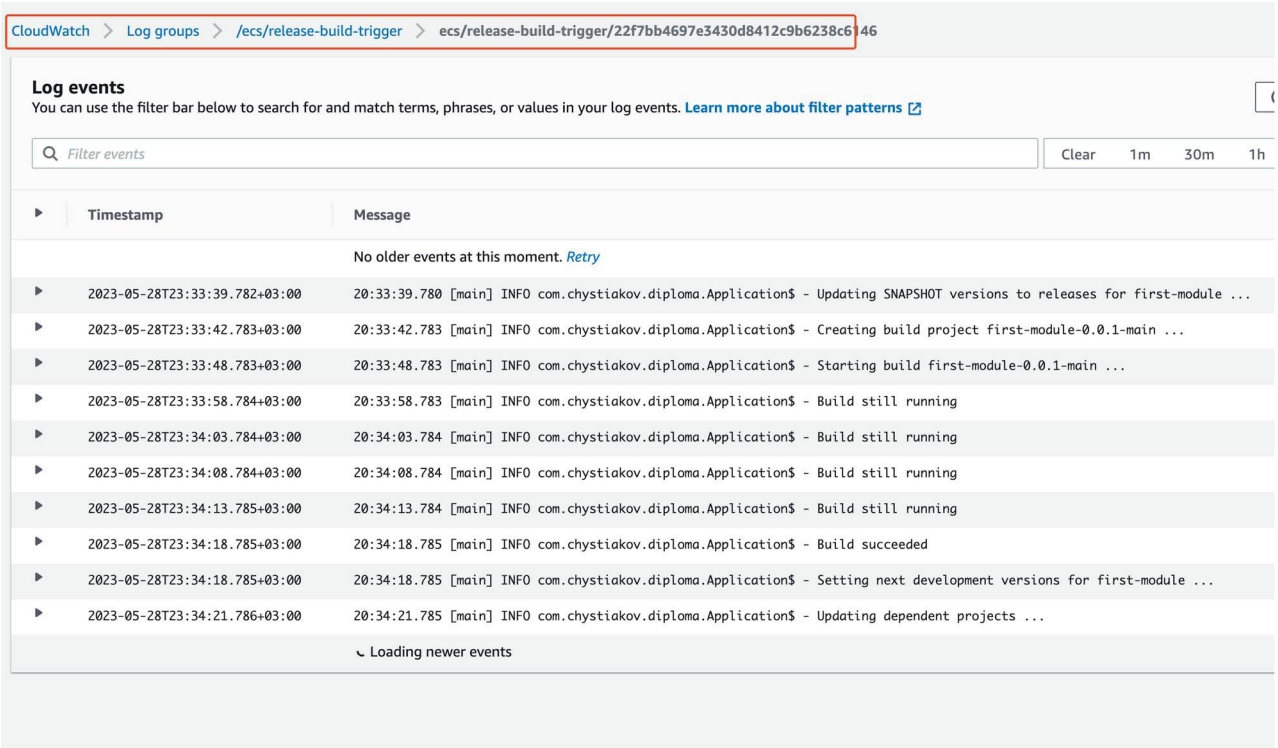


Рисунок 3.8 – Логи ECS task defifnition (програми з основною логікою)

Також є можливість спостерігати за процесом збору самого артефакту і переглядати статус цього процесу у CodeBuild вкладці, як показано на рисунку 3.9.

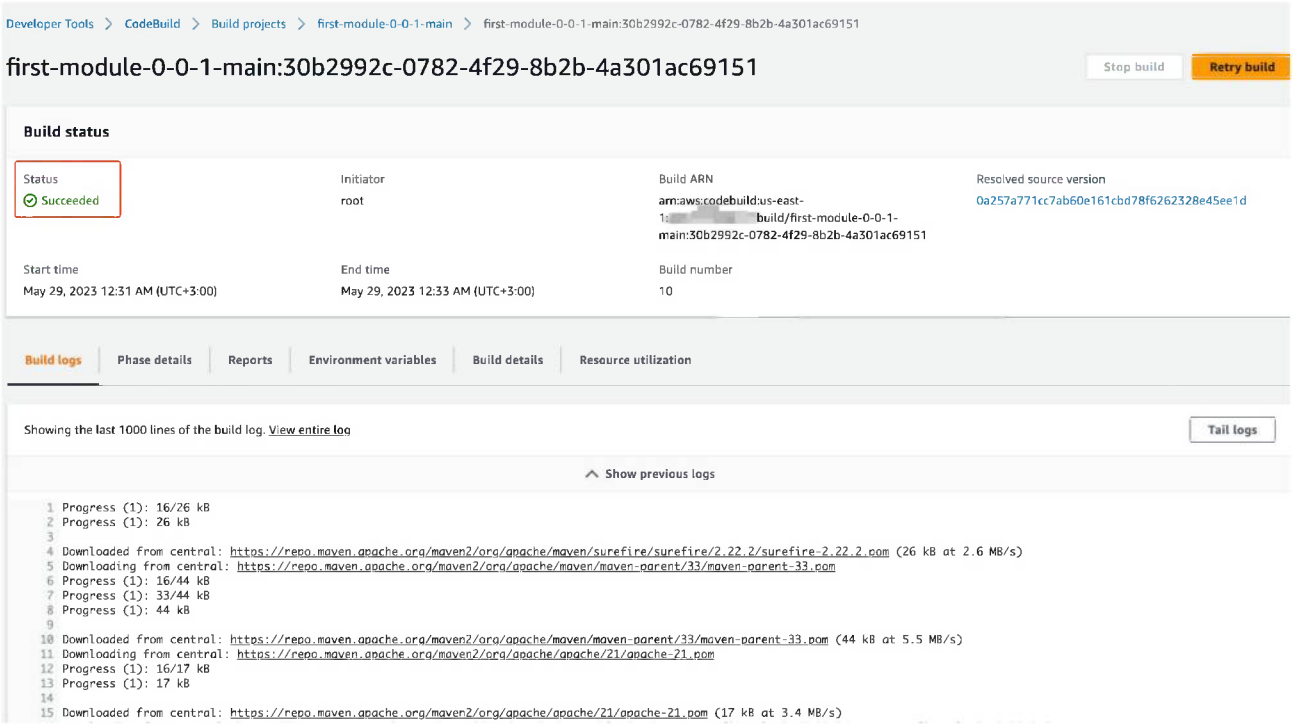


Рисунок 3.9 – Протікання процесу збору у CodeBuild

Фінальним результатом роботи сервісу повинен бути готовий до запуску артефакт, збережений у CodeArtifact, який можна побачити на рисунку 3.10.

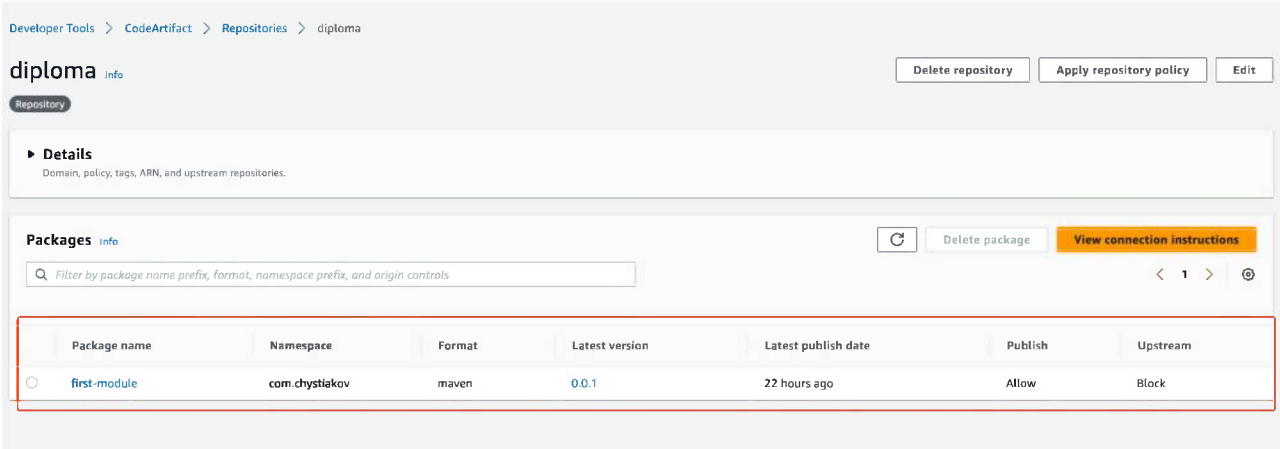


Рис 3.10 – Приклад зібраного артефакту збереженого у CodeArtifact

Висновки за розділом 3

У даному розділі було розглянуто Scala, Maven та AWS SDK як інструменти для розробки високоякісних додатків. Було описано використання хмарних сервісів AWS Serverless для розробки та виконання додатків. Було проведено розробку та аналіз програмного сервісу, включаючи побудову основної архітектури сервісу, аналіз взаємодії користувача з сервісом та розробку інтерфейсу та функціоналу сервісу для адміністратора.

ВИСНОВКИ

Розробка системного програмного забезпечення з використанням технологій DevOps стала все більш популярною в останні роки завдяки її здатності оптимізувати процес розробки програмного забезпечення. Цей підхід наголошує на співпраці між командами розробки та операцій з метою швидшої та ефективнішої доставки програмного забезпечення. Однак реалізація DevOps може бути складною і вимагати значних змін у існуючих процесах та робочих потоках.

Комп'ютерна модель, запропонована у цій дипломній роботі, надає структурований підхід до організації та управління процесом розробки системного програмного забезпечення з використанням принципів DevOps. Модель призначена для допомоги командам працювати більш ефективно разом з фокусом на постійній інтеграції, постійній доставці та постійному тестуванні. Вона надає фреймворк для управління проектами розробки програмного забезпечення від початку до кінця з особливим наголосом на забезпеченні доставки програмного забезпечення вчасно та відповідно до вимог якості.

Однією з ключових переваг запропонованої моделі є її здатність сприяти співпраці між командами розробки та операцій. Усуваючи традиційні бар'єри між командами та сприяючи культурі спільної відповідальності, модель може допомогти покращити комунікацію та зменшити ризик помилок та затримок. Крім того, модель може допомогти зменшити витрати, підвищити продуктивність та покращити загальну якість програмного забезпечення.

Реалізація запропонованої моделі потребує значних витрат часу та ресурсів, але потенційні переваги значні. Приймаючи принципи DevOps та використовуючи структурований підхід до розробки програмного забезпечення, організації можуть покращити свою здатність доставляти високоякісне програмне забезпечення швидше та ефективніше. Таким чином, запропонована модель є цінним внеском у галузь програмної інженерії та має потенціал суттєво

вплинути на спосіб розробки та доставки системного програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kim, G., Humble, J., Debois, P., & Willis, J. (2016). The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations. Portland, OR: IT Revolution Press, 2016. 480 p.

2. Humble, J., & Farley, D. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley Professional, 2010. 512 p.

3. Davis, J., & Daniels, K. Effective DevOps: Building a Culture of Collaboration, Affinity, and Tooling at Scale. O'Reilly Media, 2016. 410 p.

4. Rankin, K. DevOps Troubleshooting: Linux Server Best Practices. Addison-Wesley Professional, 2012. 205 p.

5. Kim, G., Behr, K., & Spafford, G. The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win. IT Revolution Press, 2013. 345 p.

6. Bass, L., Weber, I., & Zhu, L. DevOps: A Software Architect's Perspective. Addison-Wesley Professional, 2015. 352 p.

7. Sharma, S. The DevOps Adoption Playbook: A Guide to Adopting DevOps in a Multi-Speed IT Enterprise. Apress, 2017. 416 p.

8. Mueller, J. P. AWS for Developers For Dummies. John Wiley & Sons, 2016. 384 p.

9. Vyas, U. Mastering AWS Development. Packt Publishing, 2015. 416 p.

10. Wittig, A., & Wittig, M. Amazon Web Services in Action. Manning Publications, 2015. 424 p.

11. Chan, L., & Braun, M. AWS Administration Cookbook: Core and Advanced Recipes for Amazon Web Services. Packt Publishing, 2017. 394 p.

12. Sarkar, A. Learning AWS: Design, build, and deploy responsive applications using AWS Cloud components. Packt Publishing, 2018. 412 p.

13. Alexander, A. Scala Cookbook. O'Reilly Media, 2013. 722 p.

14. Bauer, T., & King, J. Maven: The Definitive Guide. O'Reilly Media, 2008. 470 p.

15. AWS Documentation [Електронний ресурс]. – режим доступу: URL: https://docs.aws.amazon.com/ (дата звернення – 04.06.2023)

16. Top 14 CI CD Tools for your DevOps project [Електронний ресурс]. – режим доступу: URL: https://www.browserstack.com/guide/top-ci-cd-tools?utm_source=google&utm_medium=cpc&utm_platform=paidads&utm_content=645400465181&utm_campaign=Search-DSA-NB-T2Geo-Exp&utm_campaigncode=Guide-Page+1012840&utm_term=+&gclid=CjwKCAjwvdajBhBEEiwAeMh1U_rjHeUnK8zuQes5dHLE9gcC5JjKI-eLqZ9l0WDhqGWS-9KXHV2qDBoCmFUQAvD_BwE (дата звернення – 01.06.2023)

17. The 50 Best CI/CD Tools All DevOps Teams Should Know In 2023 [Електронний ресурс]. – режим доступу: URL: https://www.cloudzero.com/blog/cicd-tools (дата звернення – 30.05.2023)

18. CI/CD Tools Comparison: Jenkins, TeamCity, Bamboo, Travis CI, and More [Електронний ресурс]. – режим доступу: URL: https://www.altexsoft.com/blog/engineering/cicd-tools-comparison/ (дата звернення – 28.05.2023)

ДОДАТКИ

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Факультет комп'ютерних наук
Кафедра теоретичної та прикладної системотехніки
Рівень вищої освіти (освітньо-кваліфікаційний рівень) Бакалавр
Галузь знань: 12 – Інформаційні технології
Спеціальність: 123 – Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри теоретичної
та прикладної системотехніки
_____ д.т.н., проф. Шматков С. І.

«17»листопада 2022 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Чистякова Богдана Юрійовича
(прізвище, ім'я, по батькові студента)

1. Тема роботи «Комп'ютерна модель організації створення системного програмного забезпечення з використанням технологій DevOps»
керівник роботи Мороз Ольга Юріївна, старший викладач кафедри теоретичної та прикладної системотехніки,
затверджені наказом по університету від «23» травня 2023року № 4101-5/895
2. Строк подання студентом роботи 26 травня 2023
3. Перелік питань, які потрібно розробити
   1) Аналіз предметної області задля вивчення ключових моментів створення системного програмного забезпечення з використанням технологій DevOps.
   2) Аналіз сучасного робочого процесу адміністраторів DevOps під час роботи з Git і віддаленими репозиторіями.
   3) Огляд основних підходів, методів та існуючих інструментів неперервної інтеграції коду та його розгортання та тестування.
   4) Порівняльний аналіз провідних інструментів побудови CI/CD.
   5) Побудова комп'ютерної моделі на основі проведеного аналізу із застосуванням сучасних хмарних CI/CD інструментів.
   6) Тестування програмного засобу на основі обраного CI/CD інструменту
   7) Розробка рекомендації по впровадженню технології, що використовуються в роботі.

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1.	Аналіз і підбір літератури та створення літературної бази з DevOps методології.	19.10.2022–16.01.2023
2.	Аналіз сучасних методології розробки системного забезпечення workflow із системами контролю версій	19.10.2022–16.01.2023
3.	Аналіз предметної області задля вивчення CI/CD засобів, розгляд найпопулярніших з них і виділення недоліків	19.10.2022–16.01.2023
4.	Огляд та порівняння аналогів сучасних хмарних CI/CD інструментів. Визначення критеріїв аналізу. Огляд визначених інструментів	16.01.2023 – 20.02.2023
5.	Побудова комп'ютерної моделі на основі проведеного аналізу із застосуванням сучасних хмарних CI/CD інструментів.	20.02.2023–24.04.2023
6.	Розробка технічного завдання на розроблену модель.	20.02.2023–24.04.2023
7.	Побудова архітектури CI та CT та налаштування.	20.02.2023–24.04.2023
8.	Побудова архітектури CD та поєднання з налаштованим CI/CT	20.02.2023–24.04.2023
9.	Оформлення пояснювальної записки	24.04.2023 – 25.05.2023
10.	Підготовка доповіді на тему кваліфікаційної роботи на науково-технічну конференцію	24.04.2023 – 25.05.2023
11.	Оформлення звіту за результатами переддипломної практики	24.04.2023 – 25.05.2023
12.	Перед захист кваліфікаційної роботи	Травень 2023
13.	Представлення кваліфікаційної роботи керівнику та рецензенту	26.05.2023

5. Дата видачі завдання «19» жовтня 2022року

Студент Чистяков Б. Ю. 
ініціали, прізвище підпис

Керівник роботи Мороз О. Ю. 
ініціали, прізвище підпис

Додаток Б

Затверджую

_____

«__» _____ 2023 р.

Технічне завдання

на розробку програмного виробу «Комп'ютерна модель організації створення системного програмного забезпечення з використанням технологій DevOps»

1.	Введення	1.1. Назва: Комп'ютерна модель організації створення системного програмного забезпечення з використанням технологій DevOps 1.2. Галузь застосування: Інформаційні технології
2.	Підстава для розробки	2.1. Навчальний план за спеціальністю 123 – Комп'ютерна інженерія 2.2. Завдання на кваліфікаційну роботу бакалавра № ___4101-5/895_____ від «_23_» _____05____2023 (представити як Додаток А до пояснювальної записки до кваліфікаційної роботи).
3.	Призначення розробки	3.1. Мета розробки: створення сервісу, який допомагає збирати стабільні версії артефактів. 3.2. Призначення розробки надає можливість після відправки запиту на кінцеву точку спостерігати за зборкою проекту. 3.3. Вихідні дані розробки: зібраний артефакт; вхідні дані: збирач проектів з первинним кодом.
4.	Технічні вимоги до програмного виробу	4.1. Вимоги до функціональних характеристик: підтримка збирача проєктів maven, рекурсивне збирання всіх залежностей і батьків оголошених у pom.xml. 4.2. Вимоги до надійності: забезпечення безперебійної роботи програмного виробу при будь-яких вимогах користувача в рамках призначення виробу . 4.3.Вимоги до умов експлуатації: немає 4.4. Вимоги до складу і параметрів технічних засобів: для виконання програми повинен підходити ПК із будь-якою операційною системою сімейства Windows, Linux/Unix, Mac OS X, OS/2, Amiga. Крім того, для роботи потрібний інтерпретатор мови програмування. 4.5. Вимоги до інформаційної та програмної сумісності: підтримка ОС Linux або Windows 10,

		підтримка мови програмування, підтримка різних платформ. 4.6. Вимоги до маркування та упаковки: вимоги до маркування та упакування не представляються. 4.7. Вимоги до транспортування і зберігання: вимоги до транспортування та зберігання не представляються. 4.8. Спеціальні вимоги: спеціальні вимоги до програмного виробу не пред'являються.	
5.	Вимоги до програмної документації	Програмною документацією до виробу «Метод аналізу інформативності змінних стану при діагностиці систем з використанням інформаційних критеріїв» вважати: 1) Справжнє Технічне завдання на розробку виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи). 2) Методику розрахунку інформативності змінних стану (у вигляді глав 3.2 та 3.3 пояснювальної записки до кваліфікаційної роботи). 3) Опис виробу (представити в розділі 3 пояснювальної записки до кваліфікаційної роботи)	
6.	Вимоги до техніко-економічних показників	Програмною документацією до виробу «Комп'ютерна модель організації створення системного програмного забезпечення з використанням технологій DevOps» вважати: 1) Справжнє Технічне завдання на розробку виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи). 2) Опис програмного виробу (представити в Розділі 3 пояснювальної записки до кваліфікаційної роботи). 3) Джерела базової інформації.	
7.	Стадії і етапи розробки	Дата	Назва етапу
		від 19 жовтня 2022 до 16 січня 2023	Аналіз і підбір літератури та створення літературної бази з DevOps методології.
		від 19 жовтня 2022 до 16 січня 2023	Аналіз сучасних методології розробки системного забезпечення workflow із системами контролю версій
		від 19 жовтня 2022 до 16 січня 2023	Аналіз предметної області задля вивчення CI/CD засобів,

			розгляд найпопулярніших з них і виділення недоліків
		від 16 січня 2023 до 20 лютого 2023	Огляд та порівняння аналогів сучасних хмарних CI/CD інструментів. Визначення критеріїв аналізу. Огляд визначених інструментів
		від 20 лютого 2023 до 24 квітня 2023	Побудова комп'ютерної моделі на основі проведеного аналізу із застосуванням сучасних хмарних CI/CD інструментів.
		від 20 лютого 2023 до 24 квітня 2023	Розробка технічного завдання на розроблену модель.
		від 20 лютого 2023 до 24 квітня 2023	Побудова архітектури CI та CT та налаштування.
		від 20 лютого 2023 до 24 квітня 2023	Побудова архітектури CD та поєднання з налаштованим CI/CT
		від 24 квітня 2023 до 25 травня 2023	Оформлення пояснювальної записки
		від 24 квітня 2023 до 25 травня 2023	Підготовка доповіді на тему кваліфікаційної роботи на науково-технічну конференцію
		від 24 квітня 2023 до 25 травня 2023	Оформлення звіту за результатами переддипломної практики
		травень 2023	Перед захист кваліфікаційної роботи
		26 травня 2023	Представлення

			рецензенту
8.	Порядок контролю і приймання програмного продукту (моделі)	1. Перевірку ходу розробки програми виконувати раз в 3 тижні. 2. Захист розробленої моделі провести на засіданні Атестаційної комісії. 3. Пояснювальну записку подати на паперових носіях в 1 примірнику і в електронному вигляді в 1 примірнику на CD-R компакт-диску.	

Виконавець
студент групи КІ- 41
Чистяков Б. Ю.
_____

Замовник
старший викладач
Мороз О. Ю.
_____

Програма і методика випробувань програмного виробу

«Комп'ютерна модель організації створення системного програмного забезпечення з використанням технологій DevOps»

1 **Об'єкт випробувань**

2 Назва програмного виробу : «Комп'ютерна модель організації створення системного програмного забезпечення з використанням технологій DevOps»

3 Галузь застосування : Інформаційні технології

4 Перераховані відомості запозичуються з відповідних розділів Технічного завдання.

2. Мета випробувань

Перевірка відповідності функціональності програмної реалізації системи заявленим функціональним можливостям в технічному завданні (Додаток Б до пояснювальної записки до кваліфікаційної роботи).

3. Загальні положення

1. Підстави для проведення випробувань

Підставою для проведення випробувань є наказ про призначення атестаційної комісії.

2. Місце і тривалість випробувань

Приймальні (приймально-здавальні) випробування проводяться на базі комп'ютерного класу кафедри в період роботи атестаційної комісії.

3. Обсяг випробувань

Приймальні випробування програмного вироби проводяться в обсязі відповідному цієї програми і методики випробувань.

4. Організації, які беруть участь у випробуваннях

Приймальні випробування проводяться атестаційною комісією напередодні засідання (або в процесі засідання) за участю Замовника, Виконавця та інших осіб, присутніх на засіданні.

4. Вимоги до програми або програмного виробу

Модель повинна задовольняти наступним вимогам:

4.1. Вимоги до функціональних характеристик: підтримка збирача проектів maven, рекурсивне збирання всіх залежностей і батьків оголошених у pom.xml.
4.2. Вимоги до надійності: забезпечення безперебійної роботи програмного виробу при будь-яких вимогах користувача в рамках призначення виробу .
4.3.Вимоги до умов експлуатації: немає
4.4. Вимоги до складу і параметрів технічних засобів: для виконання програми повинен підходити ПК із будь-якою операційною системою сімейства Windows, Linux/Unix, Mac OS X, OS/2, Amiga. Крім того, для роботи потрібний інтерпретатор мови програмування.
4.5. Вимоги до інформаційної та програмної сумісності: підтримка ОС Linux або Windows 10, підтримка мови програмування, підтримка різних платформ.
4.6. Вимоги до маркування та упаковки: вимоги до маркування та упакування не представляються.
4.7. Вимоги до транспортування і зберігання: вимоги до транспортування та зберігання не представляються.
4.8. Спеціальні вимоги: спеціальні вимоги до програмного виробу не пред'являються.

5. Вимоги до програмної документації

Документацією до виробу «Комп'ютерна модель організації створення системного програмного забезпечення з використанням технологій DevOps» вважати:

1) Документація по мові програмування та додаткові мануали.

2) Програму і методику випробувань розробленої програми (представити як Додаток В до пояснювальної записки до кваліфікаційної роботи).

3) Опис програмного виробу (представити в Розділі 3 пояснювальної записки до кваліфікаційної роботи).

4) Джерела базової інформації.

6. Засоби і порядок випробувань

6.1 Засоби випробувань

Засоби випробувань представлено на ПК на яких встановлено наступні програмні засоби: інтерпретатор мови програмування.

6.2 Порядок проведення випробувань

Як правило, випробування проводяться в два етапи:

- ознайомчий (1-й етап);
- власне випробування програмного виробу (2-й етап).

Перелік перевірок, що проводяться на 1 етапі випробувань, включає в себе:

1) Перевірку комплектності складу програмної документації здійснюється за критерієм наявності зазначеної в ТЗ документації.

2) Перевірку якості програмної документації. Перевірку здійснювати за критерієм відповідності вимогам ГОСТ 19.301-79 ЕСПД. «Програма і методика випробувань».

Перелік перевірок, що проводяться на 2 етапі випробувань, включає в себе:

1) Перевірку відповідності технічних характеристик програми вимогам технічного завдання.

2) Перевірку ступеня виконання функціональних вимог до програми.

3) Методику проведення перевірок:

a) Запустити програмне забезпечення.

b) Порядок проведення випробувань:

- Зробити налаштування.
- Перевірити чи працює програма.
- Перевірити чи формується звіт.

4) Якщо перевірки на першому та другому етапах виконано успішно, то виріб вважається таким, що пройшов випробування.

Для проведення випробувань пропонується тест 1, тест 2 та тест 3.

Тест 1

1. Перевірка виконання програми;
2. Основний тригер сервісу;
3. Отримання відповіді серверу про успішне створення об'єкту.

```
AWSTemplateFormatVersion: '2010-09-09'
Description: CloudFormation template for EventBridge rule 'code-commit-event'
Resources:
  EventRule0:
    Type: AWS::Events::Rule
    Properties:
      EventBusName: default
      EventPattern:
        source:
          - aws.codecommit
        detail-type:
          - CodeCommit Repository State Change
        resources:
          - arn:aws:codecommit:us-east-1:336193388427:first-module
        detail:
          event:
            - referenceCreated
          referenceType:
            - branch
          referenceName:
            - prefix: release
      Name: code-commit-event
      State: ENABLEDAWSTemplateFormatVersion/ '2010-09-09'
      Targets:
        - Id: Id4254c832-496c-4eb5-862d-00f039b6d4cd
          Arn: arn:aws:lambda:us-east-1:336193388427:function:release-repo-handler
```

Рис. В.1 Тест 1

Тест 2

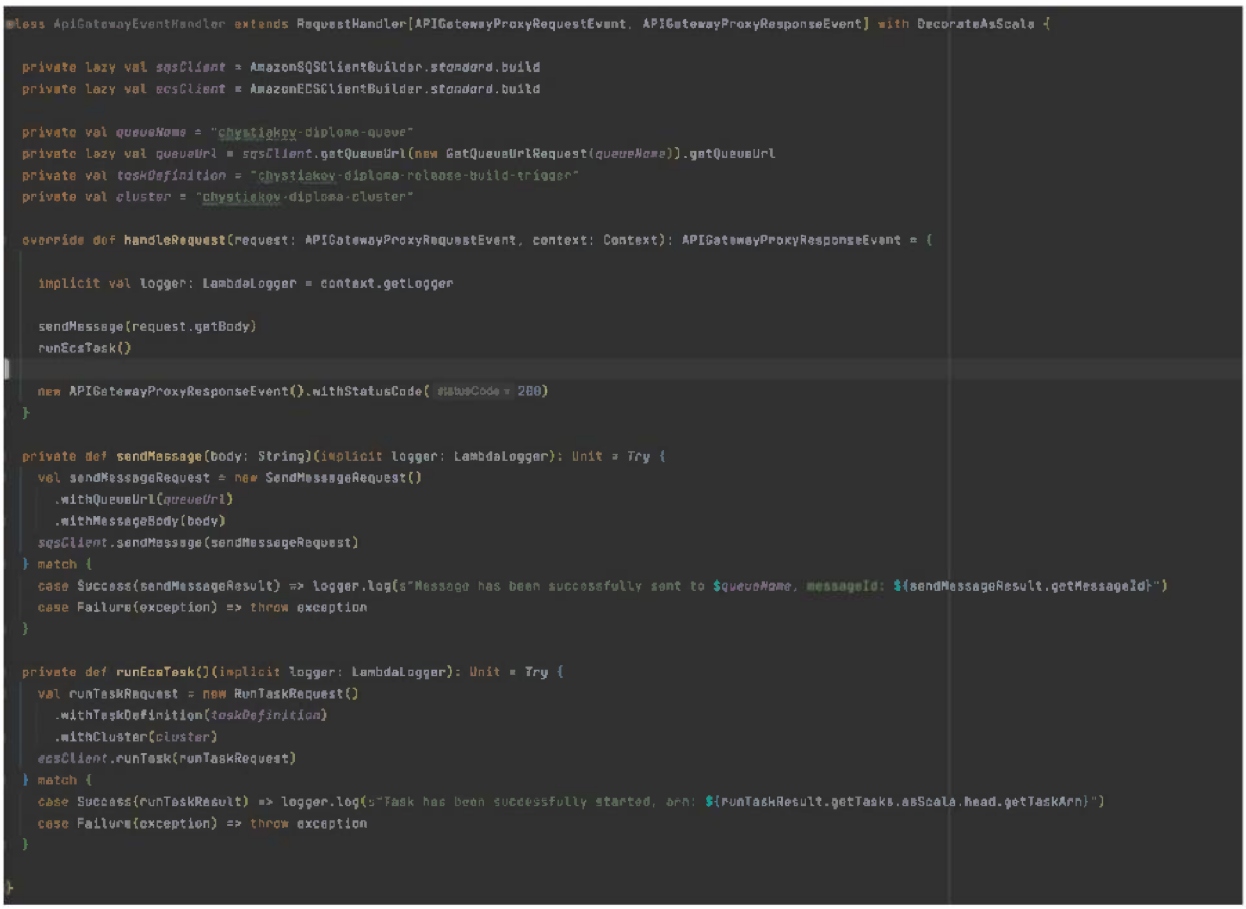
1. Перевірка виконання програми
2. Конфігурація обчислювального сервісу, який дозволяє запускати код без резервування;
3. Отримання результатів.

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: An AWS Serverless Specification template describing your function.
Resources:
  releaserepohandler:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: 'com.chystiakov.diploma.ReleaseRepoHandler::handleRequest'
      Runtime: java11
      CodeUri: .
      Description: ''
      MemorySize: 512
      Timeout: 15
      Role: >-
        arn:aws:iam::336193388427:role/service-role/release-repo-handler-role-0n6fcat7
      Layers:
        - 'arn:aws:lambda:us-east-1:553035198032:layer:git-lambda2:8'
      RuntimeManagementConfig:
        UpdateRuntimeOn: Auto
```

Рис. В.2 Тест 2

Тест 3

4. Перевірка виконання програми
5. Основний код, який отримує повідомлення про зміни репозиторія;
6. Отримання результату.



```
class ApiGatewayEventHandler extends RequestHandler[APIGatewayProxyRequestEvent, APIGatewayProxyResponseEvent] with DecorateAsScala {

  private lazy val sqsClient = AmazonSQSClientBuilder.standard.build
  private lazy val ecsClient = AmazonECSClientBuilder.standard.build

  private val queueName = "chystiakov-diploma-queue"
  private lazy val queueUrl = sqsClient.getQueueUrl(new GetQueueUrlRequest(queueName)).getQueueUrl
  private val taskDefinition = "chystiakov-diploma-release-build-trigger"
  private val cluster = "chystiakov-diploma-cluster"

  override def handleRequest(request: APIGatewayProxyRequestEvent, context: Context): APIGatewayProxyResponseEvent = {

    implicit val logger: LambdaLogger = context.getLogger

    sendMessage(request.getBody)
    runEcsTask()

    new APIGatewayProxyResponseEvent().withStatusCode( statusCode = 200)
  }

  private def sendMessage(body: String)(implicit logger: LambdaLogger): Unit = Try {
    val sendMessageRequest = new SendMessageRequest()
      .withQueueUrl(queueUrl)
      .withMessageBody(body)
    sqsClient.sendMessage(sendMessageRequest)
  } match {
    case Success(sendMessageResult) => logger.log(s"Message has been successfully sent to $queueName, messageId: ${sendMessageResult.getMessageId}")
    case Failure(exception) => throw exception
  }

  private def runEcsTask()(implicit logger: LambdaLogger): Unit = Try {
    val runTaskRequest = new RunTaskRequest()
      .withTaskDefinition(taskDefinition)
      .withCluster(cluster)
    ecsClient.runTask(runTaskRequest)
  } match {
    case Success(runTaskResult) => logger.log(s"Task has been successfully started, arn: ${runTaskResult.getTasks.asScala.head.getTaskArn}")
    case Failure(exception) => throw exception
  }

}
```

Рис. В.3 Тест 3

Тест вважається пройденим, якщо відбуваються вказані операції і їх відображення у програмному продукті.

Висновки: тест 1 успішно пройшов випробування, тест 2 успішно пройшов випробування і тест 3 успішно пройшов випробування. Випробування пройшло успішно.

Виконавець: студент групи КІ-41, Чистяков Б. Ю.

Додаток Г

Лістинг коду

«Комп'ютерна модель організації створення системного програмного забезпечення з використанням технологій DevOps»

Клас із основною логікою

```
package com.chystiakov.diploma

import com.chystiakov.diploma.CodeBuildHelper._
import com.chystiakov.diploma.CodeCommitHelper._
import com.chystiakov.diploma.MavenModelExtensions._
import com.chystiakov.diploma.StringExtensions._
import com.typesafe.scalalogging.LazyLogging
import org.apache.maven.model.Model
import org.apache.maven.model.io.xpp3.{MavenXpp3Reader, MavenXpp3Writer}
import resource._

import java.io.{File, FileInputStream, FileWriter}
import java.nio.file.{Files, Paths}
import scala.collection.convert.DecorateAsScala
import scala.collection.immutable


class MavenManagerV2(artifactId: String) extends DecorateAsScala with LazyLogging {

  private val pomReader = new MavenXpp3Reader
  private val pomWriter = new MavenXpp3Writer

  private val gitManager = new GitManager(artifactId)

  private lazy val pomModels = extractMavenModels(gitManager.path.toAbsolutePath.toString)
  private implicit lazy val groupId: String = pomModels.flatMap(m => Option(m.getGroupId)).head

  def updateVersions(): Unit = {
    getAllDependenciesForRelease match {
      case releaseDependencies => releaseDependencies
        .map { case (_, artId) => new MavenManagerV2(artId) }
        .filter(!_.gitManager.hasBeenClonedBefore)
        .foreach(_.updateVersions())
      case _ =>
    }
    val releaseBranch: String = doRelease()
    createAndRunReleaseBuild(gitManager.repoName, rootVersion, releaseBranch, gitManager.cloneUrl)
    doNxtDevelopment()
    updateDependentProjects()
  }
```

```
  private def doRelease(): String = {
    val releaseBranch = gitManager.releaseBranchCheckout(rootVersion.release)
    val commitMessage = "Updating SNAPSHOT versions to releases"
    logger.info(s"$commitMessage for ${gitManager.repoName} ...")
    updateModelsAndApplyChanges(pomModels.map(_.clone), MavenVersionsManager.updateToReleaseVersions(_), commitMessage)
    releaseBranch
  }

  private def doNxtDevelopment(): Unit = {
    gitManager.mainBranchCheckout()
    val commitMessage = "Setting next development versions"
    logger.info(s"$commitMessage for ${gitManager.repoName} ...")
    updateModelsAndApplyChanges(pomModels, MavenVersionsManager.updateToNxtDevVersions, commitMessage)
  }

  private def updateDependentProjects(): Unit = {
    logger.info("Updating dependent projects ...")
    val artifacts = pomModels.map(model => model.getArtifactId)
    val depRepos = artifacts.flatMap(dependentRepos)
    val commitMessage = "Setting last development versions for dependencies"
    val dependentMavenProjects = depRepos.map(new MavenManagerV2(_)).filter(!_.gitManager.hasBeenClonedBefore)
    dependentMavenProjects.foreach(mp => updateModelsAndApplyChanges(
      mp.pomModels,
      MavenVersionsManager.updateSpecificDependencies(artifacts, rootVersion, _),
      commitMessage
    ))
  }

  private def updateModelsAndApplyChanges(pomModels: List[Model], update: Model => Boolean, commitMessage: String): Unit = {
    val updatedPoms = pomModels.collect { case pom if update(pom) => pom }
    if (updatedPoms.nonEmpty) {
      val updatedFiles = updatedPoms.map(_.getPomFile.getPath)
      updatedPoms.foreach(writeChanges)
      gitManager.applyChanges(updatedFiles, commitMessage)
    }
  }

  private def rootVersion: String = pomModels.flatMap(model => Option(model.getVersion)).head
```

```
private def rootVersion: String = pomModels.flatMap(model => Option(model.getVersion)).head

private def writeChanges(model: Model): Unit = managed(new FileWriter(model.getPomFile)).foreach(pomWriter.write(_, model)
```

```
private def findAllPomFiles(path: String): immutable.Seq[File] = Files.walk(Paths.get(path))  : Stream[Path]
  .iterator()                                                                                 : util.Iterator[Path]
  .asScala                                                                                    : Iterator[Path]
  .filter(_.getFileName.toString == "pom.xml")                                                : Iterator[Path]
  .map(_.toFile)                                                                              : Iterator[File]
  .toList                                                                                     : List[File]

private def extractMavenModels(path: String): List[Model] = {
  val pomFiles = findAllPomFiles(path)
  pomFiles.map(file => {
    val model = pomReader.read(new FileInputStream(file))
    model.setPomFile(file)
    model
  }).toList
}
```

```
  private def getAllDependenciesForRelease: List[(String, String)] = {
    val dependencies = pomModels.flatMap(_.getDependencies.asScala)
    val dependencyManagement = pomModels.flatMap(p => Option(p.getDependencyManagement)).flatMap(_.getDependencies.asScala)
    val maybeParent = pomModels.map(_.getParent.toDependency)
    val blacklist = pomModels.map(_.getArtifactId)

    val allDependencies = dependencies ++ dependencyManagement ++ maybeParent

    allDependencies
      .groupBy(d => (d.getGroupId, d.getArtifactId))
      .keys
      .filter({
        case (_groupId: String, artifactId: String) =>
          _groupId.startsWith(groupId) && !blacklist.contains(artifactId)
      })
      .toList
  }

}
```

Клас для адміністрування git репозиторіїв

```
import com.chystiakov.diploma.CodeCommitHelper._
import com.typesafe.scalalogging.LazyLogging
import org.eclipse.jgit.api.Git

import java.nio.file.{Files, Path, Paths}

class GitManager(artifactId: String) extends LazyLogging with Retry {

  val (repoName, cloneUrl) = getCloneUrl(artifactId)

  val path: Path = Paths.get( first= s"${System.getProperty("user.home")}/diploma/$repoName")

  val hasBeenClonedBefore: Boolean = Files.exists(path) && Files.isDirectory(path)

  private val clonedRepo: Git = retryWithBackOff( max= 10, sleepDur = 10000) {
    if (!hasBeenClonedBefore) {
      logger.info(s"Cloning $repoName ...")
      Git.cloneRepository().setURI(cloneUrl).setDirectory(path.toFile).call()
    } else Git.open(path.toFile)
  }

  def applyChanges(paths: Iterable[String], message: String): Unit = {
    val addCommand = clonedRepo.add()
    paths.foreach(addCommand.addFilepattern)
    addCommand.call()
    clonedRepo.commit().setMessage(message).call()
    logger.info("Committing and pushing changes ...")
    clonedRepo.push().call()
  }
```

```
  def releaseBranchCheckout(rootVersion: String): String = {
    val branch = s"release-$rootVersion"
    clonedRepo.branchCreate().setName(branch).call()
    logger.info(s"Checking out to the $branch ...")
    clonedRepo.checkout().setName(branch).call()
    branch
  }

  def mainBranchCheckout(branch: String = "main"): Unit = {
    logger.info(s"Checking out to the $branch ...")
    clonedRepo.checkout().setName(branch).call()
  }


}
```