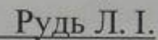
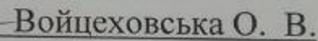
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

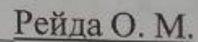
МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

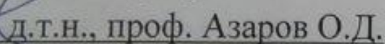
на тему:

«Програмний засіб безпечної контейнеризації веб-додатку»

Виконала: студент 2-го курсу, групи 1КІ-21м
спеціальності 123 — Комп'ютерна інженерія

_______________ Рудь Л. І.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ОТ

_______________ Войцеховська О. В.
(прізвище та ініціали)
«16» 12 2022 р.

Опонент: к.т.н., доц. каф. ПЗ

_______________ Рейда О. М.
(прізвище та ініціали)
«19» 12 2022 р.

Допущено до захисту
Завідувач кафедри ОТ

_______ д.т.н., проф. Азаров О.Д.
(прізвище та ініціали)
«19» 12 2022 р

Вінниця 2022

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо-кваліфікаційний рівень — магістр
Спеціальність 123 — «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
Азаров О. Д.
«15» вересня 2022 р

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Рудь Людмилі Ігорівні
(прізвище, ім'я, по-батькові)

1 Тема роботи: «Програмний засіб безпечної контейнеризації веб-додатку»

Керівник роботи: к.т.н., доцент кафедри ОТ Войцеховська Олена Валеріївна затверджені наказом Вінницького національного технічного університету від 15.09.2022 року № 205-А.

2 Строк подання студентом роботи 10.12.2022.

3 Вихідні дані до роботи: інформаційний контент для розгортання веб-додатку, методи розгортання та підтримки веб-додатків в хмарному середовищі, способи інтеграції тестування безпеки в CI/CD процес.

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): аналіз методів виявлення вразливостей та технології контейнеризації коду в CI/CD процесі; аналіз та вибір інструментів для розроблення та контейнеризації програмного засобу; розробка програмного компоненту безпечної контейнеризації веб-додатку, тестування та верифікація програмного засобу, економічна частина.

5 Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень): блок-схема процесу інтеграції програмного коду; структурна схема CI/CD-ковеєру системи безпечної контейнеризації веб-додатку; статистика знаходження вразливостей; запропонована структурна схема CI/CD-циклу.

6 Консультанти розділів роботи

Таблиця 6.1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1,2,3,4	**Войцеховська О.В., к.т.н., доцент каф. ОТ**		
5	**Небава М.І., к.е.н., професор каф. ЕПВМ**		

7 Дата видачі завдання____________________

8 Календарний план

Таблиця 8.1 — Календарний план

№ з/п	Назва етапів виконання бакалаврської дипломної роботи	Строк виконання етапів роботи	Підпис
1	Постановка задачі роботи	04.10.2022	вик.
2	Аналіз та дослідження методології DevSecOps, її переваг та методів.	15.10.2022	вик.
3	Аналіз використання контейнеризації та віртуалізації в хмарному середовищі.	04.11.2022	вик.
4	Програмна реалізація програмного компоненту безпечної контейнеризації веб-додатків.	21.11.2022	вик.
5	Підготовка матеріалів та опис розробки компоненту безпечної контейнеризації.	30.11.2022	вик.
6	Оформлення пояснювальної записки	03.12.2022	вик.
7	Перевірка якості виконання магістерської роботи	10.12.2022	вик.

Студент _________ Рудь Л. І.
(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _________ Войцеховська О. В.
(підпис) (прізвище та ініціали)

Консультант з економічної частини _________ Небава М. І.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.4

Рудь Л. І. Програмний засіб безпечної контейнеризації веб-додатку. Магістерська кваліфікаційна робота зі спеціальності 123 — Комп'ютерна інженерія, освітня програма — комп'ютерна інженерія. Вінниця: ВНТУ, 2022. 146 с.

Укр. мовою. Бібліогр.: 30 назв; рис. 50; табл.: 14.

В магістерській кваліфікаційній роботі досліджено та проаналізовано можливі засоби виявлення вразливостей в програмних додатках в процесі CI/CD циклу, а також метод розгортання додатку в ізольованому хмарному середовищі (контейнері) та його переваги. Розглянуто основні технології та інструменти, які дозволяють вбудовувати засоби виявлення вразливостей в безперервний цикл розробки та розгортання, з метою покращення процесу фіксування вразливостей безпеки. Розроблено методи доставки програмного продукту в хмару та його розгортання в контейнерному середовищі.

Розроблено тестовий програмний додаток на платформі ASP.NET Core для подальшого тестування безпеки та розгортання. Як засоби інтеграції та доставки використано Jenkins та хмарний сервіс AWS CodePipeline. Як інструмент проведення секьюріті-тестування обрано SonarQube, а як засіб контейнеризації використано Docker. Розроблений програмний продукт було розгорнуто в хмарному середовищі Amazon Web Services.

Ключові слова: DevSecOps, секьюріті-тестування, CI/CD процес, SonarQube, контейнер, хмарне середовище.

ABSTRACT

UDC 004.4

L. I. Rud. Software tool for secure containerization of a web application. Master's thesis in specialty 123 — computer engineering, educational program — computer engineering. Vinnytsia: VNTU, 2022.

Ukraine language Bibliography: 30 titles; Fig. 50; tab.: 14.

In the master's qualification work, the means of detecting vulnerabilities in software add-ons during the CI/CD cycle, as well as the method of deploying the add-on in an isolated cloud environment (containers) and its advantages, were investigated and analyzed. The main technologies and tools that allow the integration of vulnerability detection tools into the continuous cycle of development and deployment are considered, with the content of improving the process of fixing security vulnerabilities. Methods of delivering a software product to the cloud and deploying it in a container environment have been developed.

Developed a test software application on the ASP.NET Core platform for further security testing and deployment. Jenkins and the AWS CodePipeline cloud service were used as means of integration and delivery. SonarQube is chosen as a security testing tool, and Docker is used as a containerization tool. The developed software product was deployed in the Amazon Web Services cloud environment.

Keywords: DevSecOps, security testing, CI/CD process, SonarQube, container, cloud environment.

ЗМІСТ

ВСТУП 9
1 АНАЛІЗ МЕТОДІВ ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ ТА ТЕХНОЛОГІЇ КОНТЕЙНЕРИЗАЦІЇ КОДУ В CI/CD ПРОЦЕСІ 12
1.1 Інтеграція компоненту безпеки в цикл розробки як частина методології DevSecOps 12
1.2 Аналіз вразливостей та вимог до тестування безпеки додатків 16
1.3 Аналіз існуючих рішень виявлення вразливостей 20
1.4 Проблеми секьюріті-тестування та способи їх вирішення 21
1.5 Аналіз та порівняння технологій контейнеризації та віртуалізації в хмарних обчисленнях 22
1.6 Переваги застосування контейнеризації в CI/CD-конвеєрі 25
1.7 Контейнеризація в хмарному середовищі 27
1.8 Постановка задачі дослідження 30
2 АНАЛІЗ ТА ВИБІР ІНСТРУМЕНТІВ ДЛЯ РОЗРОБЛЕННЯ ТА КОНТЕЙНЕРИЗАЦІЇ ПРОГРАМНОГО ЗАСОБУ 32
2.1 Аналіз вимог до програмного засобу 32
2.2 Розробка методу тестування безпеки програмного продукту 33
2.2.1 Основні компоненти розроблюваного програмного засобу 34
2.2.2 Вибір програмного забезпечення для безперервної інтеграції коду 35
2.2.3 Вибір програмного забезпечення для проведення тестування безпеки додатку 37
2.2.4 Вибір системи управління версіями та репозиторію 39
2.2.5 Вибір засобу контейнеризації додатку 40

2.3 Розробка структурної схеми CI/CD-ковеєра безперервної інтеграції та розгортання 41
3 РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ БЕЗПЕЧНОЇ КОНТЕЙНЕРИЗАЦІЇ ВЕБ-ДОДАТКУ 44
3.1 Реалізація тестового проекту 44
3.2 Налаштування хмарного репозиторію AWS CodeCommit і його інтеграція з Git 48
3.3 Налаштування серверу безперервної інтеграції Jenkins та конвеєру AWS CodePipeline 52
3.4 Налаштування серверу для постійного аналізу та вимірювання якості коду SonarQube 59
3.5 Налаштування реєстру контейнерів Amazon ECR 67
3.6 Налаштування служби оркестрування контейнерів Amazon ECS 70
3.7 Налаштування стратегії розгортання контейнерів на екземплярах Amazon EC2 та групи автоматичного масштабування 76
3.8 Автоматизація керування об’єктами AWS за допомогою скрипта Python 79
3.9 Аналіз ефективності розробленого програмного засобу 82
4 ТЕСТУВАННЯ ТА ВЕРИФІКАЦІЯ ПРОГРАМНОГО ЗАСОБУ 85
4.1 Функції програмного засобу, які необхідно протестувати 85
4.2 Верифікація програмного засобу 85
4.3 Тестування програмного засобу 86
5 ЕКОНОМІЧНА ЧАСТИНА 90
5.1 Комерційний та технологічний аудит науково-технічної розробки 90
5.2 Прогнозування витрат на виконання науково-дослідної роботи 93
5.3 Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором 98
5.3.1 Розробка чи суттєве вдосконалення програмного засобу для використання масовим споживачем 99

5.3.2 Розрахунок ефективності вкладених інвестицій та періоду їх окупності 100
ВИСНОВКИ 104
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 106
ДОДАТОК А Технічне завдання 110
ДОДАТОК Б Блок-схема процесу інтеграції програмного коду 114
ДОДАТОК В Структурна схема CI/CD-ковеєру системи безпечної контейнеризації веб-додатку 115
ДОДАТОК Г Статистика знаходження вразливостей 116
ДОДАТОК Д Запропонована структурна схема CI/CD-циклу 117
ДОДАТОК Е Вміст файлу Task Definition в форматі JSON 118
ДОДАТОК Ж Конфігураційний файл .yml для налаштування SonarQube 121
ДОДАТОК И Лістинг скрипта для керування об’єктами AWS з використанням бібліотеки Python SDK 123
ДОДАТОК К Протокол перевірки навчальної (кваліфікаційної) роботи...146

ВСТУП

В сучасному світі проблема безпеки даних стає все більш актуальною і вимагає впровадження нових технологій та інструментальних засобів. Існуючі на даний момент підходи не здатні цілком вирішити проблему безпеки і повноцінно ізолювати продукт від хакерських атак. За ідеальних умов, зміни в коді проекту мають бути доставлені у середовище розгортання без використання оточення для тестування, тому дуже важливо забезпечити неможливість успішної збірки проекту у разі потрапляння неякісного коду до артефакту збірки (набір вихідних файлів, придатних для скачування чи використання при роботі системи) або у разі знаходження потенційних вразливостей.

На даний час проведення тестування безпеки проводиться відокремленою командою (це можуть бути як ручні так і автоматизовані тестувальники) з використанням окремого оточення, причому подібна перевірка не проводиться при кожній зміні у коді, що значно збільшує шанси пропущення вразливостей безпеки у системі до середовища розгортання або потрапляння неякісного коду до продукту.

Враховуючи той факт, що виявлення та виправлення помилок потребує багато часу та ресурсів команди, з'являється необхідність використовувати інший підхід до тестування безпеки продукту, який є значно ефективнішим — DevSecOps. Його головною ідеєю є впровадження автоматизованого тестування у CI/CD процес.

Використання такого підходу у поєднанні з контейнеризацією дозволить будувати повністю автоматизовані ланцюжки безперервної інтеграції-розгортання додатків (CI/CD), в яких ручною частиною буде в основному написання коду. Тестування, перевірку якості коду, упаковку програми в образ контейнера та розгортання його на сервері можна виконувати без участі людини.

Метою роботи є створення програмного засобу для автоматизації проведення тестування безпеки програмного продукту, який стане частиною CI/CD процесу та дозволить знизити ймовірність потрапляння неякісного коду до середовища.

Для досягнення поставленої мети необхідно вирішити такі **задачі**:

— проаналізувати засоби сучасних технологій розробки веб-додатків, їх особливостей та вразливостей;

— розглянути особливості методології DevSecOps та її методи;

— розробити метод тестування безпеки продукту з автоматизованим тестуванням у CI/CD процесі;

— проаналізувати та вибрати інструменти для розроблення та контейнерізації програмного засобу;

— створити компонент забезпечення кібербезпеки в циклі розробки програмного забезпечення (ПЗ) та аналізу коду на вразливості;

— забезпечити захищеність даних та доступу до хмарного середовища;

— розробити програмний засіб інтеграції ПЗ в ізольоване хмарне середовище з використанням розробленого компоненту кібербезпеки;

— провести тестування та верифікацію готового програмного рішення.

Об'єкт дослідження — CI/CD процес безперервної інтеграціїї та безперервного розгортання програмного продукту, який відповідає за проведення тестування вразливостей безпеки.

Предмет дослідження — методи та програмні засоби інтеграції коду в хмарному середовищі від першого коміту до повної перевірки, створення артефакту збірки та розгортання на оточенні.

Наукова новизна полягає в розробці нового методу тестування безпеки продукту з впровадженням автоматизованого тестування у CI/CD процес, який дасть можливість знизити вірогідність потрапляння неякісного коду до середовища розгортання.

Практичне значення отриманих результатів полягає у тому, що:

— розроблено програмний засіб інтеграції компоненту тестування кібербезпеки в цикл CI/CD продукту, та подальшого розгортання в контейнерному середовищі, що блокує можливість попадання неякісного коду до продукту, значно знижує ймовірність виникнення вразливості у програмі та суттєво скорочує час перевірки та фіксації знайдених проблем;

— розроблено алгоритм процесу інтерації програмного коду в хмарне середовище;

— розроблено структурну схему CI/CD-ковеєра безперервної інтеграції та розгортання.

Апробація результатів бакалаврської роботи: опубліковано доповіді на L науково-технічній конференції підрозділів Вінницького національного технічного університету та Всеукраїнській науково-практичній інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2023)».

1 АНАЛІЗ МЕТОДІВ ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ ТА ТЕХНОЛОГІЇ КОНТЕЙНЕРИЗАЦІЇ КОДУ В CI/CD ПРОЦЕСІ

1.1 Інтеграція компоненту безпеки в цикл розробки як частина методології DevSecOps

Розвиток інформаційних технологій в кінці минулого — початку нинішнього століть привів до виникнення нового напряму в науці — інформаційної безпеки. Цьому питанню були присвячені монографії, наукові статті багатьох вчених [1 — 3]. У роботах була показана залежність інформаційної безпеки від часу і представлена її математична модель. Подальшим розвитком інформаційної безпеки стала кібербезпека (КБ), яка замінила її в інформаційних технологіях.

Одним з головних превентивних методів забезпечення КБ є пошук і усунення різного роду вразливостей, що виникають в процесі розробки програмного продукту. Вразливості дозволяють здійснювати різні атаки, за допомогою яких можна отримати доступ до сайтів різних компаній і відомств, до даних кредитних карт, персональних даних громадян і т.п. Даними, що викрадають найбільше, стали медична інформація та дані платіжних карт.

Програмні додатки складні і потенційно можуть мати безліч різних проблем безпеки. Проблеми варіюються від поганого коду до неправильно сконфігурованих серверів і всього устаткування між ними. На даний момент існує досить багато способів тестування безпеки програмного продукту зі своїми перевагами і недоліками. Незважаючи на це, проблема злому і хакерства все також актуальна і, на жаль, набирає обертів з такою ж швидкістю, з якою розвивається сфера інформаційних технологій. Одним із шляхів вирішення представленої задачі є створення програмного продукту DevSecOps. DevSecOps — це нова ідеологія, при якій кожен програміст відповідає за безпеку свого продукту. Мета DevSecOps — вивести розробників всіх напрямків на високий рівень професіоналізму в області безпеки за короткий проміжок часу. DevSecOps націлений на те, щоб кожен недолік безпеки, після знаходження,

був своєчасно ідентифікований і зафіксований, для нього був створений автотест і доданий в процес постійної збірки, щоб закрити найбільш термінові і важливі прогалини в безпеці [4]. На рисунку 1.1 показано як DevSecOps інтегрується у розробку програмного продукту.

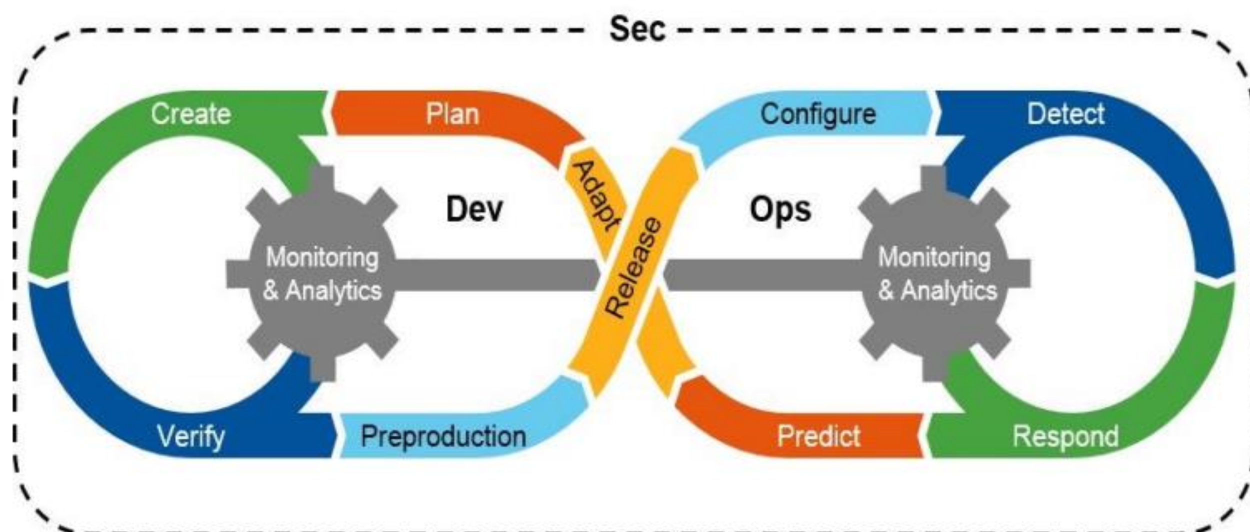


Рисунок 1.1 — Схема інтеграції DevSecOps в розробку програмного продукту

Аналіз технології використання програмного продукту DevSecOps для тестування безпеки представимо у вигляді використовуваних підходів або ідей для реалізації поставленої мети.

Незважаючи на зростаючу стурбованість з приводу ризиків використання сторонніх програмних компонентів, компанії використовують програмне забезпечення з відкритим вихідним кодом в додатках. Окремий аудит, проведений більш ніж в 1000 комерційних додатках, показав, що 96% з них включають компоненти з відкритим вихідним кодом. Більше 6 з 10 додатків містять відомі вразливості безпеки в цих компонентах, а деякі існували там вже декілька років. Незважаючи на це, тільки 27% респондентів заявили, що у них є процеси автоматичної ідентифікації та відстеження виправлень для відомих недоліків в програмному забезпеченні з відкритим вихідним кодом.

Розуміння використання відкритого вихідного коду є ключем до більш широкого впровадження методів DevSecOps. У розробників часто не вистачає

часу на перегляд коду в своїх бібліотеках з відкритим вихідним кодом або читання документації, тому автоматичні процеси управління компонентами з відкритим вихідним кодом і сторонніми розробниками є основною вимогою для DevSecOps [5].

Інструменти SAST дозволяють розробникам отримувати миттєві відгуки про недоліки, які можуть викликати проблеми безпеки під час написання коду. Ці інструменти допомагають розробникам виявляти і усувати потенційні вразливості безпеки під час звичайного робочого процесу і тому повинні бути важливим компонентом практики DevSecOps. Часто, коли команда безпеки впроваджує засіб статичного тестування в ланцюжку CI/CD, існує тенденція включати перевірки всієї множини проблем безпеки і це закінчується іншими проблемами, що несуть за собою труднощі підтримки подібних процесів. Замість цього набагато краще включити одну або дві перевірки безпеки за один раз і змусити розробників використовувати ідею включення правил безпеки в робочий процес. Наприклад, при впровадженні інструменту SAST в розробці є можливість почати з включення тільки тестів для уловлювання помилок SQL-ін'єкцій. Як тільки розробники дізнаються, як інструмент допомагає їм відстежувати помилки під час кодування, вони з більшою ймовірністю будуть працювати з ним.

Щодня з'являються нові інструменти, необхідні для тестування безпеки, тому існує кілька ключових моментів при їх виборі:

— продукти безпеки повинні бути здатні інтегруватися в конвеєр розробки і дозволяти команді розробників і безпеки працювати разом, а продукт тестування безпеки повинен спростити розробникам швидке ініціювання тестування і отримання результатів без необхідності залишати свій існуючий набір інструментів;

— іншими ключовими вимогами є швидкість і точність: засоби безпеки повинні працювати швидко;

— результати, які генерують інструменти, повинні бути швидкими, точними і негайними.

Рекомендується застосовувати моделювання загроз та оцінки ризиків до переходу на DevSecOps. Проведення моделювання загроз в середовищі DevOps може бути складним через те, що воно може уповільнити швидкість процесу CI/CD. Але моделювання загроз, як і раніше, має вирішальне значення для загального успіху зусиль DevOps, тому що воно змушує розробників думати про своє програмне забезпечення з точки зору зловмисника.

При використанні DevSecOps можна зіткнутися з деякими проблемами. Отримання інвестицій та часу, необхідних для підготовки команди розробників до безпечного кодування є однією з них. Але очевидно, що в світі безперервної інтеграції і швидких циклів випуску більше немає можливості ігнорувати безпеку додатків.

Перевагою DevSecOps є автоматизація процесів з самого початку процесу програмування, що дозволяє знизити ймовірність неправильного адміністрування і помилок, які часто призводять до простоїв або відкривають можливості для атак [6]. Також автоматизація позбавляє ІБ-фахівців від необхідності налаштовувати консолі вручну. Таким чином, функції безпеки такі, як управління ідентифікацією, доступ IAM (Identity and Access Management), робота брандмауерів і сканування вразливостей активуються програмно, в процесі DevOps. В результаті такого підходу команди ІБ-фахівців можуть зосередитися на установці політик. Фахівці вважають, що 80% команд розробників використовуватимуть DevSecOps до 2023 року.

Ще одна з основних переваг DevSecOps полягає в тому, що за безпеку відповідає кожна команда, що бере участь в розробці. Такий підхід приводить до створення спеціальних інструментів, спрямованих на підвищення безпеки на різних етапах ланцюжка DevOps.

Як основний недолік слід відзначити той факт, що кількість фахівців в галузі безпеки, добре обізнаних в DevSecOps, як і раніше залишається малою. Підготовка таких фахівців потребує додаткових витрат часу та ресурсів на навчання [7].

Приклади проблем безпеки, які були зафіксовані на етапі розробки за допомогою підходу DevSecOps:

— закінчення терміну дії сертифіката;

— застарілі бібліотеки;

— уразливості коду;

— відповідність OWASP;

— секретні загрози.

1.2 Аналіз вразливостей та вимог до тестування безпеки додатків

В умовах безперервної інтеграції та безперервного розгортання (CI/CD), реалізується можливість швидко отримати працюючий програмний продукт. Для забезпечення безпеки, програмний компонент повинен бути автоматизований щоб бути частиною цього робочого процесу. Контроль безпеки і тести повинні бути впроваджені на ранньому етапі і всюди в життєвому циклі розробки, і вони повинні проводитись в автоматичному режимі, оскільки розробники вводять нові версії коду в виробництво по десятки разів на день для однієї програми. Це має велику перевагу порівняно з існуючою моделлю розробки, в якій автоматичні тести безпеки запускаються безпосередньо перед розгортанням кінцевого продукту.

З'явилося все більше інструментів з широким спектром можливостей для проведення аналізу безпеки і тестування протягом всього життєвого циклу розробки програмного забезпечення, починаючи з аналізу вихідного коду і закінчуючи інтеграцією і моніторингом після розгортання. До них відносяться SpotBugs, Checkmarx, Splunk, Contrast Security, Sonatype, Tanium, InSpec, FireEye і Metasploit [8].

Можливість проведення динамічного та статичного аналізу коду одночасно є ще однією вимогою до тестування безпеки додатків. Намагаючись запускати автоматичне статичне тестування всього вихідного коду програми кожен день, можна витратити багато часу і втратити можливість відстежувати щоденні зміни. Також потрібно розглядати можливість впровадження

автоматизованого тестування безпеки динамічних додатків (DAST) в життєвий цикл розробки програмного забезпечення. На відміну від статичного аналізу, який фокусується на пошуку потенційних проблем безпеки в самому коді, DAST шукає вразливості в реальному часі, в той час як додаток працює. Автоматизація DAST сканує і запускає тести для недавніх або нових змін коду з метою виявлення вразливостей безпеки, перерахованих в списку найбільш поширених недоліків Open Web Application Security Project (OWASP), таких як помилки SQL-ін'єкції, які можна пропустити під час статичного аналізу [9].

Тестування безпеки повинно забезпечити можливість убезпечити продукт від помилок розробників. З метою усунення ризиків використання сторонніх програмних компонентів, компанії використовують програмне забезпечення з відкритим вихідним кодом у додатках. В ході роботи необхідно стежити чи використовує реалізація з відкритим вихідним кодом контекстуальні та інші вразливості в коді і який вплив ці вразливості можуть мати на залежний код. Перевірки вразливості коду є важливими для програмного компоненту, що розробляється, і такі утиліти, як OWASP Dependency-Check, що запобігають використанню коду з відомими вразливими в програмному забезпеченні. Утиліта OWASP працює шляхом тестування коду і залежних бібліотек компонентів з відкритим вихідним кодом, щоб дізнатися, чи містять вони будь-які ключові недоліки OWASP. Вона працює з постійно оновленою базою даних всіх відомих вразливостей в програмному забезпеченні з відкритим вихідним кодом.

Вразливості, що зустрічаються найчастіше в існуючих програмах наведені в таблиці 1.1 [10].

Таблиця 1.1 — Перелік вразливостей, що зустрічаються найчастіше

Назва вразливості	Опис
XSS (Cross-Site Scripting)	Вид вразливості програмного забезпечення (Web додатків), при якій, на генерованій сервером сторінці, виконуються шкідливі скрипти, з метою атаки клієнта.

Продовження таблиці 1.1

Назва вразливості	Опис
XSRF / CSRF (Request Forgery)	Вид вразливості, що дозволяє використовувати недоліки HTTP протоколу, при переході по шкідливому посиланню виконується скрипт, який зберігає особисті дані користувача (паролі, платіжні дані та інше).
Code injections (SQL, PHP, ASP та інше)	Вид вразливості, при якому стає можливо здійснити запуск виконуваного коду з метою отримання доступу до системних ресурсів, несанкціонованого доступу до даних або виведення системи з ладу.
Server-Side Includes (SSI) Injection	Вид вразливості, що використовує вставку серверних команд в HTML код або запуск їх безпосередньо з сервера.
Authorization Bypass	Вид вразливості, при якому можливо отримати несанкціонований доступ до облікового запису або даних іншого користувача.

На рисунках 1.2 — 1.4 наведена статистика знайдених вразливостей у період 2019 — початку 2021 року:

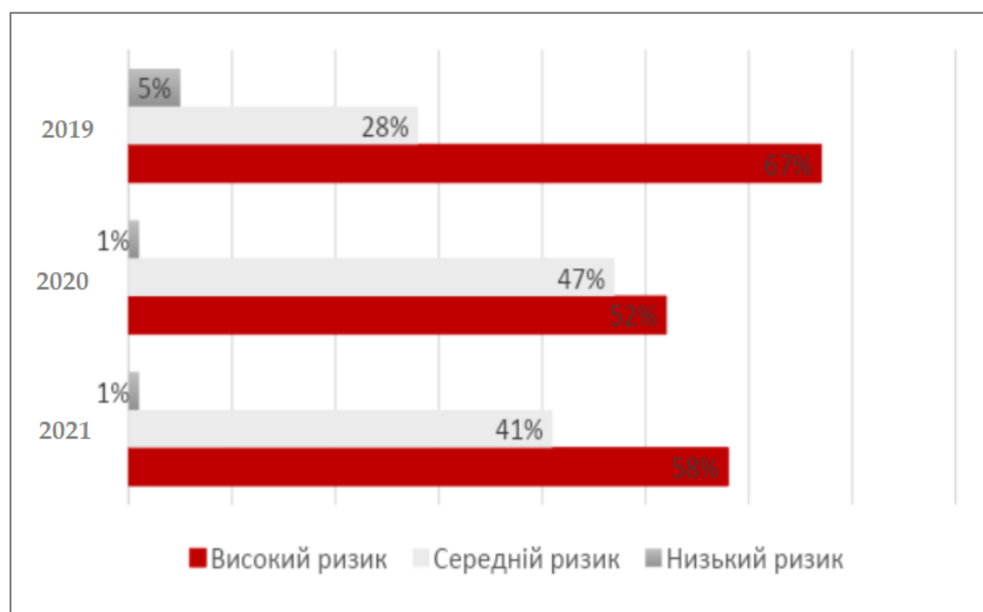


Рисунок 1.2 — Частка вразливих сайтів залежно від максимального ступеня ризику вразливостей

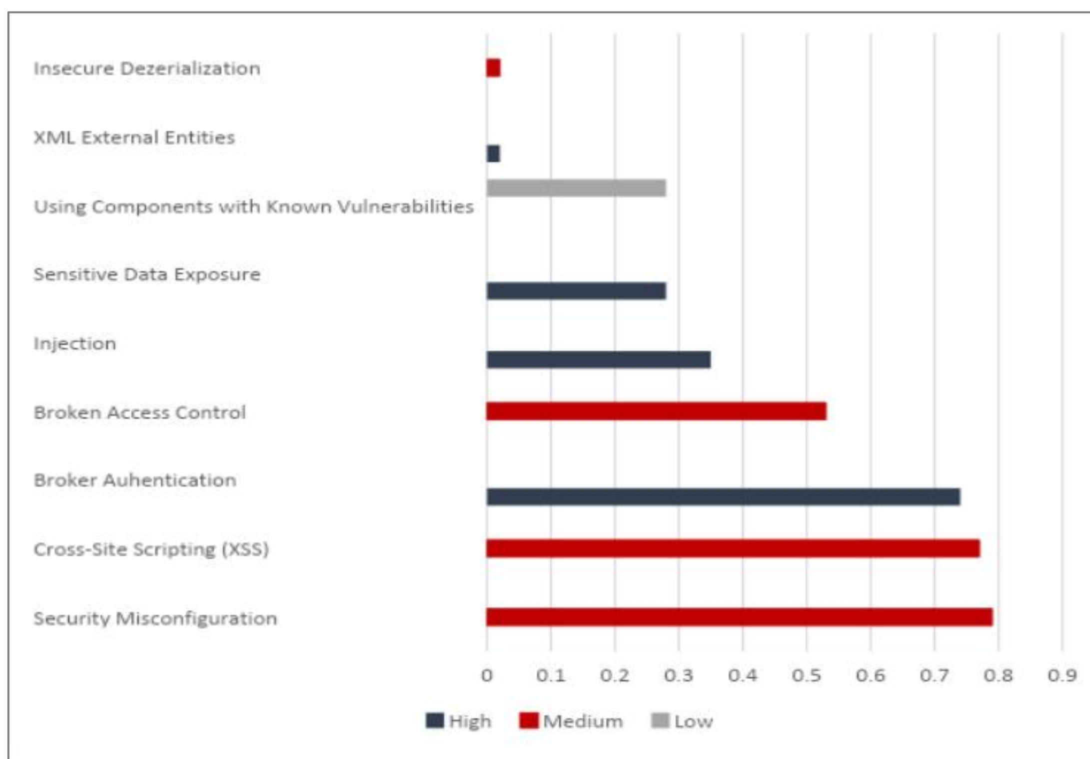


Рисунок 1.3 — Статистика вразливостей, що зустрічаються найчастіше

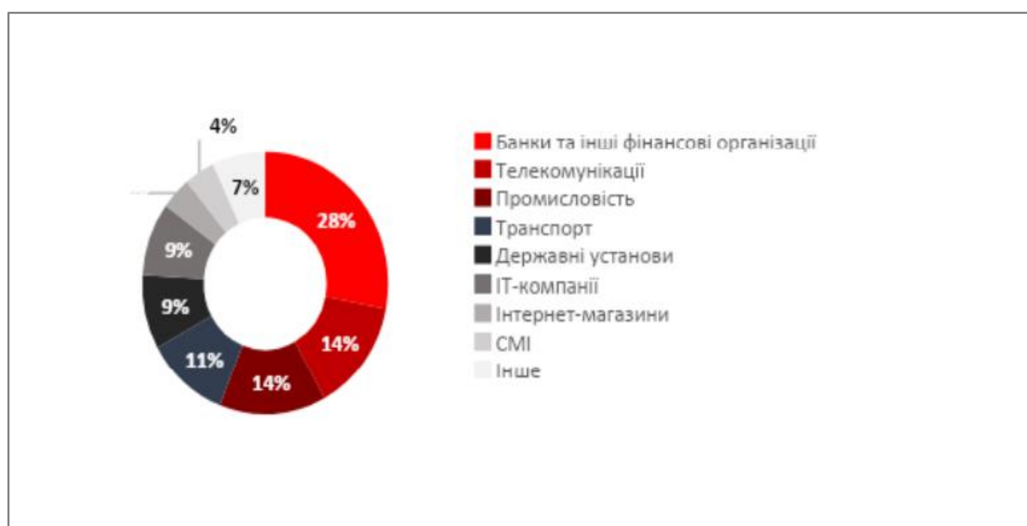


Рисунок 1.4 — Області використання продуктів, що мають найбільше вразливостей

Програмні компоненти подібні до OWASP, дозволяють розробникам отримувати миттєві відгуки про недоліки, які можуть викликати проблеми безпеки під час написання коду. Ці інструменти допомагають розробникам виявляти і усувати потенційно вразливі місця безпеки під час звичайного

робочого процесу і тому повинні бути важливим компонентом практики DevOps.

1.3 Аналіз існуючих рішень виявлення вразливостей

На даний час статичне та динамічне тестування безпеки програмного продукту виконується окремо один від одного на різних стадіях розробки. Для більшості проектів лише статичне тестування є частиною безперервної збірки, у той час як динамічне тестування безпеки проводиться лише після розгортання створеного артефакту на оточенні.

Для статичного аналізу коду частіше використовують такий аналізатор як Sonar. SonarQube — платформа з відкритим вихідним кодом для безперервного аналізу і визначення якості коду. Підтримує аналіз коду і пошук помилок згідно з правилами стандартів програмування MISRA C, MISRA C ++, MITRE / CWE і CERT Secure Coding Standards. Також вона вміє розпізнавати помилки зі списків OWASP і CWE/SANS. Не дивлячись на те, що платформа використовує різні готові інструменти, SonarQube зводить результати до єдиної інформаційної панелі, веде історію прогонів і дозволяє побачити загальну тенденцію зміни якості програмного забезпечення в ході розробки.

Динамічне тестування безпеки додатків (DAST) — це технологія, яка здатна знаходити видимі уразливості шляхом подачі URL-адреси в автоматичний сканер. Цей метод легко масштабується, легко інтегрується і є швидким. Недоліки DAST полягають в необхідності експертної конфігурації і високій ймовірності помилкового спрацьовування.

Інтерактивне тестування безпеки додатків (IAST) — це рішення, яке оцінює додатки зсередини, використовуючи програмні засоби. Цей метод дозволяє IAST використовувати переваги SAST і DAST, а також надає доступ до коду, HTTP-трафіку, інформації про бібліотеки, внутрішніх з'єднаннях та інформації про конфігурацію. Деякі продукти IAST вимагають, щоб додаток піддавався атаці, в той час як інші можуть використовуватися під час звичайного тестування якості [11].

Таким чином, використовуючи подібний підхід є великий ризик допустити потрапляння вразливостей безпеки до артефакту, що може призвести до витоку даних користувачів тощо, бо на постійній основі проводиться лише статичний аналіз, який не може гарантувати повної безпеки коду.

1.4 Проблеми секьюріті-тестування та способи їх вирішення

Після проведення аналізу існуючих рішень для забезпечення безпеки програмного продукту були виявлені такі недоліки у секьюріті-тестуванні:

Динамічне тестування коду проходить лише на фінальній стадії розробки продукту, що робить будь-яке виправлення у коді значно дорожчим ніж якщо б ця помилка була виявлена під час першого формування артефакту з використанням цього коду.

На рисунку 1.5 наведено статистику проведення статичного та динамічного тестування безпеки на період 2019 — початок 2021 року.

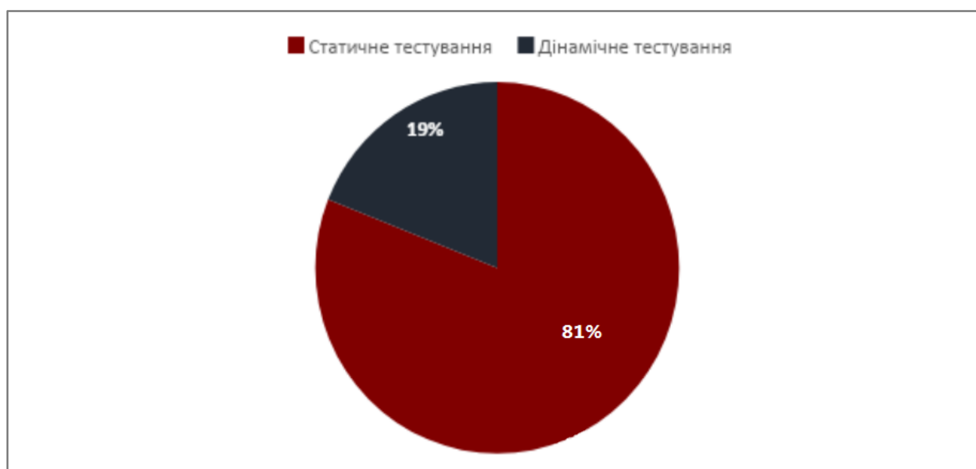


Рисунок 1.5 — Статистика проведення статичного та динамічного тестування безпеки на період 2019 — початок 2021 року

На даний час динамічне секьюріті-тестування проходить не перед кожним релізом, що заздалегідь піддає сумнівам безпеку даних у програмному продукті. Проведення тестування безпеки програмного продукту вимагає ресурсів, в той час як використання програмного компоненту в процесі

формування збірки є повністю автоматизованим і потребує змін лише у випадку знаходження нової проблеми.

Для динамічного тестування було обрано такий програмний продукт, як SonarQube. SonarQube — це автоматизоване рішення для динамічного тестування, яке виявляє проблеми з конфігурацією та ідентифікує і визначає пріоритети вразливостей безпеки в запущених програмах. Він імітує реальні методи злому і забезпечує всебічний динамічний аналіз складних веб-додатків і сервісів. Панелі та звіти надають чіткість та точну позицію ризику для програм.

SonarQube дозволяє спостерігати реакцію програми на атаки на рівні коду під час динамічного сканування, сканувати великі програми, розширюючи покриття поверхні атаки, а також надавати стеки і SQL-запити до підтверджених вразливостей. SonarQube надає можливість створювати гнучкі, розширювані та масштабовані звіти, які відповідають бізнес-вимогам [12].

Таким чином використання запропонованого підходу дозволяє зекономити бюджет проекту, не вимагає додаткових ресурсів для підтримки, запобігає появі неякісного коду у середовищі розгортання.

1.5 Аналіз та порівняння технологій контейнеризації та віртуалізації в хмарних обчисленнях

Контейнерізація та віртуалізація є двома ключовими технологіями хмарних обчислень. Віртуалізація відновила популярність із заснуванням VMWare, а контейнеризація стала надзвичайно популярною за останнє десятиліття завдяки платформі Docker. Docker — це технологія, яка надає інструменти для створення, запуску, тестування та розгортання розподілених програм у контейнерах [13]. Використовуючи послуги від постачальника хмарних послуг, швидше за все, буде використовуватись одна з цих технологій при розгортанні додатків в хмарному середовищі.

Віртуалізація — це технологія, яка створює абстрактний рівень над апаратним забезпеченням комп’ютера, який дозволяє використовувати апаратне забезпечення одного комп’ютера та розділяти його на кілька

віртуальних комп’ютерів, відомих як віртуальні машини [14]. Кожна віртуальна машина працює під керуванням власної операційної системи. Віртуальні машини можна використовувати, якщо потрібні різні або кілька однакових операційних систем на одному фізичному комп’ютері (Рисунок 1.6).

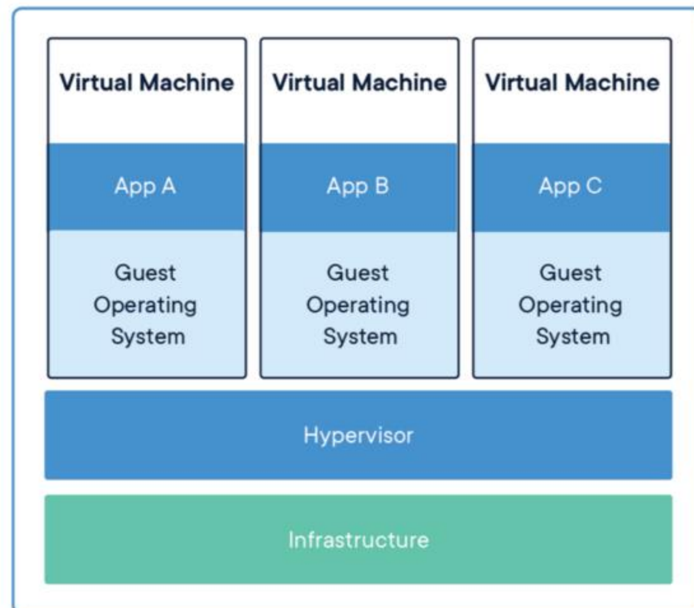


Рисунок 1.6 — Структурна схема віртуалізації

Контейнерізація — це упаковка програмного коду лише з бібліотеками операційної системи і залежностями, необхідними для запуску коду для створення єдиного легкого виконуваного файлу, який називається контейнером, який працює послідовно в будь-якій інфраструктурі [15]. Контейнерізацію можна використовувати для розміщення мікросервісів в ізольованому середовищі, наприклад REST API (Рисунок 1.7).

Хоча віртуалізація та контейнеризація є подібними технологіями, які часто плутають одна з одною, у них є деякі суттєві відмінності.

Однією з таких відмінностей є операційна система. Під час створення віртуальної машини необхідно вибрати операційну систему для встановлення. З іншого боку, контейнер використовує ядро поточної операційної системи та додає бібліотеки та залежності з вибраного образу. Це призводить до того, що

віртуальні машини займають набагато більше місця для зберігання, ніж контейнери.

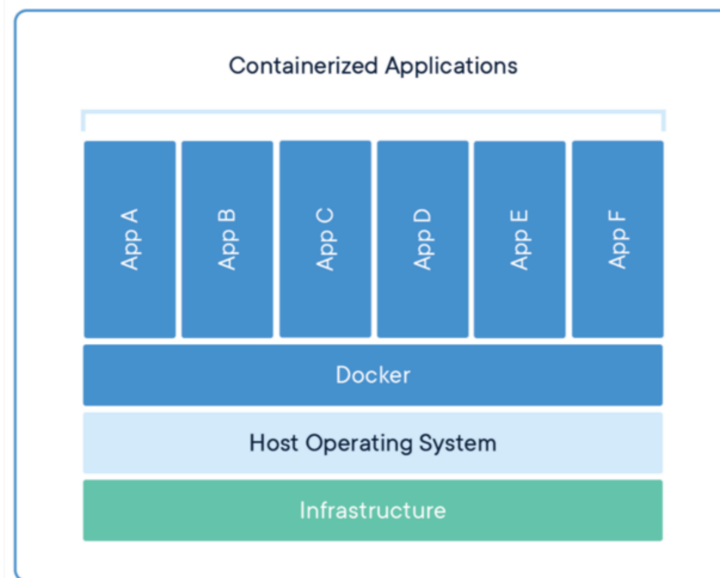


Рисунок 1.7 — Структурна схема контейнеризації

Ще одна відмінність між контейнеризацією та віртуалізацією — це витрати. Docker є легким та має низьку складність порівняно зі складнішою віртуальною машиною, що призводить до менших витрат.

Значною перевагою контейнеризації є швидкість, з якою можна запустити програму. Контейнер не потребує встановлення ОС або створення користувачів у системі. Це суттєва відмінність від віртуальної машини, де потрібно встановити та налаштувати всю ОС, перш ніж мати можливість запустити свою програму [16].

Варто зазначити, що під час розгортання програми, яка використовує HTTP-запити в певній формі, все зводилося б до трафіку та дій, які виконує сервер. Якщо програма, про яку йде мова, матиме відносно низький трафік, але вирішуватиме завдання, що вимагають високої продуктивності ЦП, тоді рішення Docker може стати правильним виходом. З іншого боку, якщо очікується, що програма матиме дуже високий трафік, але навантаження на процесор не є особливо вимогливим, тоді віртуальна машина може бути кращим вибором [17].

Віртуалізація та запровадження центрів обробки даних позитивно вплинули на компанії з точки зору фінансового аспекту, обмеживши вимоги щодо придбання апаратного забезпечення для розміщення їх продуктів та підтримки інфраструктури розміщення своїх серверів усередині компанії. В роботі [18] виявили, що менші компанії могли б заощадити понад 200% поточного бюджету IT-інфраструктури цих компаній, впровадивши хмарні обчислення замість розміщення власної серверної інфраструктури.

Впровадження віртуалізації також впливає на середовище, оскільки все більше компаній використовують хмарні сервіси для своїх продуктів, що на практиці змушує більше компаній спільно використовують обладнання, якщо вони розміщені в одному центрі обробки даних.

Хмарні обчислення — це практика, де контейнеризація та віртуалізація значною мірою реалізовані в різних типах послуг, які пропонують постачальники хмарних послуг. В роботі [19] підкреслюється, що віртуалізація є важливою частиною у визначенні хмарних обчислень, і з її допомогою ресурсами, віртуалізованими в хмарі, можна керувати ефективніше. Віртуалізована хмарна інфраструктура забезпечує абстракцію, необхідну для забезпечення незалежності програми або моделі бізнес-сервісу від основного апаратного забезпечення, такого як сервери, сховище або мережі.

1.6 Переваги застосування контейнеризації в CI/CD-конвеєрі

Сьогодні організації все частіше використовують контейнеризацію для створення нових програм і модернізації існуючих програм для хмари. У нещодавньому опитуванні IBM 61% користувачів контейнерів повідомили про використання контейнерів у 50% або більше нових програм, які вони створили протягом попередніх двох років; 64% користувачів очікують, що більше 50% їхніх існуючих додатків будуть розміщені в контейнерах протягом наступних двох років.

Більш портативні та ресурсоефективні, ніж віртуальні машини, контейнери де-факто стали обчислювальними одиницями сучасних хмарних додатків.

Контейнерізація дозволяє розробникам створювати та розгортати програми швидше та безпечніше. За допомогою традиційних методів код розробляється в спеціальному обчислювальному середовищі, яке, перенесене в нове місце, часто призводить до помилок. Наприклад, коли розробник переносить код із настільного комп'ютера на віртуальну машину або з операційної системи Linux на операційну систему Windows [20].

Контейнери часто називають «легкими», тобто вони мають спільне ядро операційної системи машини і не потребують накладних витрат на інтеграцію кожної програми з операційною системою. Контейнери за своєю суттю мають меншу місткість, ніж віртуальна машина, і потребують менше часу на запуск, що дозволяє працювати набагато більшій кількості контейнерів із тією ж обчислювальною потужністю, що й одна віртуальна машина. Це підвищує ефективність серверів і, у свою чергу, зменшує витрати на сервер і ліцензування.

Контейнери інкапсулюють програму як єдиний виконуваний пакет програмного забезпечення, який об'єднує код програми разом із усіма пов'язаними конфігураційними файлами, бібліотеками та залежностями, необхідними для її роботи. Контейнерні програми «ізольовані» в тому, що вони не об'єднуються в копію операційної системи. Замість цього механізм виконання з відкритим кодом (наприклад, механізм виконання Docker) встановлюється в операційну систему хоста і стає каналом для контейнерів, які спільно використовують операційну систему з іншими контейнерами в одній обчислювальній системі.

Ізоляція програм, як контейнерів, також зменшує ймовірність того, що шкідливий код, присутній в одному контейнері, вплине на інші контейнери або вторгнеться в хост-систему.

Абстрагування від основної операційної системи робить контейнерні програми портативними та здатними працювати однаково та узгоджено на будь-якій платформі чи хмарі. Контейнери можна легко транспортувати з настільного комп’ютера на віртуальну машину (VM) або з операційної системи Linux на операційну систему Windows, і вони будуть працювати узгоджено на віртуалізованих інфраструктурах або на традиційних «голих» серверах, локальних або всередині хмари. Це гарантує, що розробники програмного забезпечення можуть продовжувати використовувати інструменти та процеси, які їм найбільше зручні.

Контейнерізація дозволяє розробникам створювати та розгортати додатки швидше та безпечніше, незалежно від того, чи є це традиційним монолітом (однорівневим програмним додатком) чи модульним мікросервісом (сукупністю слабко пов’язаних служб). Нові хмарні програми можна створювати з нуля як контейнерні мікросервіси, розбиваючи складну програму на серію менших спеціалізованих і керованих служб. Існуючі програми можна переупакувати в контейнери (або контейнерні мікросервіси), які ефективніше використовують обчислювальні ресурси [21].

1.7 Контейнеризація в хмарному середовищі

Поява хмарних обчислень значно змінила способи розміщення та доставки програмного забезпечення. Практично для всього, що сьогодні потрібно, існують веб-сервіси — це стосується сфери торгівлі та інфраструктури обчислень.

Якщо автоматизація складання, створення середовищ та тестування супроводжується практикою «невеликих і частих змін», це скорочує затримку між розробкою та релізом, а також дозволяє бути впевненішими в якості продукту.

Пізніші стадії CI/CD-процесу, як правило, включають наскрізне тестування та тестування продуктивності, для яких потрібні оточення, що досить точно повторюють продакшн. Щоб тестування було максимально

ефективним та дотримувалося принципу консистентності, ці оточення повинні оновлюватися автоматично, а не вручну. Щоб реалізувати це, потрібні не тільки навички та інструменти DevOps, але й інфраструктура, яка забезпечила б необхідні обчислювальні потужності для Continuous Integration (CI) сервера, білд-агентів, тестових середовищ і сховищ даних.

Те, скільки пайплайну (конвеєру інтеграції та розгортання) знадобиться пристроїв, залежить від обсягу та складності проекту та від кількості залучених розробників. Також кількість пристроїв може змінюватися з часом.

При самостійному розміщенні та підтриманні інфраструктури CI/CD-процесу, потрібно знайти баланс між достатніми потужностями, що дозволяють в момент високого попиту одночасно запускати кілька пайплайнів, і витратами на покупку та підтримку пристроїв, які більшу частину часу простоюватимуть. І в цьому випадку хмарна інфраструктура може запропонувати низку переваг.

Модель Infrastructure as a Service (IaaS) передбачає, що обчислювальні ресурси надаються через віртуальні машини чи контейнери. В разі потребування більшої кількості потужностей, необхідно просто запитати їх, і в цьому випадку не потрібно дбати ні про постачання, ні встановлення та адміністрування, ні про обладнання [22].

Стаючи споживачем хмарної інфраструктури, організація не знатиме, на яких пристроях розміщені віртуальні машини і контейнери, які вона використовує (відомо може бути лише їх територіальне розташування — в нормативних цілях і на випадок аварійного відновлення).

Контейнери реалізують принципи IaC (Infrastructure as Code) та дозволяють ще ефективніше використовувати хмарну інфраструктуру. Контейнер надає програмі всі залежності, необхідні для виконання, тому більше не виникає потреби шукати відмінності в конфігураціях.

Використовуючи контейнери в CI/CD-конвеєрі, спрощується процес розгортання свіжої збірки на різних стадіях. Артефактом збірки виступає образ контейнера. Готуючись до розгортання в кінцевому середовищі, можна

розгорнути його на кожному тестовому оточенні, дотримуючись повної консистентності.

Подібно до віртуальних машин, контейнери дозволяють запускати кілька додатків на одному сервері, при цьому забезпечуючи їм повну ізоляцію один від одного. Однак, на відміну від віртуальних машин, контейнерам не потрібна операційна система, завдяки чому вони важать менше, а також вони вимагають постійного закріплення для них серверних ресурсів. У результаті той самий пристрій може вмістити набагато більше контейнерів, ніж віртуальних машин, тому контейнери ідеально підходять для швидкого розгортання додатків у хмарній інфраструктурі.

Використовуючи контейнери, сама програма, залежності середовища та деталі конфігурації упаковуються в один артефакт, який можна розгорнути на будь-якому пристрої, що надає середовище запуску контейнерів.

Програма може працювати в кількох контейнерах (як у випадку мікросервісної архітектури). Ці контейнери повинні бути розгорнуті на тому самому пристрої або на кластері мережі пристроїв. Інструменти управління контейнерами, такі як Kubernetes, розроблялися для того, щоб полегшити роботу з великою кількістю контейнерів за рахунок автоматизації таких завдань, як розгортання, адміністрування та масштабування [23].

При розміщенні CI/CD-конвеєрі у хмарі контейнери дозволяють ефективно використовувати обчислювальні ресурси та застосовувати інструменти автоматизації. Коли попит зростає, можна збільшувати потужність, а коли знижується попит — вимикати контейнери, звільняти всю задіяну інфраструктуру і таким чином скорочувати витрати.

На додаток до IaaS деякі хмарні провайдери тепер також пропонують CaaS (Containers as a Service) — підхід, який дозволяє організаціям робити розгортання всередині контейнерів без необхідності оркестрування та налаштування кластера.

При розробленні програмного забезпечення для хмари та використання мікросервісів, контейнерів та інших хмарних технологій, автоматизувати

процес за допомогою CI/CD-ковеєру, який використовує контейнери, буде досить просто. З іншого боку, в разі використання монолітної архітектури, запакувати додаток у контейнер може бути досить складно.

У хмарному пайплайні зовсім не обов'язково використовувати контейнери — можна обійтися віртуальними машинами, які розміщені в інфраструктурі хмарного провайдера. За допомогою них, можна точно так само запускати збірку та забезпечувати консистентні оточення для тестування. Однак віртуальні машини вимагають більше ресурсів, ніж контейнери, і доведеться окремо настроювати середовище. Зважаючи на це, використання контейнеризації в хмарному середовищі є переважаючим фактором, який дозволить забезпечити ізольованість та портативність додатку, легке налаштування оточення та знизить витрати на інфраструктуру [24].

1.8 Постановка задачі дослідження

Завданням даної роботи є створення компоненту, який являє собою аналізатор програмного коду, що включає тестування на вразливості, його інтеграція в CI/CD-процес та подальше розгортання протестованого програмного додатку в ізольованому хмарному середовищі.

Основні етапи роботи компоненту тестування безпеки є такими:

— проведення тестування безпеки програмного продукту;

— створення звіту на вкладці аналізатору;

— репортування на поштову скриню у разі виявлення вразливостей;

— зупинка розгортання артефакту на середовищі;

Основні етапи CI/CD-процесу є такими:

— відправка змін в коді додатку на віддалений репозиторій;

— завантаження проекту з репозиторію;

— початок аналізу коду на вразливості;

— формування звіту з результатами тестування;

— збірка проекту;

— завантаження артефактів збірки в репозиторій для контейнерів;

— завантаження образу контейнера на сервер;

— запуск контейнера з додатком.

Сформуємо основні вимоги до програмного компоненту тестування безпеки:

— програмний компонент має бути незалежним від обраної платформи, доменної області програми, мови написання коду;

— у разі потреби сканер, що буде використано, може бути змінено на більш вдосконалений, кількість перевірок може бути змінена;

— компонент повинен мати можливість проводити аналіз незалежно від статичного аналізатора коду, звичайних юніт та автотестів;

— формат звіту може бути змінено в залежності від потреб проекту;

— кількість невдалих тестів, через яких збірка буде вважатись проваленою, може бути зконфігурована за потребами команди.

Отже, аналіз показав, що існує багато прикладів вразливостей в системі безпеки продукту. Навіть після проведення повного циклу тестування не можна бути на 100% впевненими, що система повністю безпечна. Однак з високою ймовірністю можна сказати, що процент несанкціонованих проникнень, крадіжок інформації та втрат даних буде набагато менший. Автоматизація та введення тестування безпеки до процесів створення артефакту та розгортання коду в потрібному середовищі знижує цей процент до мінімуму. Тому, головною метою магістерської кваліфікаційної роботи є вдосконалення процесу доставки програмного продукту до кінцевого користувача, а також можливість дати гарантію захисту персональної інформації користувача.

2 АНАЛІЗ ТА ВИБІР ІНСТРУМЕНТІВ ДЛЯ РОЗРОБЛЕННЯ ТА КОНТЕЙНЕРИЗАЦІЇ ПРОГРАМНОГО ЗАСОБУ

2.1 Аналіз вимог до програмного засобу

Програмний засіб, що розроблюється, призначений для інтеграції системи безперервного розгортання та секьюріті-сканеру у рамках CI/CD процесу, з подальшою контейнеризацію.

Головними вимогами до розробки є:

— програмний компонент повинен відстежувати секьюріті-вразливості, які будуть зазначені у налаштуваннях секьюріті-сканера;

— сканування проводиться після створення мерж-реквесту (запиту на злиття локальних змін з кодом додатку на віддаленому репозиторії), коли почнеться збірка артефакту;

— програмний компонент має можливість працювати разом з автотестами та юніт тестами;

— програмний компонент повинен мати можливість поєднувати у собі динамічне секюріті тестування та статичний аналізатор коду на можливі вразливості;

— у разі знаходження вразливостей, збірка має вважатись проваленою, а розгортання коду не повинно розпочатися;

— у разі знаходження вразливості, має бути згенеровано звіт, який містить у собі опис знайденої проблеми, також звіт має бути відправлено на поштову скриню;

— у разі успішного проходження секьюріті-тестування, готовий артефакт має бути розгорнуто в ізольованому середовищі (контейнері) на сервері.

— під час розгортання додатку повинен створюватись додатковий (резервний) сервер для забезпечення безперебійної роботи додатку.

— у разі високого навантаження повинні створюватись додаткові сервери, та реалізація балансування навантаження.

2.2 Розробка методу тестування безпеки програмного продукту

Розроблюваний програмний компонент використовується для тестування безпеки продукту. Його можна використовувати як для динамічного, так і для статичного аналізу коду.

Головними функціями є:

— перевірка коду за системою OWASP;

— генерування звіту;

— відправлення звіту на поштову скриню розробника;

— коментар на вразливій ділянці коду про виявлену проблему.

На рисунку 2.1 подано блок-схему алгоритму процесу інтеграції програмного коду.

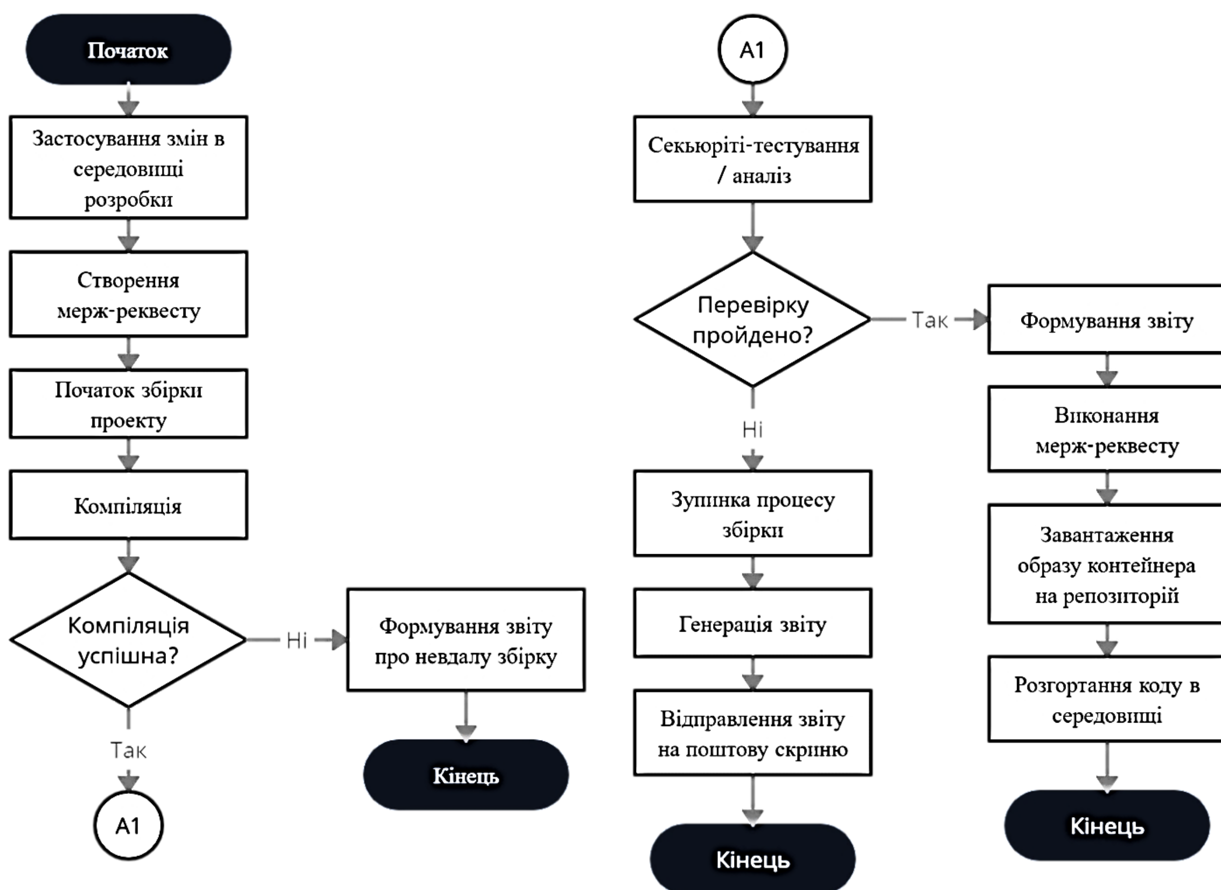


Рисунок 2.1 — Блок-схема алгоритму процесу інтеграції програмного коду

Система безпечної контейнеризації веб-додатку забезпечить безперервну доставку контенту з подальшим розгортанням в хмарному контейнерному середовищі.

Після внесення змін на віддалений репозиторій за допомогою системи контролю версій, буде запущено конвеєр безперервної інтеграції та розгортання, після чого готовий додаток буде розгорнуто в контейнері на сервері. Структурну схему процесу розгортання веб-додатку наведено на рисунку 2.2.

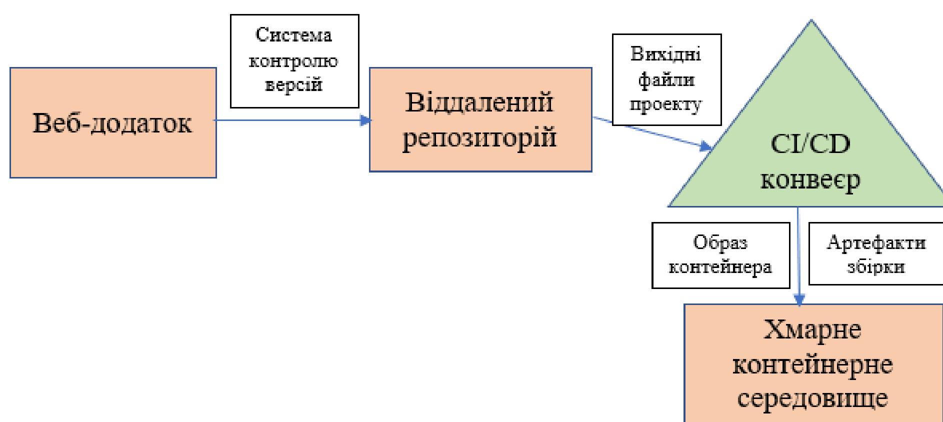


Рисунок 2.2 — Структурна схема процесу розгортання програмного компоненту

В наступних підрозділах буде проводитись аналіз та вибір засобів реалізації програмного компоненту безпечної контейнеризації для досягнення поставлених задач.

2.2.1 Основні компоненти розроблюваного програмного засобу

Програмний засіб виконує автоматичне збереження програмного додатку та внесених в код змін на хмарному репозиторії, тестування на наявність вразливостей, контейнеризацію та доставку до кінцевого середовища розгортання.

Визначимо основні компоненти програмного засобу.

Першим компонентом є Web-застосунок, що було створено для перевірки роботи програмного компоненту.

Другим компонентом є система управління версіями та репозиторій. Програмний засіб, що було обрано для розробки, інтегрований з системою контролю версій та хмарним репозиторієм. Таким чином, після створення коміту (фіксація змін в програмному коді) з написаним кодом його буде відправлено на віддалений хмарний репозиторій.

Третім компонентом є сервер інтеграції для безперервного доставлення коду до кінцевого середовища розгортання. Таким чином, після потрапляння коду на репозиторій, почнеться формування артефакту, його тестування та розгортання на оточенні, у разі провалення тестування, збірка буде вважатись не пройденою.

Наступним є компонент проведення секьюріті-тестування. Після того, як стартує збірка, вона проходить статичне або/та динамічне тестування. Таким чином, у разі, якщо тестування буде пройдене, і помилок та вразливостей виявлено не буде, збірку буде розгорнуто в середовищі. Якщо ж були виявлені вразливості, збірка артефакту буде вважатися проваленою, на поштову скриню буде відправлено звіт, який буде містити у собі опис знайдених вразливостей, також звіт буде збережено на вкладці тестування.

Останнім компонентом є програмна платформа для контейнеризації додатку. Всі компоненти та залежності, необхідні для функціонування коду, будуть поміщені в контейнер для легкого розгортання та забезпечення ізольованості середовища виконання. Таким чином, у разі успішного тестування, готовий артефакт буде розгорнуто в контейнері у хмарному середовищі.

2.2.2 Вибір програмного забезпечення для безперервної інтеграції коду

Існує багато програмних рішень для безперервної інтеграції: Jenkins, Buddy, JetBrains TeamCity, Buildbot, Codeship, Hudson та інші.

Jenkins — програмна система з відкритим вихідним кодом Java, призначена для забезпечення процесу безперервної інтеграції ПЗ.

Система Buddy дозволяє створювати, тестувати і розгортати веб-проекти простим способом.

Travis CI — служба непрерівной інтеграції для проектів з відкритим вихідним кодом. Він підтримує Ruby, PHP, Python, Java, Node.js і багато інших.

JetBrains TeamCity — це система безперервної інтеграції та управління проектуванням. З допомогою TeamCity можна налаштувати серверну збірку за лічені хвилини з безперервним модульним тестуванням, аналізом якості повідомлень і звітністю про проблему збірки.

Buildbot — це середовище з відкритим вихідним кодом для автоматизації процесів складання, тестування та випуску програмного забезпечення.

Codeship — це служба безперервної доставки, яка інтегрується з GitHub і Bitbucket.

Hudson контролює виконання повторних завдань, таких як створення програмного проекту або задач, що виконуються Cron.

Semaphore — це служба інтеграції та розгортання, яка використовується для розгортання та тестування відкритих та приватних проектів.

Для безперервної інтеграції коду було обрано Jenkins, так як на сьогоднішній день це найпопулярніша система для забезпечення процесу безперервної інтеграції ПЗ (Рисунок 2.3), її легко налаштовувати, вона має широкий вибір необхідних плагінів.

Jenkins дозволяє автоматизувати частину процесу розробки програмного забезпечення, в якому не обов'язкова участь людини, забезпечуючи функції безперервної інтеграції. Він підтримує інструменти системи управління версіями, включаючи AccuRev, CVS, Subversion, Git, Mercurial, Perforce, Clearcase і RTC [25]. Може збирати проекти з використанням Apache Ant і Apache Maven, а також виконувати довільні сценарії оболонки і пакетні файли Windows. Збірка може бути запущена різними способами, наприклад, за подією

фіксації змін в системі управління версіями, за розкладом, по запиту на певний URL, після завершення другої збірки в черзі.

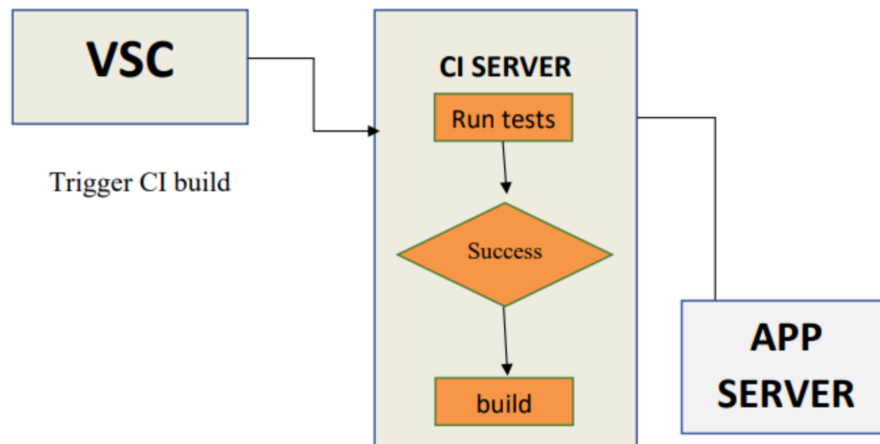


Рисунок 2.3 — Структурна схема процесу безперервної інтеграції

Контроль доступу реалізується двома способами: перевіркою достовірності користувача і авторизацією. Підтримується захист від зовнішніх загроз, в тому числі від CSRF-атак і шкідливих збірок.

2.2.3 Вибір програмного забезпечення для проведення тестування безпеки додатку

Плагіни IDE та Linters — легкі інструменти статичного аналізу, які виявляють такі дефекти, як помилки кодування, погані методи кодування та вразливі місця під час написання коду розробником. Вони заощаджують час, надаючи інформаційні виправлення під час побудови програми і зводячи до мінімуму спроби перегляду коду на більш пізніх етапах розробки. Одна з проблем полягає в тому, що їхнє використання на основі команд є складним.

Приклади інструментів:

— плагіни Veracode для Eclipse, Intelli-J, Visual Studio;

— OWASP ASIDE для Eclipse (з відкритим вихідним кодом);

— Synopsys SecureAssist для Eclipse, Visual Studio;

— FindSecBugs для Eclipse, Intelli-J, Netbeans (з відкритим вихідним кодом).

Експертний огляд коду — покращення якості коду за допомогою парного програмування, неформальних покрокових інструкцій або формальних перевірок.

Оскільки не кожний рядок коду може бути переглянутий, можна сформувати «контрольний список безпеки» у команді або організації, для забезпечення простих тестів та перевірок. Ці перевірки можуть виконуватися вручну навіть з дуже базовими знаннями щодо безпеки додатків і забезпечувати уникнення вразливостей під час процесу розробки. Наприклад, OWASP Cheat Sheet Series надає стислі рекомендації для різних категорій дизайну і тестування.

Приклади інструментів:

— Gerrit (з відкритим вихідним кодом);

— Phabricator (з відкритим вихідним кодом);

— Atlassian Crucible;

— SmartBear;

— Codacy.

Статичний та динамічний аналіз коду забезпечується інструментами якості коду, інтегрованими в програми CI, такі як Jenkins, Travis CI або CircleCI. Статичний і динамічний аналіз коду є звичним явищем у сучасному CI/CD конвеєрі і економить час завдяки автоматизованому перегляду коду в таких сферах, як стиль, найкращі практики, сумісність і безпека.

Тестування, проведене під час процесу збірки, має виконуватися швидко. Для будь-яких довготривалих служб безпеки необхідно запланувати виконання так само, як в деяких організаціях заплановано тестування інтеграції (асинхронне тестування).

Усі наведені нижче інструменти інтегруються з Jenkins та/або іншими серверами збирання [25].

Приклади інструментів:

— Snyk;

— SonarQube з ввімкненими правилами безпеки (з відкритим вихідним кодом);

— OWASP ZAP (з відкритим вихідним кодом);

— Minion (з відкритим вихідним кодом);

— Rogue Wave Software Klocwork;

— Codacy;

— Synopsys Coverity;

— IBM AppScan;

— Veracode Scanner.

Як інструмент проведення секюріті-тестування було обрано SonarQube.

SonarQube — це платформа з відкритим вихідним кодом, розроблена SonarSource, для безперервної оцінки якості коду шляхом статичного аналізу. Завершивши сканування коду, SonarQube формує звіт, який можна переглянути в GUI через браузер. Всі виявлені проблеми є «інтерактивними білетами», що дозволяють писати до них коментарі, делегувати їх іншим користувачам, відкривати або закривати тощо.

SonarQube надає систематизований звіт про якість коду, безпеки та загальний Quality Gate Status, тобто логічне значення, що свідчить про готовність програми для відправки в середовище розгортання [27]. Він дозволяє іншим учасникам проекту спільно працювати над станом якості. Проблеми можна обговорювати та делегувати, що дуже сприяє співпраці, і саме у можливості спільної роботи полягає найбільша перевага SonarQube.

2.2.4 Вибір системи управління версіями та репозиторію

Найпопулярнішими сучасними системами управління версіями є Gerrit, Git, Phabricator та інші.

Gerrit — вільне програмне забезпечення з веб-інтерфейсом, інтегроване з розподіленою системою контролю версій Git, призначене для спільного проведення інспекції вихідного коду. Спочатку Gerrit являв собою набір патчів

для Rietveld, але пізніше перетворився в повноцінний проект. Gerrit використовує Google Web Toolkit для генерації клієнтського JavaScript-коду.

Git — розподілена система управління версіями. Проект був створений для управління використанням ядра Linux. Серед проектів, які використовують Git — ядро Linux, Swift, Android, Drupal, Cairo, GNU Core Utilities, Mesa, Wine, Chromium, Compiz Fusion, FlightGear, jQuery, PHP, NASM, MediaWiki, ряд дистрибутивів Linux.

Phabricator — набір взаємодіючих веб-інструментів для спільної ефективної розробки програмного забезпечення, який був розроблений як внутрішній інструмент для Facebook. Phabricator є вільним програмним забезпеченням з ліцензією Apache v2.

Як систему управління версій для проекту було обрано Git, так як він є найпопулярнішою безкоштовною системою, є простим у використанні, та його легко інтегрувати з такими системами, як Jenkins, Sonar, SpotBugs і т.п.

Використання Git надає можливість інтеграції з AWS CodeCommit — повністю керованим сервісом управління вихідним кодом, який дозволяє легко розміщувати безпечні та високомасштабовані приватні репозиторії Git. Перевагою розміщення вихідного коду в CodeCommit є повна керованість, безпека, висока доступність і розширеність сервісу за рахунок уже існуючих інструментів.

2.2.5 Вибір засобу контейнеризації додатку

Контейнери дозволяють заморозити та перезапустити точну копію системи, яку планується розгорнути, включаючи операційну систему та файли конфігурації. Це спрощує налагодження та миттєве тестування, і навіть змінює спосіб розгортання додатку.

Пакети-контейнери не тільки повні, але й достатньо малі та ефективні, щоб завантажитись та запуститись за лічені секунди. Менеджери кластерів забезпечують балансування навантаження та масштабування для забезпечення безвідмовної роботи навіть під час розгортання.

Зростання та розширення технології контейнерів дає широкий вибір, зокрема, які стандарти використовувати, як зберігати старі версії та розгортати нові образи, а також як керувати контейнерами у середовищі розгортання.

Основані інструменти контейнеризації додатків:

Docker — перша і найпопулярніша контейнерна технологія з відкритим вихідним кодом, що працює з більшістю продуктів, а також з багатьма інструментами з відкритим кодом.

CRI-O — перша реалізація Container Runtime Interface, є неймовірно легкою еталонною реалізацією з відкритим кодом.

Rktlet — перероблений і оновлений для використання CRI, який має набір підтримуваних інструментів.

Containerd — проект Фонду Хмарних Обчислень (Cloud Native Computing Foundation), був раннім форматом контейнера. Нещодавно розробники Containerd створили плагін CRI, який дозволяє Kubernetes запускати контейнер так само, як він запускає Rktlet або CRI-O.

Microsoft Containers — контейнери Microsoft, позиціоновані як альтернатива Linux, можуть також підтримувати контейнери Windows. Зазвичай вони працюють у справжній віртуальній машині, а не в менеджері кластерів, як Kubernetes.

Для реалізації контейнеризації проекту було обрано Docker, так як сьогодні це є найпопулярнішою технологією свого роду, яка забезпечує повну ізоляцію додатку на рівні файлової системи.

2.3 Розробка структурної схеми CI/CD-ковеєра безперервної інтеграції та розгортання

Використання хмарних сервісів Amazon Web Services (AWS) при розробці системи допоможе автоматизувати етапи процесу доставки програмного забезпечення, наприклад, автоматичне створення збірки та її розгортання на екземплярах Amazon EC2. Сервіс AWS CodePipeline допоможе виконати компонування, тестування та розгортання програмного коду при

кожному внесенні в нього змін на базі визначеної моделі процесу випуску ПЗ, а також оркестрування кожного етапу процесу випуску. У процесі налаштування конвеєр доставки програмного забезпечення буде доповнено іншими сервісами AWS шляхом їх підключення до сервісу CodePipeline.

Конвеєр безперервної інтеграції та розгортання веб-додатку буде функціонувати таким чином:

— розробник завантажує зміни в коді на віддалений репозиторій AWS CodeCommit за допомогою системи контролю версій Git, що спричиняє початок виконання конвеєру AWS CodePipeline;

— AWS CodeCommit формує артефакти з вихідного коду репозиторію, Jenkins за допомогою артефактів починає збірку проекту;

— SonarQube сканер аналізує процес збірки та відправляє на SonarQube сервер для формування загального звіту, що включає перевірку на безпеку, та повертає результат проходження перевірки на Jenkins;

— якщо перевірка успішна, Jenkins формує образ контейнера та відправляє його на сховище контейнерів Amazon ECR;

— Jenkins сигналізує про успішне завершення збірки та передає виконання на наступну стадію AWS CodePipeline;

— на стадії розгортання створюється запит на Amazon ECS для оновлення образу контейнера;

— створюються нові завдання (Task) відповідно до образу контейнера з Amazon ECR;

— створюються нові екземпляри Amazon EC2 відповідно до лімітів групи автоматичного масштабування AutoScaling Group та кількості завдань;

— Docker контейнери запускаються на екземплярах Amazon EC2.

На рисунку 2.4 наведено структурну схему циклу CI/CD системи безпечної контейнеризації веб-додатку.

Отже, програмний компонент, що розробляється, включатиме секьюріті-сканер SonarQube, та передбачатиме контейнеризацію Docker. Засіб буде інтегровано з такими системами, як Jenkins, з метою створення повноцінного

процесу доставки коду до середовища розгортання. У майбутньому компонент може бути поєднано з баг-трекінговою системою Jira тощо.

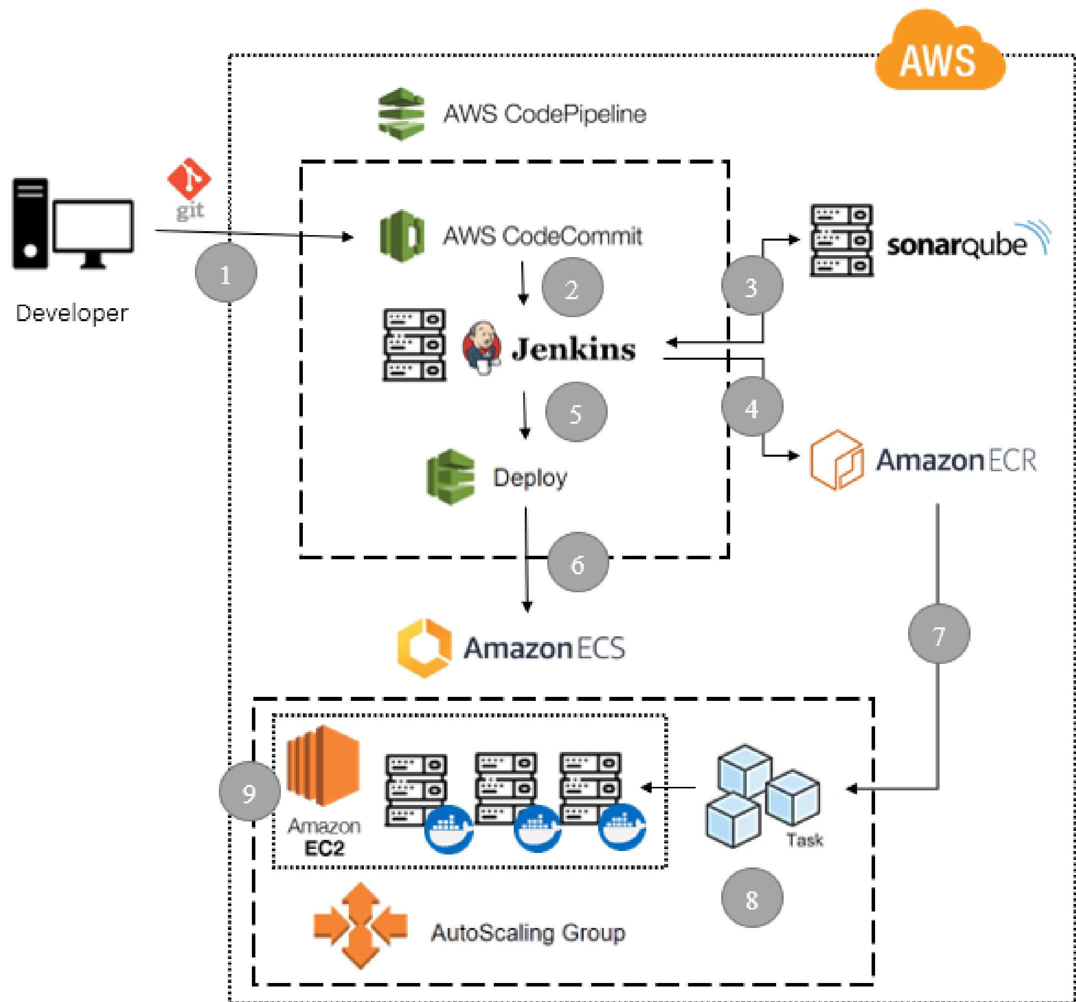


Рисунок 2.4 — Структурна схема CI/CD-ковеєру системи безпечної контейнеризації веб-додатку

3 РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ БЕЗПЕЧНОЇ КОНТЕЙНЕРИЗАЦІЇ ВЕБ-ДОДАТКУ

3.1 Реалізація тестового проекту

Для проведення тестування програмного компоненту було створено веб-застосунок на фреймворку ASP.NET Core у який були додані вразливості безпеки. Тестовий проект було написано об'єктно-орієнтованою мовою програмування C#. Для нього була розроблена примітивна база даних.

C# — це об'єктно-орієнтована мова програмування, створена Microsoft, яка працює на .NET Framework. C# належить до сімейства C, і ця мова близька до інших популярних мов, таких як C++ і Java. Як і інші мови програмування загального призначення, C# можна використовувати для створення низки різних програм і додатків: мобільних додатків, настільних додатків, хмарних служб, веб-сайтів, корпоративного програмного забезпечення та ігор.

Додаток було створено з використанням фреймворку Blazor. Blazor — це UI-фреймворк для створення інтерактивних програм, які можуть працювати як на стороні сервера, так і на стороні клієнта, на платформі .NET. На цей фреймворк вплинули сучасні фреймворки для створення клієнтських додатків — Angular, React, VueJS. Зокрема, це проявляється в ролі компонентів при побудові інтерфейсу користувача. У той же час і на стороні клієнта, і на стороні сервера при визначенні коду як мова програмування використовується C#, замість JavaScript, а для опису візуального інтерфейсу використовуються стандартні HTML і CSS [28].

У Blazor Server основна логіка програми розташовується на стороні сервера. Якщо на стороні клієнта відбуваються якісь події, то за допомогою SignalR клієнт надсилає серверу інформацію про дії, що відбулися. Сервер отримує цю інформацію, обробляє її та посилає клієнту відповідь. Оновлення елементів інтерфейсу користувача, обробка подій, виклики JavaScript на клієнтській стороні здійснюються за допомогою взаємодії сервера і клієнта через SignalR.

При створенні додатку WebAssemply Blazor в середовищі програмування Visual Studio 2022, створюється структура проекту, схожу на структуру проекту ASP.NET Core (Рисунок 3.1):

— папка wwwroot для зберігання статичних файлів, за замовчуванням зберігає файли css, зокрема, файли фреймворку bootstrap;

— папка Data зберігає класи C#, які описують дані, що використовуються;

— папка Pages містить сторінки Razor Pages, що визначають візуальну частину програми та її логіку, а також компоненти Razor, які представляють основний зміст сторінки;

— папка Shared зберігає додаткові компоненти Razor;

— файл _Imports.razor містить підключення просторів імен за допомогою директиви using, які будуть підключатися до компонентів Razor (файли з розширенням .razor);

— файл App.razor містить визначення кореневого компонента програми, який дозволяє встановити маршрутизацію між вкладеними компонентами за допомогою іншого вбудованого компонента Router;

— файл appsettings.json зберігає конфігурацію програми;

— файл Program.cs містить клас Program, який представляє точку входу до програми, в даному випадку це стандартний для програми ASP.NET Core клас Program, який запускає та конфігурує хост, в рамках якого розгортається програма з Blazor.

Веб-сайт являє собою простий каталог продуктів, тому структура бази даних містить таблиці Products (містить дані про продукти, такі як ім’я, категорії, фото тощо) та Categories (містить дані про категорії), як видно з вмісту файлу StoreDbContext.cs:

```
public class StoreDbContext : DbContext
{
    public DbSet<ProductModel> Products { get; set; }
            public DbSet<ProductType> Categories { get; set; }
```

…

}

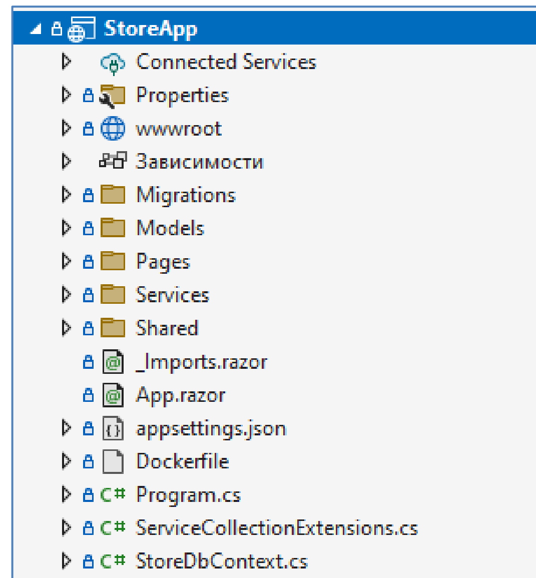


Рисунок 3.1 — Структура проекту Blazor WebAssemply App

Сторінка Razor _Host.cshtml, яка знаходиться в папці Pages, є кореневою сторінкою всієї програми. Коли до будь-якої сторінки (компонента) надходить запит, то саме ця сторінка повертається у відповідь. А окремі компоненти розміщуються всередині цієї сторінки. _Host.cshtml являє собою типову сторінку Razor Page, яка містить код C# та HTML.

Файл appsettings.json — це файл конфігурації програми, який використовується для зберігання налаштувань конфігурації, таких як рядки підключення до бази даних, будь-які глобальні змінні області дії програми тощо. В проекті даний файл містить налаштування, що включають дані про віддалену базу даних в середовищі AWS (хост, ім'я бази, ім'я користувача, пароль):

```
{
  "AllowedHosts": "*",
  "ConnectionStrings": {
```

"DefaultConnection": "Server=database-1.****yd3.us-east-1.rds.amazonaws.com;Database=database-1;User Id=liuda;Password=****011;"

}

Веб-додаток було розроблено з демонстраційною метою для подальшого тестування та розгортання, тому його функціональність обмежується веб-каталогом продуктів з простою базою даних. Веб-каталог передбачає такі функції як додавання категорії для продуктів з вкладки Categories Management (Рисунок 3.2) та додавання самого продукту з вкладки Products Management (Рисунок 3.3).

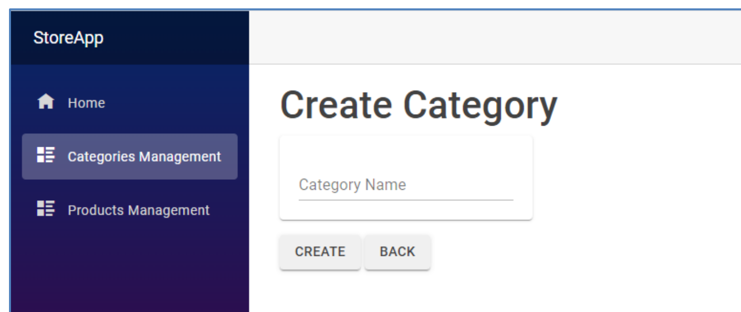


Рисунок 3.2 — Додавання категорії до веб-каталогу

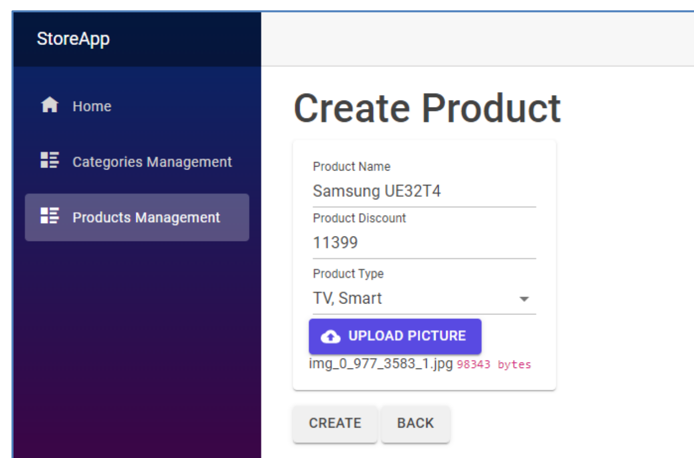


Рисунок 3.3 — Додавання продукту до веб-каталогу

Додані категорії можна переглянути на відповідній вкладці додатку, також головна сторінка додатку (Home) відображає всі додані продукти (Рисунок 3.4).

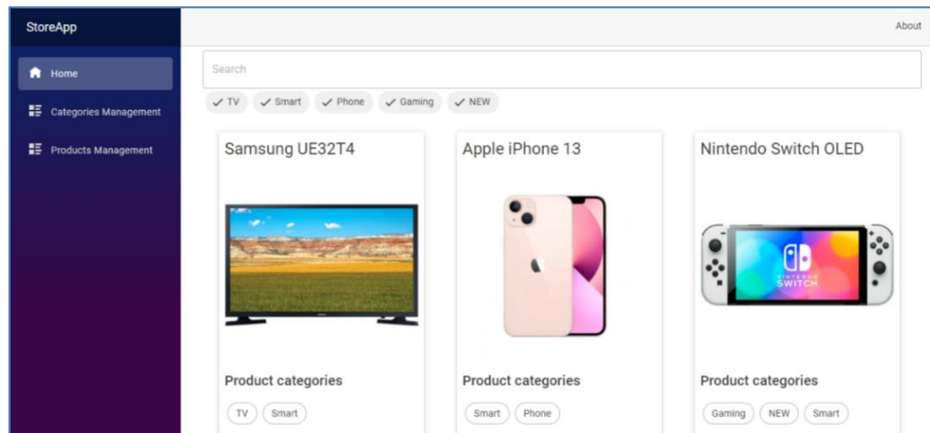


Рисунок 3.4 — Головна сторінка додатку

Для подальшого тестування було обрано саме веб-застосунок тому, що на сьогоднішній день кількість веб-програм, що розроблюються, росте з кожним днем, а знаходження вразливостей є глобальною проблемою для всіх комерційних продуктів, починаючи від проблем захисту даних користувачів, інформації з їх платіжних систем до витоку конфіденційної інформації, коду, структури бази даних.

3.2 Налаштування хмарного репозиторію AWS CodeCommit і його інтеграція з Git

Після внесення змін в код додатку, розробник відправляє зміни на віддалений репозиторій AWS CodeCommit за допомогою системи контролю версій Git.

AWS CodeCommit — це служба контролю версій, розміщена в Amazon Web Services, яку можна використовувати для приватного зберігання та керування активами (такими як документи, вихідний код і двійкові файли) у хмарі. Він підтримує стандартну функціональність Git, тому бездоганно працює з наявними інструментами на основі Git.

CodeCommit забезпечує високу доступність і довговічність послуг і усуває адміністративні накладні витрати на керування власним апаратним і програмним забезпеченням. Дана служба зберігає репозиторії поруч із іншими виробничими ресурсами в хмарі AWS, що допомагає збільшити швидкість і

частоту життєвого циклу розробки. Він інтегрований з IAM і може використовуватися з іншими службами AWS і паралельно з іншими репозиторіями.

CodeCommit надає консоль для легкого створення репозиторіїв і переліку існуючих репозиторіїв і гілок. З даним сервісом можна працювати за допомогою командного рядка на своїй локальній машині або використовувати графічний редактор.

На рисунку 3.5 показано, як можна використовувати локальну машину для розробки, консоль AWS CLI або CodeCommit і службу CodeCommit для створення сховищ і керування ними.

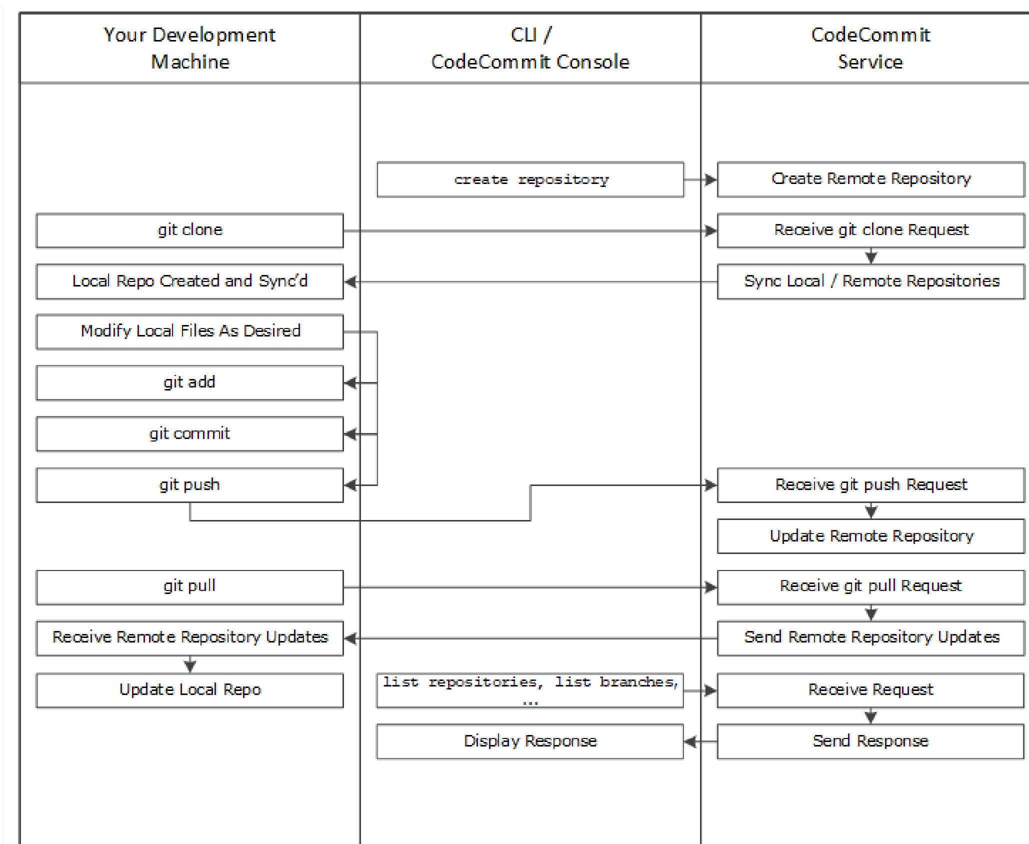


Рисунок 3.5 — Діаграма послідовностей операцій Git та їх виконання сервісом CodeCommit

При першому підключенні до репозиторію CodeCommit, потрібно клонувати його вміст на свою локальну машину. Також можна додавати та редагувати файли в репозиторії безпосередньо з консолі CodeCommit.

Перш ніж клонувати репозиторій CodeCommit або підключити локальний репозиторій до репозиторію CodeCommit необхідно виконати наступні налаштування:

— налаштувати свій локальний комп'ютер за допомогою програмного забезпечення та параметрів, необхідних для підключення до CodeCommit, що включає встановлення та налаштування Git;

— отримати URL-адресу клону сховища CodeCommit, до якого буде відбуватись підключення;

— щоб зберегти локальну копію репозиторію CodeCommit, до якого відбувається підключення, потрібно мати місце на локальній машині (ця локальна копія репозиторію CodeCommit відома як локальне сховище).

Потім необхідно перейти в необхідну директорію і запускати команди Git з цього розташування. Наприклад, можна використовувати шлях до каталогу /tmp (для Linux, macOS або Unix) або C:\temp (для Windows).

З обраного каталогу проекту необхідно використати Git, щоб запустити команду клонування:

git clone https://git-codecommit.us-east-1.amazonaws.com/v1/repos/StoreApp my-store-app

У наведеній команді git-codecommit.us-east-1.amazonaws.com є точкою з'єднання Git для Східного регіону США (Північна Вірджинія), де існує сховище. StoreApp — це назва сховища CodeCommit, а my-store-app — назва каталогу, який Git створює в каталозі проекту. Після того, як Git створить каталог, він витягне копію репозиторію CodeCommit у щойно створений каталог.

Після успішного підключення локального репозиторію до репозиторію CodeCommit можна запускати команди Git із локального репозиторію для створення комітів, гілок і тегів, а також надсилання та отримання з репозиторію CodeCommit.

Використовуємо команди git add та git commit для занесення змін до проекту та створення нового коміту (Рисунок 3.6).

```
user@WIN-56VCOIJROPL MINGW64 /d/Діплом/StoreApp (master)
$ git add .

user@WIN-56VCOIJROPL MINGW64 /d/Діплом/StoreApp (master)
$ git commit -m "This is my first commit"
[master 9b3eeea] This is my first commit
 19 files changed, 6 insertions(+), 9 deletions(-)
 rewrite .vs/StoreApp/DesignTimeBuild/.dtbcache.v2 (89%)
 delete mode 100644 .vs/StoreApp/FileContentIndex/3bf856c5-a204-434c-804d-a40f04f8bd90.vsidx
 delete mode 100644 .vs/StoreApp/FileContentIndex/4cadba3f-db0a-49d7-8033-6712f3bee08d.vsidx
 delete mode 100644 .vs/StoreApp/FileContentIndex/892bb3e1-c0ae-4b28-969e-758cf1ca92bd.vsidx
 delete mode 100644 .vs/StoreApp/FileContentIndex/f6805d5d-c713-406f-a137-5edea4440c0a.vsidx
 rewrite .vs/StoreApp/v17/.suo (76%)
 rewrite StoreApp/bin/Debug/net6.0/StoreApp.dll (78%)
 rewrite StoreApp/bin/Debug/net6.0/StoreApp.pdb (94%)
 rewrite StoreApp/obj/Debug/net6.0/StoreApp.dll (78%)
 rewrite StoreApp/obj/Debug/net6.0/StoreApp.pdb (94%)
 rewrite StoreApp/obj/Debug/net6.0/ref/StoreApp.dll (83%)
 rewrite StoreApp/obj/Debug/net6.0/refint/StoreApp.dll (83%)
 create mode 100644 StoreApp/wwwroot/images/Pink-1631803720-600x600.jpg
 create mode 100644 StoreApp/wwwroot/images/img_0_752_200_0_1_637909781915929904.jpg
 create mode 100644 StoreApp/wwwroot/images/img_0_977_3583_1.jpg
```

Рисунок 3.6 — Виконання команд git add та git commit в командному рядку Git

Вивантажуємо вміст локального репозиторію у віддалений репозиторій AWS CodeCommit за допомогою команди git push (Рисунок 3.7).

```
user@WIN-56VCOIJROPL MINGW64 /d/Діплом/StoreApp (master)
$ git push
Enumerating objects: 53, done.
Counting objects: 100% (53/53), done.
Delta compression using up to 4 threads
Compressing objects: 100% (21/21), done.
Writing objects: 100% (29/29), 292.68 KiB | 3.40 MiB/s, done.
Total 29 (delta 8), reused 0 (delta 0), pack-reused 0
To https://git-codecommit.us-east-1.amazonaws.com/v1/repos/StoreApp
   e2726c2..9b3eeea  master -> master
```

Рисунок 3.7 — Виконання команди git push

Після вивантаження змін, файли проекту будуть відображатись в репозиторії AWS CodeCommit, як показано на рисунку 3.8.

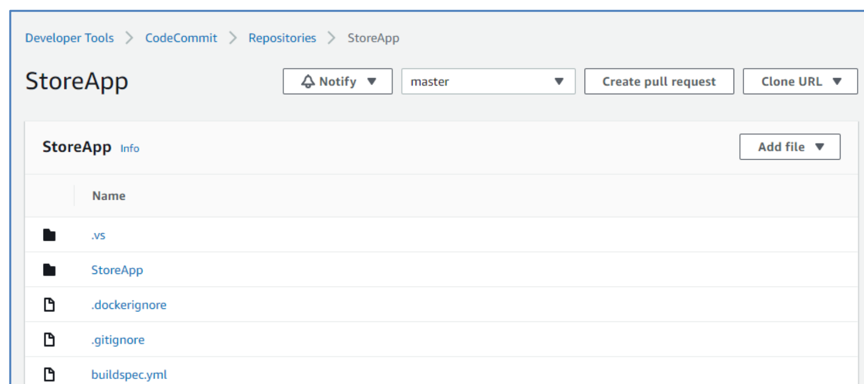


Рисунок 3.8 — Файли проекту на репозиторії AWS CodeCommit

Інтерфейс AWS CodeCommit також дозволяє зручно переглядати список комітів проекту (Рисунок 3.9) та створені гілки (git branch) (Рисунок 3.10).

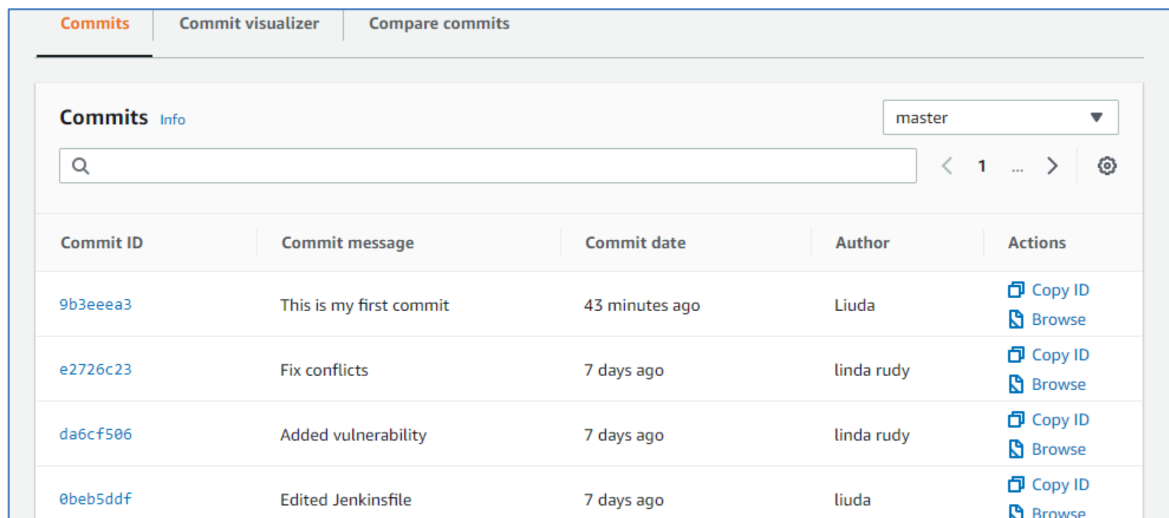


Рисунок 3.9 — Інтерфейс для перегляду списку комітів AWS CodeCommit

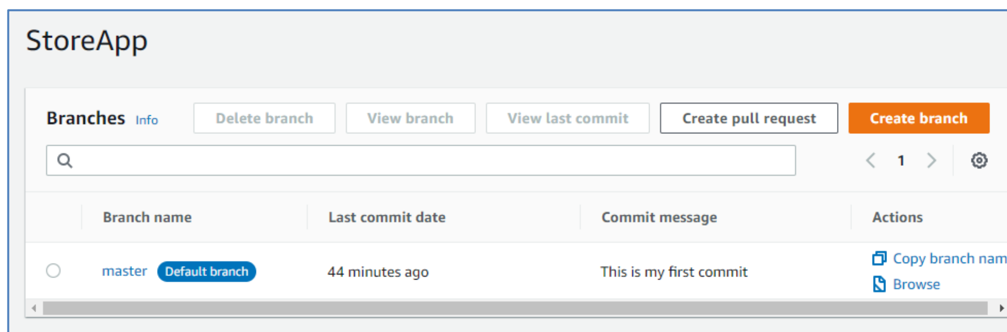


Рисунок 3.10 — Інтерфейс для перегляду гілок AWS CodeCommit

3.3 Налаштування серверу безперервної інтеграції Jenkins та конвеєру AWS CodePipeline

Jenkins за допомогою артефактів, сформованих з файлів репозиторію, починає збірку проекту та проводить секьюріті-тестування.

Jenkins здатний організувати послідовність дій, які допомагають автоматизувати процес безперервної інтеграції. Використовуючи Jenkins, можна прискорити процес розробки програмного забезпечення, оскільки Jenkins може автоматизувати збірку та тестування з високою швидкістю. Jenkins підтримує повний життєвий цикл розробки від збірки, тестування до документування, розгортання та інших етапів життєвого циклу розробки ПЗ.

Створюємо конвеєр автоматизованого випуску програмного забезпечення шляхом інтеграції Jenkins та AWS CodePipeline для тестування та розгортання коду після будь-яких змін. Готовий конвеєр зможе виявляти зміни, внесені до вихідного репозиторію, де зберігається зразок програми, а потім автоматично оновлювати робочий зразок програми. Безперервне розгортання дозволяє розгортати версії в робочому середовищі автоматично, без підтвердження розробника, що автоматизує весь процес випуску ПЗ.

Для інтеграції Jenkins з AWS CodePipeline необхідно інсталювали плагін CodePipeline для Jenkins, та плагін SonarQube для подальшого сканування безпеки. Також потрібно налаштувати виділеного користувача IAM для використання дозволів між проектом Jenkins і CodePipeline. Найпростіший спосіб інтегрувати Jenkins і CodePipeline — це встановити Jenkins на екземплярі EC2, який використовує роль IAM, яка була створена попередньо для інтеграції Jenkins та містить стандартну політику AWSCodePipeline_FullAccess (Рисунок 3.11).

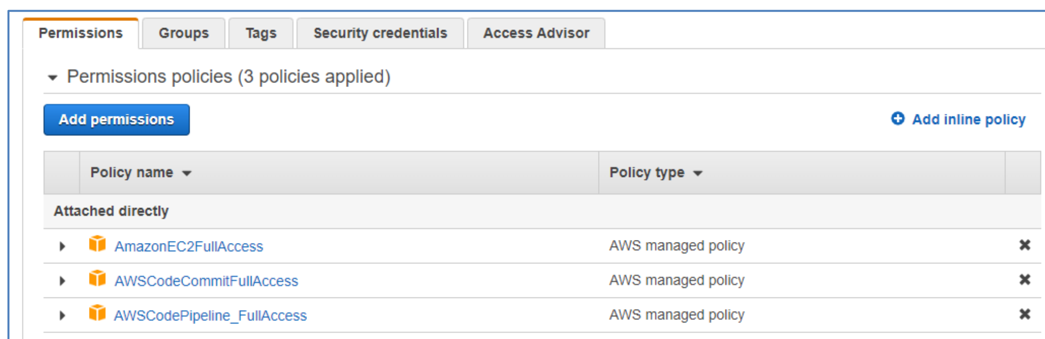


Рисунок 3.11 — Дозволи IAM користувача

Робочий процес розгортання починається з розміщення коду програми в репозиторії AWS CodeCommit. При створенні нового проекту в Jenkins, необхідно вказати тригери збірки (Build Triggers). Ця опція змушує Jenkins перевіряти налаштований репозиторій на наявність нових комітів/змін коду з указаною частотою. Для цього оберемо опцію Poll SCM та введемо в розкладі п’ять зірочок, розділених пробілами, як показано на рисунку 3.12, що відповідає щохвилинному запиту змін з системи контролю версій. Jenkins

перевіряє наявність нових змін на репозиторії кожну хвилину, та якщо зміни не виявлені, виходить із процедури.

Рисунок 3.12 — Тригери збірки в налаштуваннях проекту Jenkins

Додамо перший крок збірки — «SonarScanner для MSBuild — початок аналізу». Початковий крок виконується, коли додається аргумент командного рядка begin. Він підключається до конвеєра збірки, завантажує профілі та налаштування якості SonarQube і готує проект до аналізу. Можна переконатися, що код відповідає стандартам якості, адже SonarQube змушує збірку Jenkins зупинитись з помилкою, коли код не проходить поріг якості (Quality Gate).

За допомогою Jenkins можна призупинити виконання конвеєра, доки не буде відомий статус перевірки якості аналізу. Використаємо параметр аналізу sonar.qualitygate.wait=true у конфігурації. Якщо встановити для sonar.qualitygate.wait значення true, етап аналізу буде опитувати екземпляр SonarQube, доки не стане доступним статус Quality Gate. Це збільшує тривалість конвеєра та спричиняє збій кроку аналізу кожного разу, коли поріг якості дає збій, навіть якщо фактичний аналіз успішний.

Між початковим і кінцевим етапами потрібно створити проект, виконати тести та згенерувати дані про покриття коду, для цього вказуємо наступну команду в полі «Виконати команду Shell»:

dotnet build StoreApp/StoreApp.csproj

Кінцевий крок виконується, коли додається аргумент командного рядка "end". Він очищає перехоплювачі збірки MSBuild/dotnet, збирає дані аналізу, згенеровані збіркою, результати тестування, покриття коду, а потім завантажує все на SonarQube сервер.

В наступному кроці збірки здійснюємо вхід в Amazon ECR для подальшого завантаження образу контейнера:

echo Logging in to Amazon ECR...

aws --version

aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 892461827874.dkr.ecr.us-east-1.amazonaws.com

COMMIT_HASH=$(echo $CODEBUILD_RESOLVED_SOURCE_VERSION | cut -c 1-7)

Збираємо образ контейнера та додаємо для нього тег:

echo Build started on `date`

echo Building the Docker image...

cd StoreApp

docker build -t 892461827874.dkr.ecr.us-east-1.amazonaws.com/store-app:latest .

docker tag 892461827874.dkr.ecr.us-east-1.amazonaws.com/store-app:latest 892461827874.dkr.ecr.us-east-1.amazonaws.com/store-app:${COMMIT_HASH:=latest}

В останньому кроці збірки завантажуємо на віддалений репозиторій Amazon ECR образ контейнера:

echo Build completed on `date`

echo Pushing the Docker images...

docker push 892461827874.dkr.ecr.us-east-1.amazonaws.com/store-app:latest

docker push 892461827874.dkr.ecr.us-east-1.amazonaws.com/store-app:${COMMIT_HASH:=latest}

echo Writing image definitions file...

printf '[{"name":"store-app","imageUri":"%s"}]' "892461827874.dkr.ecr.us-east-1.amazonaws.com/store-app:latest" > imagedefinitions.json

Вказуємо персональну електронну адресу в полі «Сповіщення на пошту» в налаштуваннях проекту, що дозволить легко відслідковувати статус збірок (Рисунок 3.13).

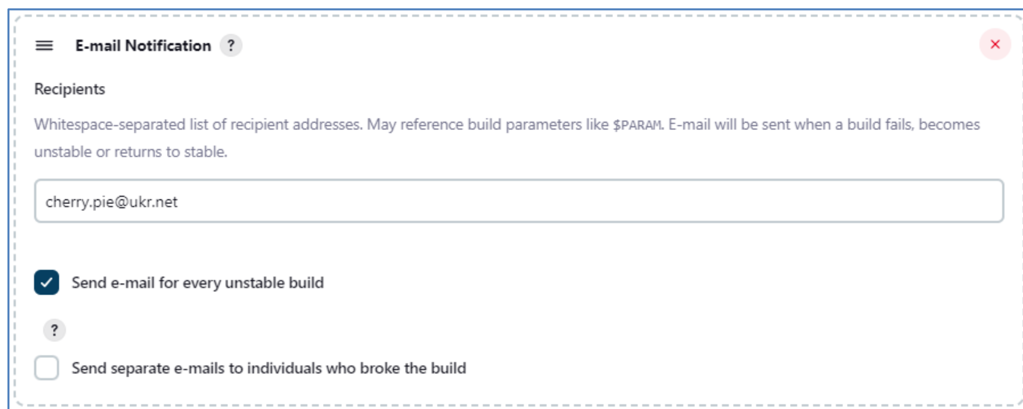


Рисунок 3.13 — Конфігурація сповіщень на пошту в Jenkins

Jenkins надішле повідомлення електронною поштою на цю адресу під час наступних подій:

— кожна провалена збірка;

— успішна збірка після провалених (або нестабільних);

— нестабільна збірка після стабільної (повідомляє про прояв регресії);

— кожна нестабільна збірка.

Приклад листа-сповіщення про успішну збірку наведено на рисунку 3.14.

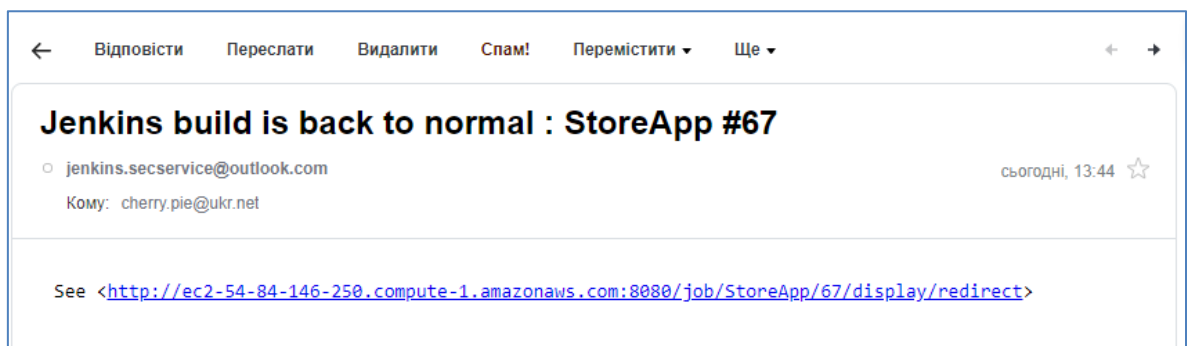


Рисунок 3.14 — Лист-сповіщення про успішну збірку від Jenkins

В конвеєрі AWS CodePipeline створюємо 3 стадії розгортання додатку.

На першій стадії — Source, CodePipeline виявляє зміни у програмному коді на репозиторії AWS CodeCommit та формує з нього вихідний артефакт. В якості режиму виявлення (Detection mode) обираємо Amazon CloudWatch Events, щоб автоматично запускати конвеєр, коли відбувається зміна у вихідному коді. Amazon CloudWatch Events надає потік системних подій майже в реальному часі, які описують зміни в ресурсах Amazon Web Services (AWS). CloudWatch Events дізнається про операційні зміни, коли вони відбуваються, реагує на ці робочі зміни та вживає коригувальні дії, якщо це необхідно, шляхом надсилання повідомлень у відповідь на середовище, активації функцій, внесення змін та збору інформації про стан. В результаті виконання даної стадії отримуємо вихідний артефакт — SourceArtifact.

На другій стадії — Build, запускається сервіс інтеграції Jenkins, який виконує збірку та делегує процес сканування коду. Як вхідний артефакт використаємо SourceArtifact. Вхідний артефакт дії має точно відповідати вихідному артефакту, оголошеному в попередній дії. Це справедливо для всіх дій, незалежно від того, чи знаходяться вони на одній стадії, чи на наступних стадіях, але вхідний артефакт не обов’язково має бути наступною дією в суворій послідовності від дії, яка забезпечила вихідний артефакт. Паралельні дії можуть оголошувати різні пакети вихідних артефактів, які, у свою чергу, використовуються різними наступними діями. Як вихідний артефакт даної стадії вказуємо ImageDefinition.

На останній стадії — Deploy, створюється запит на оновлення образу контейнера на Amazon ECS. Як вхідний артефакт використовується ImageDefinition, отриманий на попередній стадії. Також обираємо кластер та сервіс для розгортання, які вже були створені попередньо в Amazon ECS. Кластер Amazon ECS — це логічне групування завдань або послуг, а сервіс Amazon ECS підтримує та запускає екземпляри шаблонів завдань в кластері Amazon ECS.

Візуальний вигляд конвеєра в інтерфейсі AWS CodePipeline, що успішно завершив збірку, показано на рисунку 3.15.

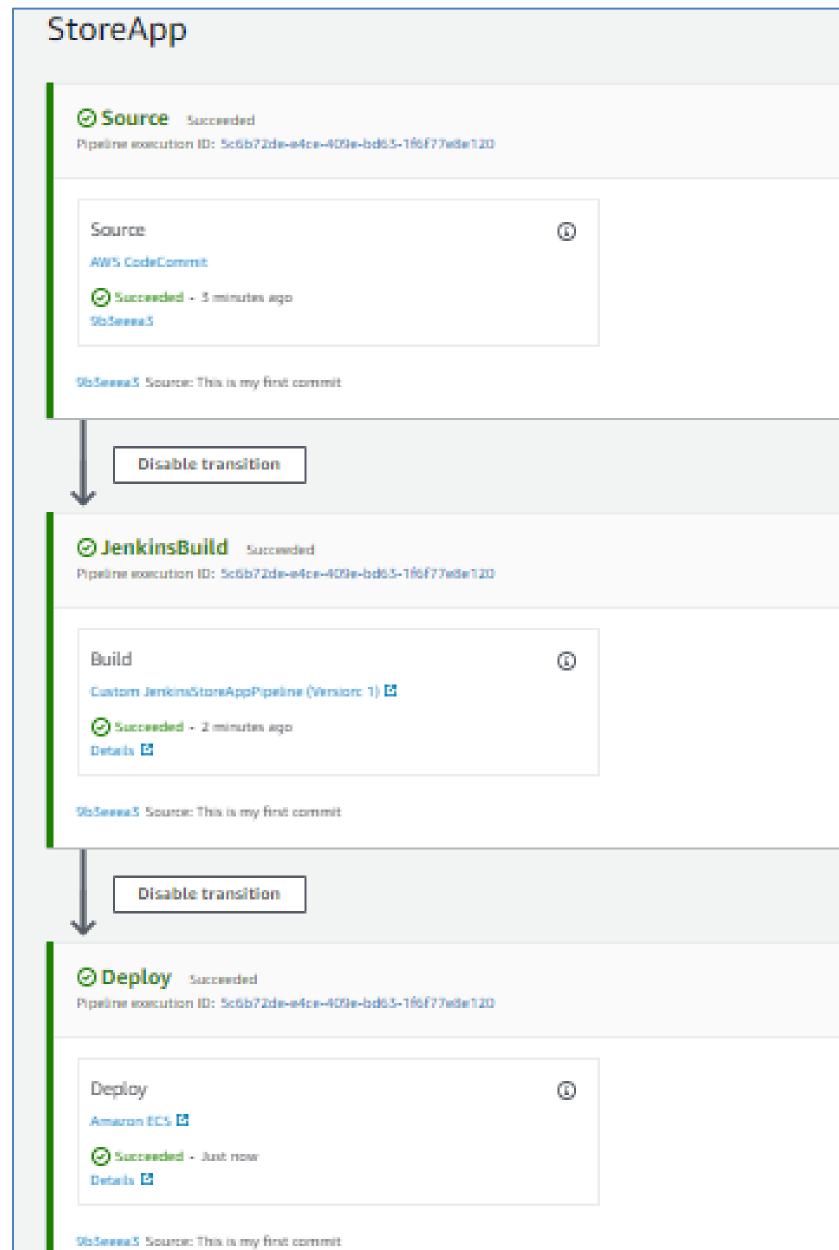


Рисунок 3.15 — Успішне завершення збірки конвеєром AWS CodePipeline

CodePipeline автоматично запускається та пропускає програмний код через конвеєр. На даному інтерфейсі можна переглядати перебіг роботи конвеєру, а також повідомлення про успіх і невдачу, коли конвеєр створює артефакт і делегує виконання завдань Jenkins та Amazon ECS.

3.4 Налаштування серверу для постійного аналізу та вимірювання якості коду SonarQube

SonarQube інтегрується в існуючий робочий процес щоб виконувати постійні перевірки коду проекту. Написання чистого коду є важливим для підтримки здорової кодової бази (Healthy code). Чистий код є таким, що відповідає певному визначеному стандарту, тобто є надійним, безпечним, придатним для обслуговування, читабельним і модульним, а також має інші ключові атрибути.

Екземпляр SonarQube складається з трьох компонентів, які подано на рисунку 3.16.

Першим компонентом є сервер SonarQube, який виконує такі процеси:

— веб-сервер, який обслуговує інтерфейс користувача SonarQube;

— пошуковий сервер на основі Elasticsearch;

— обчислювальний механізм, відповідальний за обробку звітів про аналіз коду та збереження їх у базі даних SonarQube.

Другим компонентом є база даних для зберігання таких даних:

— показники та проблеми з якістю та безпекою коду, виявлені під час сканування коду;

— конфігурація екземпляру SonarQube.

Третім компонентом є один або кілька сканерів, запущених на серверах збірки або постійної інтеграції для аналізу проектів.

Підхід Sonar «Clean as You Code» усуває багато вразливостей, які виникають під час перегляду коду на пізньому етапі процесу розробки. Даний підхід використовує поріг якості, щоб попереджати/інформувати, коли у новому коді (коді, який було додано чи змінено) потрібно щось виправити або переглянути, що дозволяє підтримувати високі стандарти та зосереджуватися на якості коду.

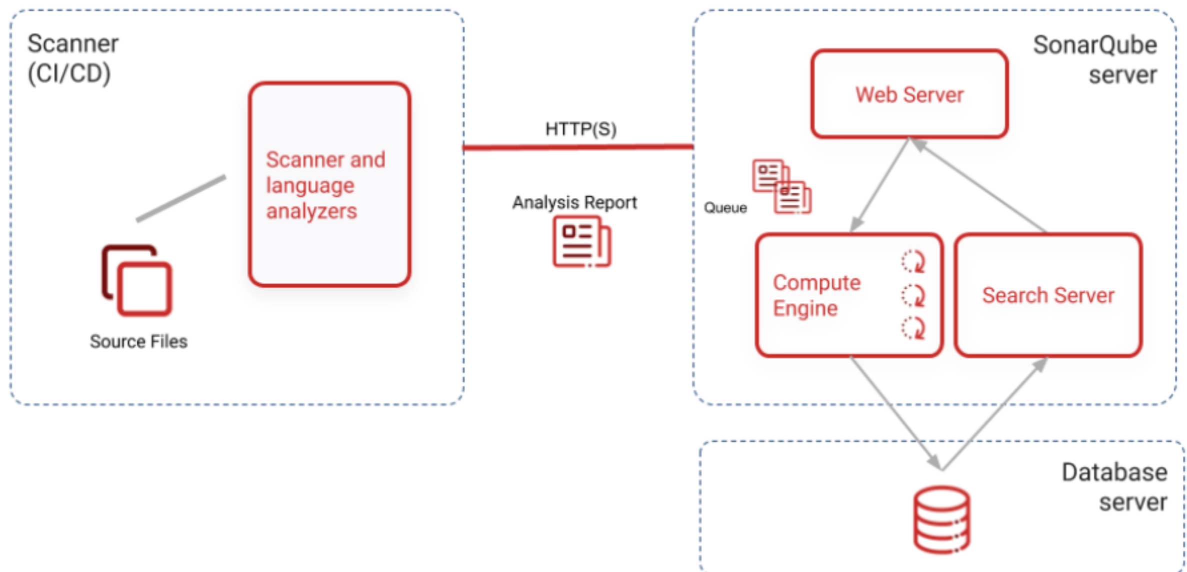


Рисунок 3.16 — Структурна схема компонентів екземпляру SonarQube

SonarQube надає зворотній зв’язок через свій інтерфейс користувача та через електронну пошту, щоб повідомити про проблеми, які потрібно вирішити. SonarQube також надає поглиблені вказівки щодо проблем, пояснюючи, чому кожна проблема є проблемою та як її вирішити (Рисунок 3.17).

Add a <title> tag to this page.
"<title>" should be present in all pages Web:PageWithoutTitleCheck
Get permalink
12 days ago L7
Bug Major Open Not assigned 5min effort 0 comments
user-experience
Where is the issue? Why is this an issue?
Store App Pages/_Layout.cshtml
See all issues in this file

```
6   <html lang="en">
7   <head>
        Add a <title> tag to this page.
8       <meta charset="utf-8" />
9       <meta name="viewport" content="width=device-width, initial-scale=1.0" />
```

Where is the issue? Why is this an issue?

Titles are important because they are displayed in search engine results as well as the browser's toolbar.

This rule verifies that the `<head>` tag contains a `<title>` one, and the `<html>` tag a `<head>` one.

Noncompliant Code Example

```
<html>        <!-- Non-Compliant -->
<body>
```

Рисунок 3.17 — Звіт про виявлену проблему в SonarQube

Для встановлення та налаштування SonarQube серверу використаємо Docker Compose та конфігураційний файл .yml з вмістом, наведеним в додатку Ж.

Розділ SonarQube містить останню версію образу. У розділі середовища вказано змінні для підключення до бази даних PostgreSQL, секція дискового простору SonarQube використовується для збереження конфігурації. PostgreSQL також використовує останню версію, а розділ середовища, яке використовується для бази даних SonarQube, — зберігання з'єднань JDBC.

Після встановлення платформи SonarQube, можна встановити сканер і створити новий проект. Для цього необхідно встановити та налаштувати плагін, який найбільше відповідає потребам проекту. Цей плагін дозволяє централізувати налаштування деталей підключення до сервера SonarQube у глобальній конфігурації Jenkins, потім можна запустити аналіз SonarQube від Jenkins за допомогою стандартних кроків збірки Jenkins. Після завершення завдання плагін виявить, що аналіз SonarQube було проведено під час збірки, і відобразить значок і віджет на сторінці завдання з посиланням на інформаційну панель SonarQube, а також статус перевірки якості.

Проект створюється в SonarQube автоматично під час першого аналізу. SonarQube може аналізувати до 29 різних мов залежно від версії, а результатом цього аналізу стануть показники якості та проблеми (випадки порушення правил кодування). Під час аналізу дані запитуються із сервера, файли, надані для аналізу, аналізуються, а отримані дані надсилаються назад на сервер у вигляді звіту, який потім аналізується асинхронно на сервері.

Звіти про аналіз ставляться в чергу та обробляються послідовно, тому цілком можливо, що протягом короткого періоду після того, як журнал аналізу сигналізує про завершення, оновлені значення не відображатимуться у проекті SonarQube. Однак на домашній сторінці проекту праворуч від назви проекту буде додано значок з додатковими відомостями (Рисунок 3.18).

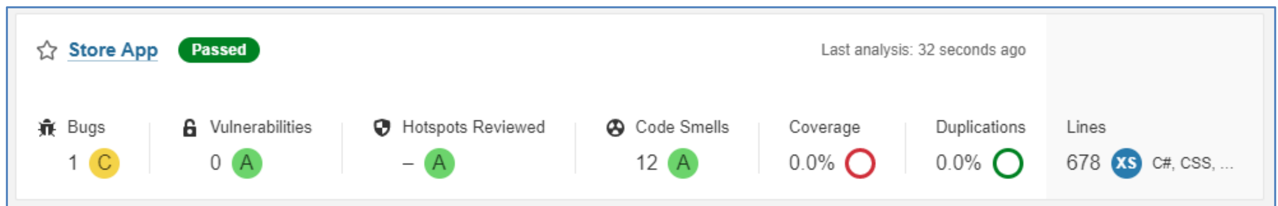


Рисунок 3.18 — Результат аналізу збірки на панелі SonarQube

Модель якості SonarQube має чотири різні типи правил: правила надійності — Bug, ремонтопридатність — Code Smells, правила безпеки — Vulnerability та Hotspot.

SonarQube базується на різних представленнях вихідного коду та технологій, щоб мати можливість виявляти будь-які проблеми безпеки.

Одним з засобів виявлення та протидії вразливостям в SonarQube є правила ін'єкції безпеки. Вони протидіють вразливості, коли вхідні дані, які обробляються програмою, контролюються користувачем (потенційно зловмисником) і не перевіряються чи не очищаються. Для цього SonarQube використовує технологію аналізу забруднення вихідного коду, яка дозволяє, наприклад, виявляти:

— CWE-89: SQL-ін'єкції (SQL Injection);

— CWE-79: Міжсайтовий сценарій (Cross-site Scripting);

— CWE-94: Ін'єкції коду (Code Injection).

Іншим засобом є правила конфігурування безпеки. Вони протидіють проблемі безпеки, коли під час виклику конфіденційної функції було встановлено неправильний параметр (наприклад, недійсний криптографічний алгоритм або версія TLS) або коли перевірка (наприклад, функція типу check_permissions()) не була виконана, або була виконана не в правильному порядку. Технологія дозволяє виявити такі проблеми:

— CWE-1004: конфіденційний файл cookie без прапорця «Тільки Http»;

— CWE-297: неправильна перевірка сертифіката з невідповідністю хосту;

— CWE-327: використання зламаного або ризикованого криптографічного алгоритму.

Ці проблеми безпеки поділяються на дві категорії: вразливості та гарячі точки. Гарячі точки безпеки (Security Hotspots) були запроваджені для захисту безпеки, який не має прямого впливу на загальну безпеку програми. Більшість правил ін'єкцій є вразливими місцями, наприклад, якщо виявлено ін'єкцію SQL, потрібне виправлення (перевірка введених даних), тому це є вразливістю. Навпаки, прапорець httpOnly під час створення файлу cookie є бажаним додатковим захистом (щоб зменшити вплив, коли з'являються вразливості XSS), але це не завжди можливо реалізувати або це не завжди є доречним залежно від контексту програми, тому це гаряча точка. Гарячі точки (Hotspots) допомагають розробникам зрозуміти ризики інформаційної безпеки, загрози, вплив, першопричини проблем із безпекою та вибрати відповідний захист програмного забезпечення.

Параметри аналізу проекту в SonarQube можна налаштувати в кількох місцях. Ієрархія в порядку пріоритету показана на рисунку 3.19.

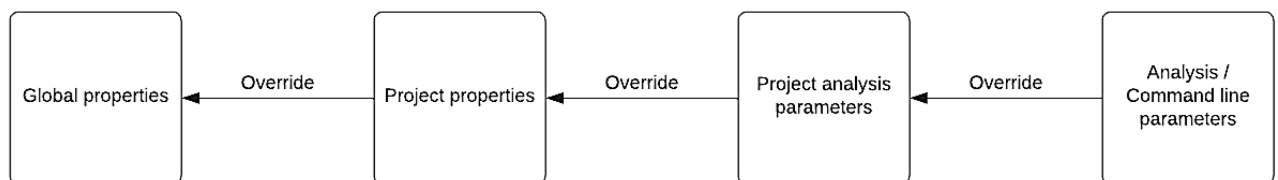


Рисунок 3.19 — Ієрархія параметрів аналізу проекту в SonarQube

Глобальні властивості (Global properties) застосовуються до всіх проектів, та визначаються в інтерфейсі користувача в «Адміністрування» > «Конфігурація» > «Загальні параметри».

Властивості проекту (Project properties) застосовуються лише до одного проекту та визначаються в інтерфейсі користувача в розділі «Налаштування проекту» > «Загальні параметри».

Параметри аналізу проекту (Project analysis parameters) визначаються у файлі конфігурації аналізу проекту або файлі конфігурації сканера.

Параметри аналізу/командного рядка (Analysis/Comand line parameters) визначаються під час запуску аналізу за допомогою параметру -D у командному рядку.

Більшість ключів властивостей, показаних в інтерфейсі як на глобальному рівні, так і на рівні проекту, також можна встановити як параметри аналізу.

Поріг якості (Quality Gate) являє собою найкращий спосіб реалізації концепції «Clean as You Code», зосереджуючись на новому коді. Поріг якості — це індикатор якості коду, який можна налаштувати для подання сигналу «запуск/незапуск» щодо поточної придатності випуску коду, він вказує на те, чи код чистий і чи може бути відправлений на наступну стадію конвеєру інтеграції та доставки. Прохідний (зелений) контроль якості означає, що код відповідає стандарту та готовий до об’єднання, тоді як несправний (червоний) контроль якості означає, що є проблеми, які потрібно вирішити.

За допомогою Quality Gate можна встановити оцінки (надійність, безпека, перевірка безпеки та ремонтопридатність) на основі показників загального коду та нового коду, які є частиною стандартного параметра якості. Однак, хоча якість тестового коду впливає на показники якості, вона вимірюється лише на основі показників ремонтопридатності та надійності, а дублювання та проблеми безпеки не вимірюються в тестовому коді.

Поточний статус порога якості видно у верхній частині сторінки проекту (Рисунок 3.20).

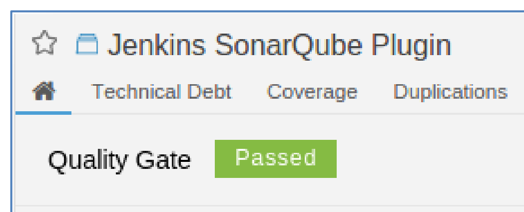


Рисунок 3.20 — Статус порогу якості проекту в SonarQube

Проект не пройде перевірку якості, якщо він переступить будь-які метричні порогові значення, встановлені в умовах (Conditions) порогу якості для нового коду або загального коду (Рисунок 3.21).

Conditions on New Code

Metric	Operator	Value
Coverage	is less than	80.0%
Duplicated Lines (%)	is greater than	3.0%
Maintainability Rating	is worse than	A
Reliability Rating	is worse than	A
Security Hotspots Reviewed	is less than	100%
Security Rating	is worse than	A

Рисунок 3.21 — Умови перевірки коду в SonarQube

Визначення показників (Metric Definitions) умов якості коду наведені в таблиці 3.1.

Таблиця 3.1 — Визначення показників умов якості коду в SonarQube

Назва показника	Пояснення	Визначення
Покриття (Coverage)	Показник, який визначає яка частина вихідного коду була охоплена модульними тестами.	$Coverage = \frac{CT+CF+LC}{2*B+EL}$, де CT — умови, які були оцінені як «істинні» принаймні один раз; CF — умови, які принаймні один раз були оцінені як «хибні»; LC — покриті лінії = лінія_покриття - непокриті_лінії; B — загальна кількість умов; EL — загальна кількість виконуваних рядків (lines_to_cover).
Дубльовані рядки (Duplicated lines)	Кількість рядків, залучених до дублювання.	$Duplicated\ lines(\%) = \frac{duplicated_lines}{lines*100}$, де duplicated_lines — кількість дубльованих рядків; lines — кількість рядків в проекті.
Рейтинг ремонтопридатності (Maintainability Rating)	Рейтинг, наданий проекту, який пов’язаний зі значенням коефіцієнта технічного боргу (Technical Debt Ratio).	Таблиця рейтингу ремонтопридатності за замовчуванням така: A=0-0,05; B=0,06-0,1; C=0,11-0,20; D=0,21-0,5; E=0,51-1.
Рейтинг надійності (Reliability Rating)	Кількість проблем із помилками.	A = 0 помилок; B = принаймні 1 незначна помилка; C = принаймні 1 серйозна помилка; D = принаймні 1 критична помилка; E = принаймні 1 помилка блокувальника.

Продовження таблиці 3.1

Перевірені точки безпеки (Security Hotspots Reviewed)	Кількість точок безпеки. Точки безпеки вважаються перевіреними, якщо вони позначені як підтверджені, фіксовані або безпечні.	Рейтинг перевірки безпеки — це буквена оцінка на основі відсотка перевірених точок безпеки: A = >= 80%; B = >= 70% і <80%; C = >= 50% і <70%; D = >= 30% і <50%; E = < 30%.
Рейтинг безпеки (Security Rating)	Рейтинг, що базується на кількості проблем із уразливістю.	A = 0 вразливостів; B = принаймні 1 незначна вразливість; C = принаймні 1 серйозна вразливість; D = принаймні 1 критична вразливість; E = принаймні 1 вразливість блокувальника.

Для того, щоб сповістити зовнішні служби про завершення аналізу проекту використовуються вебхуки (Webhooks). На кожну URL-адресу надсилається запит HTTP POST із корисним навантаженням (Payload) JSON. Консоль адміністрування Webhook показує результат і час останньої доставки кожного Webhook із корисним навантаженням (Payload), доступним через значок списку (Рисунок 3.22).

Name	URL	Secret?	Last delivery
jenkins	http://172.31.87.48:8080/sonarqube-webhook/	No	December 4, 2022 at 3:54 PM

Рисунок 3.22 — Результат доставки Webhook в SonarQube

Після завершення перевірки коду, на Jenkins сервер буде відправлено результат аналізу Payload в форматі JSON. Приклад звіту про успішну перевірку наведено на рисунку 3.23.

HTTP-заголовок «X-SonarQube-Project» із ключем проекту надсилається для швидкої ідентифікації задіяного проекту.

Корисне навантаження (Payload) JSON містить наступні дані:

— час виконання аналізу: "analysedAt";

— ідентифікацію проаналізованого проекту: "project";

— кожен перевірений критерій Quality Gate та його статус: "qualityGate";

— статус Quality Gate проекту: "qualityGate.status";

— статус та ідентифікатор фонового завдання: "status" та "taskId";

— властивості, визначені користувачем: "properties".

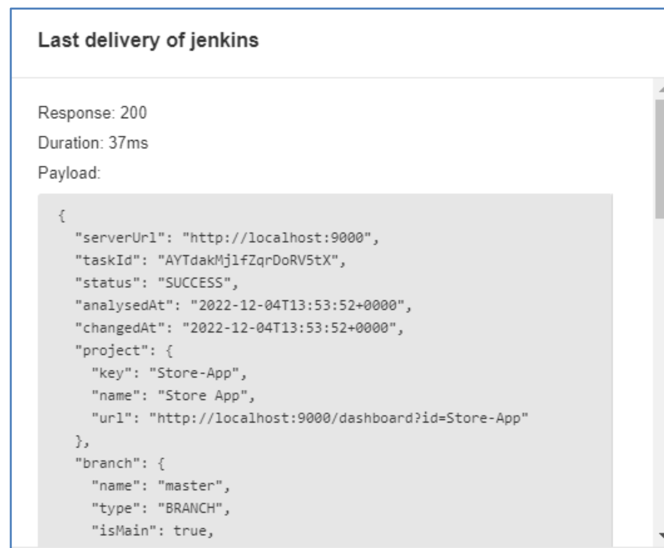


Рисунок 3.23 — Корисне навантаження (Payload) в форматі JSON

3.5 Налаштування реєстру контейнерів Amazon ECR

Після збірки проекту та успішного проведення тестування, Jenkins відправляє образ контейнера на віддалений реєстр контейнерів Amazon ECR.

Amazon Elastic Container Registry (Amazon ECR) — це безпечна, масштабована та надійна служба реєстру образів контейнерів, керована AWS.

Amazon ECR містить такі компоненти:

— реєстр — надається кожному обліковому запису AWS та дозволяє створити одне або кілька сховищ та зберігати в них образи;

— маркер авторизації — необхідний, щоб клієнт пройшов автентифікацію в реєстрах Amazon ECR як користувач AWS, перш ніж зможе надсилати та отримувати образи;

— репозиторій — містить образи Docker, образи Open Container Initiative (OCI) і артефакти, сумісні з OCI;

— політика сховища — дозволяє контролювати доступ до сховищ і образів у них;

— образ — надає можливість надсилати та витягувати образи контейнерів у сховища.

Для початку роботи в Amazon ECR, необхідно створити репозиторій — місце для зберігання образів Docker або Open Container Initiative (OCI) (Рисунок 3.24). Кожного разу, під час надсилання або витягування образу з Amazon ECR, необхідно вказувати сховище та розташування образу, які повідомляють, куди надсилати образ або звідки його завантажувати.

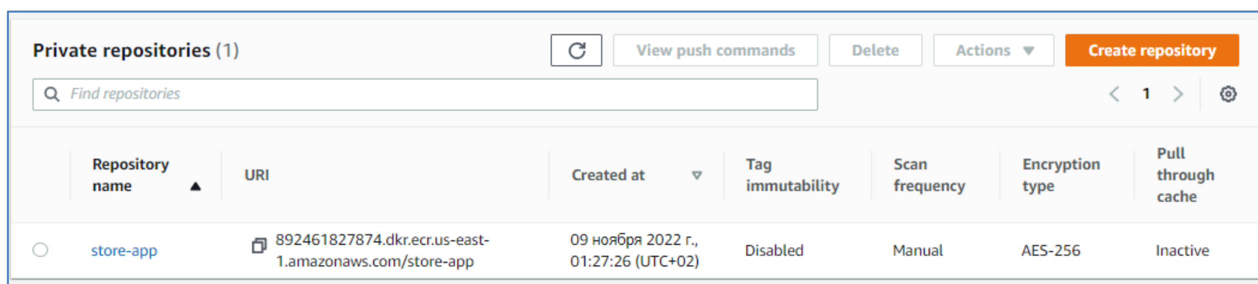


Рисунок 3.24 — Приватний репозиторій на Amazon ECR

Образ контейнера, створений Jenkins, потрапляє на Amazon ECR після збірки для подальшого зберігання. Образ створюється за допомогою файлу Dockerfile, розміщеному в репозиторії проекту.

Контейнери створюються з шаблону, доступного лише для читання, який називається образом. Образи зазвичай створюються з Dockerfile — відкритого текстового файлу, який визначає всі компоненти, що входять до складу контейнера (Рисунок 3.25).

Створення образу Docker із файлу Dockerfile відбувається за допомогою команди «docker build».

Dockerfile використовує функцію багатоетапної збірки Docker для створення та запуску в різних контейнерах. Контейнери збірки та запуску створюються із образів, наданих Microsoft у Docker Hub.

Dockerfile використовує образ dotnet/sdk для створення програми. Образ містить .NET SDK, який включає інструменти командного рядка (CLI). Образ оптимізовано для локальної розробки, налагодження та модульного тестування.

Інструменти, встановлені для розробки та компіляції, роблять образ відносно великим.

```
FROM mcr.microsoft.com/dotnet/aspnet:6.0 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build
WORKDIR /src
COPY . .
RUN ls
RUN dotnet restore "StoreApp.csproj"

RUN dotnet build "StoreApp.csproj" -c Debug -o /app/build

FROM build AS publish
RUN dotnet publish "StoreApp.csproj" -c Debug -o /app/publish /p:UseAppHost=false

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "StoreApp.dll"]
```

Рисунок 3.25 — Вміст файлу Dockerfile проекту

Dockerfile використовує образ dotnet/aspnet для запуску програми. Образ містить середовище виконання ASP.NET Core і бібліотеки та є оптимізованим для запуску програм у середовищі розгортання. Розроблений для швидкого розгортання та запуску програми, образ відносно малий, тому продуктивність мережі від реєстру Docker до хосту Docker оптимізована. У контейнер копіюються лише бінарні файли та вміст, необхідні для запуску програми [29].

Тепер у сценарії можна розгорнути програму в контейнері, скопіювавши її активи, необхідні під час виконання. Виконуємо команду «dotnet publish», щоб створити програму в режимі налагодження та створити ресурси в папці для публікації. Структурна схема створення та зберігання образів контейнерів наведена на рисунку 3.26.

Коли команда docker build створює образ, вона використовує вбудований кеш. Якщо файли *.csproj не змінилися з моменту останнього запуску команди «docker build», команду «dotnet restore» не потрібно запускати знову. Натомість повторно використовується вбудований кеш для відповідного рівня «dotnet restore».

Інструкція EXPOSE відкриває порт 80 та 443 контейнера.

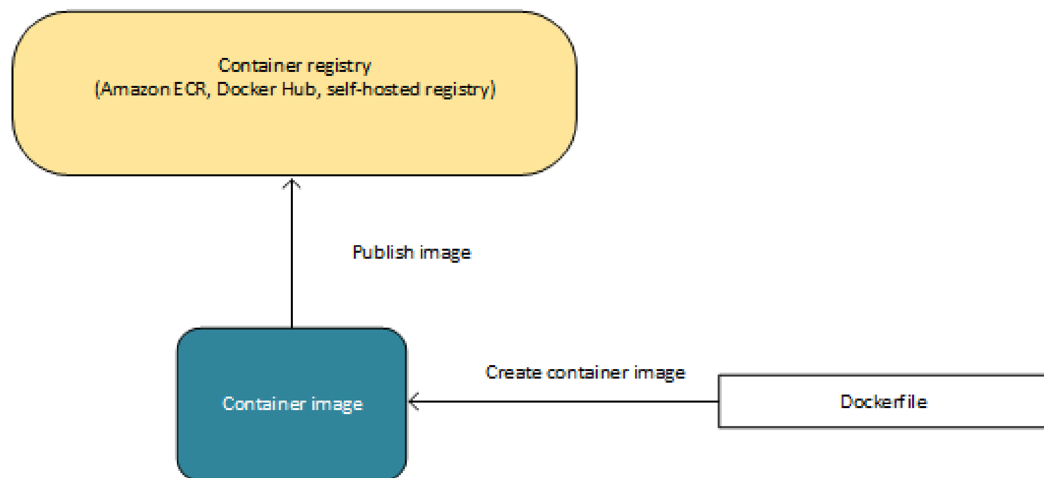


Рисунок 3.26 — Структурна схема створення та зберігання образу контейнера

Після того, як образ контейнеру буде відправлено у репозиторій, можна переглянути інформацію про нього на консолі керування AWS. Інформація включає такі деталі, як URI образу, тег, тип носія артефакту, тип маніфесту образу, статус сканування, розмір образу в Мб, коли зображення було відправлено в сховище, статус реплікації.

3.6 Налаштування служби оркестрування контейнерів Amazon ECS

Після збірки проекту, на стадії розгортання, Amazon ECS розгортає образ контейнера на екземплярах Amazon EC2 відповідно до образу з Amazon ECR.

Amazon Elastic Container Service (Amazon ECS) — це високомасштабована та швидка служба керування контейнерами, яку можна використовувати для запуску, зупинки та керування контейнерами в кластері. За допомогою Amazon ECS контейнери визначаються у визначенні завдання (Task Definition), яке використовується для виконання окремого завдання або завдання в службі. У цьому контексті служба — це конфігурація, яку можна використовувати для одночасного запуску та підтримки певної кількості завдань у кластері. Крім того, для більшого контролю над інфраструктурою, завдання та служби будуть запущені в кластері екземплярів Amazon EC2.

Будучи повністю керованим сервісом, Amazon ECS інтегрований як з AWS, так і зі сторонніми інструментами, такими як Amazon Elastic Container

Registry і Docker. Завдяки цій інтеграції командам легше зосередитися на створенні програм, а не на середовищі.

За допомогою Amazon ECS можна створити конвеєр CI/CD, який виконує такі дії:

— відстежує зміни в сховищі вихідного коду;

— створює новий образ Docker із цього джерела;

— надсилає образ до сховища зображень, наприклад Amazon ECR або Docker Hub;

— оновлює служби Amazon ECS, щоб використовувати новий образ у програмі.

За допомогою виявлення служб, мікросервісні компоненти виявляються автоматично, коли вони створюються та завершуються в певній інфраструктурі.

Amazon ECS використовує образи Docker у визначеннях завдань (task definitions) для запуску контейнерів. Після створення образу Docker і відправлення його в Amazon ECR, який є реєстром контейнерів, його використовує у визначеннях завдань Amazon ECS.

Для початку роботи з Amazon ECS створюємо кластер. Кластер Amazon ECS — це логічне групування завдань або послуг, всі завдання та служби виконуються в інфраструктурі, зареєстрованій у кластері. При першому використанні Amazon ECS, створюється кластер за замовчуванням, але можливо створити кілька кластерів в обліковому записі, щоб ресурси були розділеними (Рисунок 3.27).

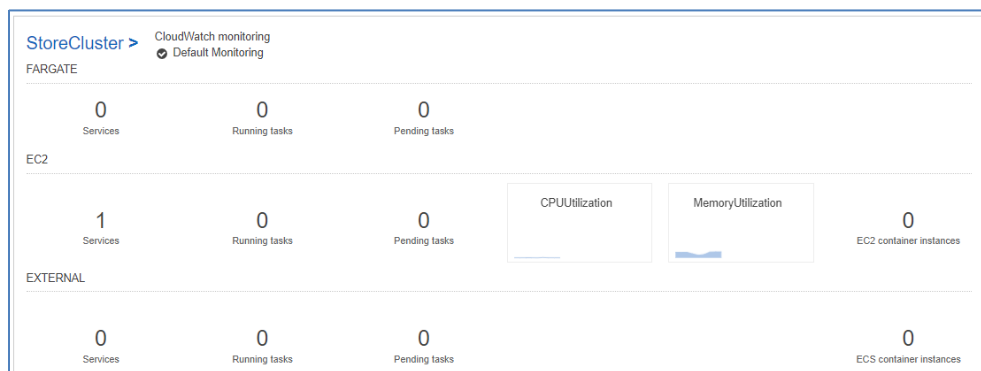


Рисунок 3.27 — Кластер в панелі керування Amazon ECS

Нижче наведено можливі стани, в яких може перебувати кластер:

— активний (Active) — кластер готовий приймати завдання, і, якщо це можливо, можна зареєструвати екземпляри контейнерів у кластері;

— забезпечення (Provisioning) — з кластером пов’язані постачальники потужностей (capacity providers), і створюються ресурси, необхідні для постачальника потужностей;

— депрівізіалізація (Deprovisioning) — з кластером пов’язані постачальники потужностей, а ресурси, необхідні для постачальника потужностей, видаляються;

— помилка (Failed) — з кластером пов’язані постачальники потужностей, і не вдалося створити ресурси, необхідні для постачальника потужностей;

— неактивний (Inactive) — кластер видалено, і він може залишатися видимим у обліковому записі протягом певного періоду часу.

Кластер може містити комбінацію постачальників потужностей групи з автоматичним масштабуванням (Auto Scaling group). Для завдань, які використовують тип запуску EC2, кластери можуть містити кілька різних типів екземплярів контейнерів, але кожен екземпляр контейнера може бути зареєстрований лише в одному кластері одночасно.

Щоб розгортати програми на Amazon ECS, компоненти програми мають бути налаштовані для роботи в контейнерах. Після створення з файлу Dockerfile, образи зберігаються в реєстрі, звідки їх можна завантажити, а потім використовувати для запуску на кластері.

Для роботи з Amazon ECS необхідно також створити визначення завдання (Task Definition) — це текстовий файл у форматі JSON, який описує один або кілька контейнерів, які утворюють програму (Рисунок 3.28). Визначення завдання функціонує як схема для програми та визначає різні параметри програми. Наприклад, воно може використовуватись щоб вказати параметри для операційної системи, які контейнери використовувати, які порти відкривати для програми та які обсяги даних використовувати з контейнерами в завданні.

Повний вміст файлу Task Definition в форматі JSON наведено в додатку Е.

Після створення визначення завдання в Amazon ECS, вказуємо кількість завдань для виконання на кластері. Завдання — це екземпляр визначення завдання в кластері.

Рисунок 3.28 — Визначення завдання в панелі керування Amazon ECS та його представлення в форматі JSON

Нижче наведено деякі з параметрів, які вказуються у визначенні завдання:

— образ Docker для використання з кожним контейнером у завданні;

— об’єм ЦП і пам’яті, який використовувати для кожного завдання або кожного контейнера в межах завдання;

— тип запуску для використання, який визначає інфраструктуру, на якій розміщено завдання;

— мережевий режим Docker для контейнерів у завданні;

— конфігурація логування для завдань;

— відображення стану продовження виконання завдання, якщо контейнер вимикається або відбувається збій;

— команда, яку виконує контейнер під час його запуску;

— будь-який об'єм даних, який використовується з контейнерами в завданні;

— роль IAM, яку використовують завдання.

Після створення визначення завдання його можна запустити як завдання або службу.

Використаємо службу Amazon ECS для одночасного виконання та підтримки бажаної кількості завдань у кластері Amazon ECS (Рисунок 3.29). Таким чином, якщо будь-яке з завдань не виконується або зупиняється з будь-якої причини, планувальник служби Amazon ECS запускає інший екземпляр на основі визначення завдання. Це робиться для того, щоб замінити його і таким чином зберегти бажану кількість завдань у службі.

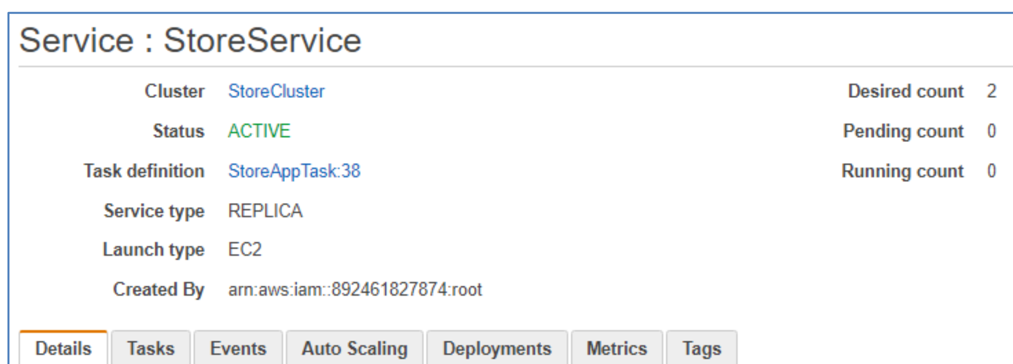


Рисунок 3.29 — Вигляд вікна служби в панелі керування Amazon ECS

Планувальник служби гарантує дотримання вказаної стратегії планування і перепланове завдання, якщо воно не виконується. Наприклад, якщо базова інфраструктура виходить з ладу, планувальник служби перепланує завдання. Якщо завдання в службі зупиняється, планувальник запускає нове завдання, щоб замінити його. Цей процес триває, доки служба не досягне бажаної кількості завдань на основі стратегії планування, яку використовує служба.

Планувальник служби містить логіку, яка регулює частоту перезапуску завдань, якщо завдання постійно не запускаються. Якщо завдання зупинено без переходу в стан «Запущено» (Running), планувальник служби починає сповільнювати спроби запуску та надсилає повідомлення про подію служби. Така поведінка запобігає використанню непотрібних ресурсів для невдалих завдань, перш ніж проблема буде вирішена. Після оновлення служби планувальник служби відновлює звичайну поведінку планування.

Доступні дві стратегії планувальника служби.

Першою стратегією є репліка (REPLICA). Стратегія планування репліки розміщує та підтримує бажану кількість завдань у кластері. За замовчуванням планувальник послуг розподіляє завдання між зонами доступності.

Другою можливою стратегією є демон (DAEMON). Стратегія планування демона розгортає рівно одне завдання на кожному активному екземплярі контейнера, яке відповідає всім обмеженням розміщення завдань, вказаним у кластері. Планувальник служби оцінює обмеження розміщення завдань для запущених завдань і зупиняє завдання, які не відповідають обмеженням розміщення.

Для розгортання контейнерів використаємо стратегію планувальника репліка (REPLICA), адже вона найкраще відповідає потребам проекту та забезпечить функціонування оптимальної кількості завдань в кластері.

Агент контейнера працює на кожному екземплярі контейнера в кластері Amazon ECS та дозволяє примірникам контейнера підключатися до кластера. Агент надсилає інформацію про поточні запущені завдання та використання ресурсів контейнерів до Amazon ECS, він запускає та зупиняє завдання щоразу, коли отримує запит від Amazon ECS. Структурну схему роботи агенту контейнера наведено на рисунку 3.30.

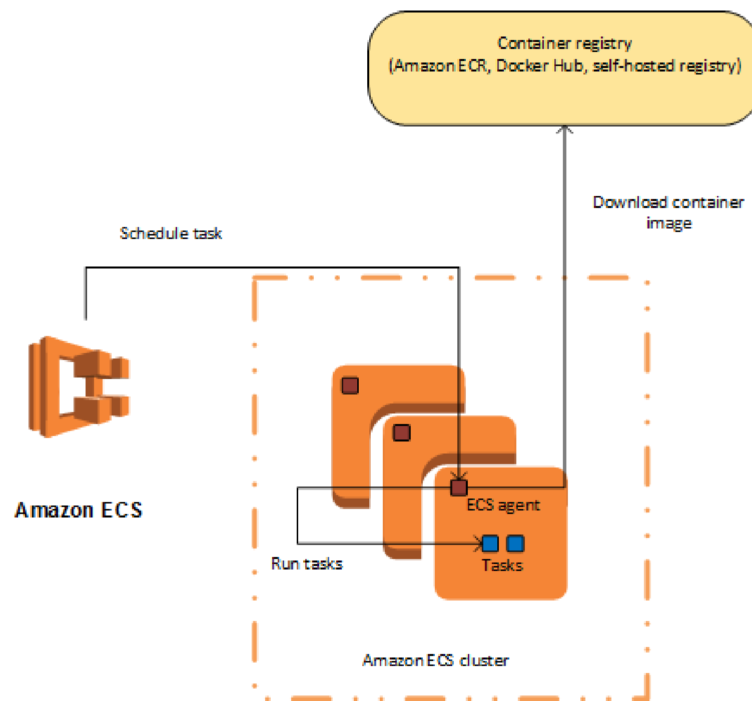


Рисунок 3.30 — Структурна схема роботи агенту контейнера в Amazon ECS

3.7 Налаштування стратегії розгортання контейнерів на екземплярах Amazon EC2 та групи автоматичного масштабування

Контейнерний додаток розгортається службою Amazon ECS на нових екземплярах Amazon EC2 відповідно до налаштувань групи автоматичного масштабування AutoScaling Group та кількості завдань.

Тип розгортання Amazon ECS визначає стратегію розгортання, яку використовує сервіс. Існує три типи розгортання: поточне оновлення (Rolling Update), синьо-зелене і зовнішнє.

Якщо служба використовує тип розгортання поточного оновлення, мінімальний відсоток представляє нижню межу кількості завдань у службі, які повинні залишатися в стані виконання під час розгортання.

Синьо-зелений тип розгортання використовує синьо-зелену модель розгортання, керовану CodeDeploy. Цей тип розгортання дає змогу перевірити нове розгортання служби перед надсиланням робочого трафіку до неї.

Зовнішній тип розгортання дозволяє використовувати будь-який сторонній контролер розгортання для контролю над процесом розгортання служби Amazon ECS.

Для розгортання проекту використаємо тип розгортання поточне оновлення, так як він витрачає менше ресурсів при розгортанні, зупиняючи неактуальні завдання та запускаючи актуальні, а також є найбільш зручним в керуванні та конфігуруванні, а також надає кращий рівень інтеграції з іншими хмарними сервісами AWS.

При використанні типу розгортання постійного оновлення, коли починається розгортання нової служби, планувальник служби Amazon ECS замінює поточні запущені завдання новими завданнями. Кількість завдань, які Amazon ECS додає або видаляє зі служби під час поточного оновлення, контролюється конфігурацією розгортання. Конфігурація розгортання складається зі значень minimumHealthyPercent і maximumPercent, які визначаються під час створення служби, але також можуть бути оновлені для наявної служби.

Значення minimumHealthyPercent являє собою нижню межу кількості завдань, які мають виконуватися для служби під час розгортання або коли екземпляру контейнера не вистачає наявних ресурсів для виконання завдань, як відсоток від бажаної кількості завдань для служби. Це значення округлюється в більшу сторону. Наприклад, якщо мінімальний відсоток становить 50, а бажана кількість завдань — чотири, тоді планувальник може зупинити два існуючих завдання перед початком двох нових. Подібним чином, якщо мінімальний відсоток працездатності становить 75%, а бажана кількість завдань дорівнює двом, тоді планувальник не може зупинити жодне завдання, оскільки кінцеве значення також дорівнює двом.

Максимальний відсоток maximumPercent являє собою верхню межу кількості завдань, які мають виконуватися для служби під час розгортання або коли екземпляру контейнера не вистачає наявних ресурсів для виконання завдань, як відсоток від бажаної кількості завдань для служби. Це значення

округлюється в меншу сторону. Наприклад, якщо максимальний відсоток дорівнює 200, а бажана кількість завдань дорівнює чотирьом, тоді планувальник може запустити чотири нові завдання, перш ніж зупинити чотири існуючі завдання. Подібним чином, якщо максимальний відсоток становить 125, а бажана кількість завдань дорівнює трьом, планувальник не може запустити жодне завдання, оскільки кінцеве значення також дорівнює трьом.

Коли починається розгортання нової служби або коли розгортання завершено, Amazon ECS надсилає подію зміни стану розгортання служби на EventBridge — програмний спосіб моніторингу стану розгортання служб.

Amazon ECS керує масштабуванням екземплярів Amazon EC2, зареєстрованих у кластері, що відбувається за допомогою постачальника ємності групи Amazon ECS Auto Scaling. Автоматичне масштабування — це можливість автоматично збільшувати або зменшувати бажану кількість завдань у службі Amazon ECS.

Під час цього процесу, Amazon ECS створює дві спеціальні метрики CloudWatch і цільову політику масштабування відстеження, яка приєднується до групи автоматичного масштабування. Після цього Amazon ECS керує діями з масштабування та масштабування групи Auto Scaling на основі навантаження, яке накладають завдання на кластер (Рисунок 3.31).

Рисунок 3.31 — Група автоматичного масштабування в Amazon EC2

Наступні показники допомагають визначити стратегію автоматичного масштабування:

— CapacityProviderReservation — відсоток примірників контейнера кластера, які використовуються для певного постачальника потужностей, Amazon ECS генерує цей показник;

— DesiredCapacity — ємність для групи автоматичного масштабування.

Amazon ECS запускає процес автоматичного масштабування кластера для кожного постачальника ресурсів, пов’язаного з кластером. Щохвилини Amazon ECS збирає інформацію, яка визначає, чи потрібно масштабувати групу автоматичного масштабування, і, якщо запущені завдання не можна розмістити в доступних екземплярах, група автоматичного масштабування масштабується шляхом запуску нових екземплярів. Коли є запущені екземпляри без завдань, група автоматичного масштабування масштабується шляхом припинення екземплярів без запущених завдань. Схему роботи Auto Scaling Group показано на рисунку 3.32.

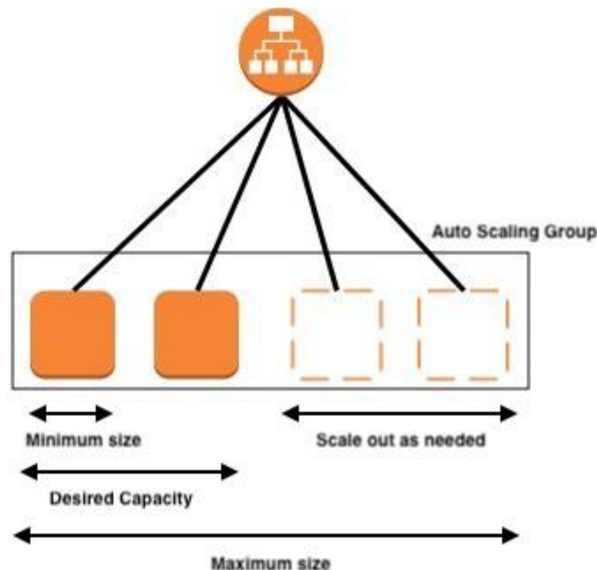


Рисунок 3.32 — Схема відтворення екземплярів в групі автоматичного масштабування

Коли запускається завдання з Amazon ECS за допомогою типу запуску EC2 або постачальника ємності групи автоматичного масштабування, завдання розміщуються на активних екземплярах контейнера. Екземпляр контейнера

Amazon ECS — це екземпляр Amazon EC2, який використовує агент контейнера Amazon ECS і зареєстрований у кластері Amazon ECS.

Коли контейнерний агент Amazon ECS реєструє екземпляр Amazon EC2 у кластері, екземпляр Amazon EC2 повідомляє про свій статус як активний (Active), а статус підключення агента як істинний (True). Цей екземпляр контейнера приймає запити RunTask.

В разі зупинки (не завершення) екземпляру контейнера Amazon ECS, статус залишиться активним, але статус підключення агента зміниться на хибний (False) протягом кількох хвилин. Будь-які завдання, які виконувалися на примірнику контейнера, припиняються. В разі повторного запуску екземпляру контейнера, агент контейнера знову підключиться до служби Amazon ECS, і з'явиться можливість знову запускати завдання на екземплярі.

3.8 Автоматизація керування об'єктами AWS за допомогою скрипта Python

Для взаємодії з хмарною інфраструктурою AWS компанія Amazon підтримує відкритий проект на GitHub, який реалізує Python SDK (System Development Kit). За допомогою бібліотеки цього проекту можна реалізувати взаємодію з хмарною інфраструктурою за допомогою скриптів. SDK надає об'єктно-орієнтований API, а також низькорівневий доступ до сервісів AWS.

SDK складається з двох ключових пакетів Python: Botocore (бібліотека, що надає функціональні можливості низького рівня, спільні між Python SDK і AWS CLI) і Boto3 (пакет, що реалізує сам Python SDK). Перед встановленням Boto3, необхідно встановити Python 3.7 або новішу версію.

Python — це мова високого рівня з динамічною типізацією, яка є однією з найпопулярніших мов програмування загального призначення. Вона використовується в машинному навчанні, веб-розробці, програмах і багатьох інших сферах.

Перш ніж використовувати Boto3, потрібно налаштувати облікові дані автентифікації для свого облікового запису AWS за допомогою консолі IAM або AWS CLI. Можна вибрати наявного користувача або створити нового.

Щоб використовувати Boto3 в скрипті, необхідно спочатку імпортувати його та вказати, які служби або послуги будуть використовуватись:

```
import boto3
```

У boto3 є два різних рівня API (Application Programming Interface):

— клієнтські API (низькорівневі) — забезпечують зв'язок з базовими операціями API HTTP.

— API ресурсів — відкривають явні мережеві виклики, що забезпечують ресурсні об’єкти та набори об’єктів для доступу до атрибутів і виконання дій.

Наприклад, створимо клієнт низького рівня з назвою служби:

```
ec2 = boto3.client('ec2')
iam = boto3.client('iam')
```

Як клієнтський, так і ресурсний інтерфейси boto3 динамічно генерують класи на основі моделі JSON, що описує API AWS. Це дозволяє швидко забезпечити оновлення з суворим непротиріччям для всіх підтримуваних сервісів.

У boto3 є функція waiter, яка автоматично виконує опитування заздалегідь визначених змін стану ресурсів AWS. Наприклад, запускаємо екземпляр Amazon EC2 і, скориставшись функцією waiter, очікуємо його переходу в робочий стан:

```
waiter = self.ec2.get_waiter('instance_running')
```

У boto3 є функції waiter як для клієнтських, так і для ресурсних API.

Повний лістинг скрипта для керування об’єктами AWS з використанням бібліотеки Python SDK наведено в додатку И.

Написаний скрипт забезпечить можливість легко керувати об’єктами AWS для швидкого створення та налаштування хмарної інфраструктури системи безпечної контейнеризації веб-додатку.

3.9 Аналіз ефективності розробленого програмного засобу

Після впровадження розробленої структури до повноцінного процесу розробки продукту було виявлено, що розроблений компонент позитивно впливає на результати виявлення секьюріті-вразливостей.

Було проаналізовано два подібних за структурою програмних продукти в рамках одного релізу. Для першого (Продукт А) було використано тестування з запуском тестів на зібраній та розгорнутій збірці. Для другого (Продукт Б) впроваджено компонент тестування у процес безперервної інтеграції та розгортання.

На рисунку 3.33 зображена статистика знайдених вразливостей для обох проектів.

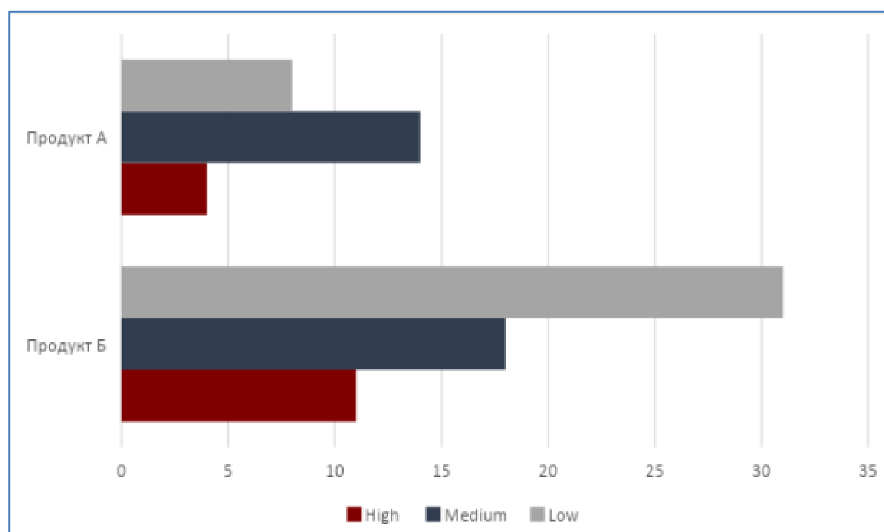


Рисунок 3.33 — Статистика знаходження вразливостей для обох проектів

Проведення розробки програмного компоненту займає період одного циклу розробки для одного розробника або спеціаліста DevOps, але є ефективним весь період розробки програмного продукту. У цей же час, використовуючи стару структуру, втрачається час на документування проблеми (створення звіту), час розробника на вирішення проблеми. З новою структурою проблема фіксується швидко, коли розробник тільки закінчив роботу над кодом, у той час як для старої структури проблема може бути знайдена через

деякий час, що може негативно вплинути на прибутковість продукту та потребує час на аналіз коду.

Таблиця 3.2 — Статистика знаходження вразливостей для обох проектів

Рівень знайдених проблем	Продукт зі старою структурою	Продукт із новою структурою
Високий (High)	4	12
Середній (Medium)	14	18
Низький (Low)	8	31

Таким чином, можемо зробити висновки, що використання розробленого програмного компоненту обійдеться дешевше, ніж виправлення знайдених помилок, які вже потрапили до артефакту.

Після впровадження програмного компоненту маємо нову схему проведення інтеграції коду. На рисунках 3.34 — 3.35 можна побачити що саме змінилось у процесі.

Були створені нові етапи в циклі безперервної інтеграції та розгортання, що передбачають виявлення критичних проблем безпеки в коді додатку, попередження атак та викрадення інформації, та автоматичне проведення тестування безпеки в процесі збірки.

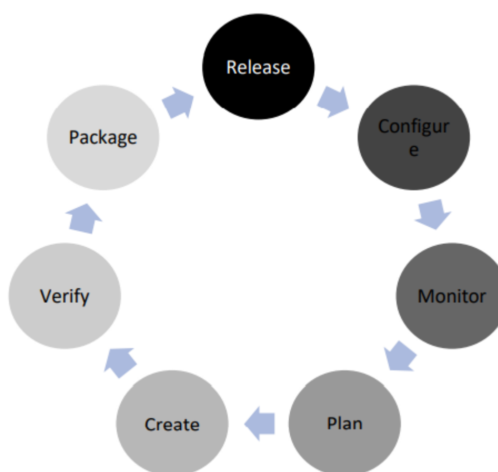


Рисунок 3.34 — Існуюча структурна схема CI/CD-циклу

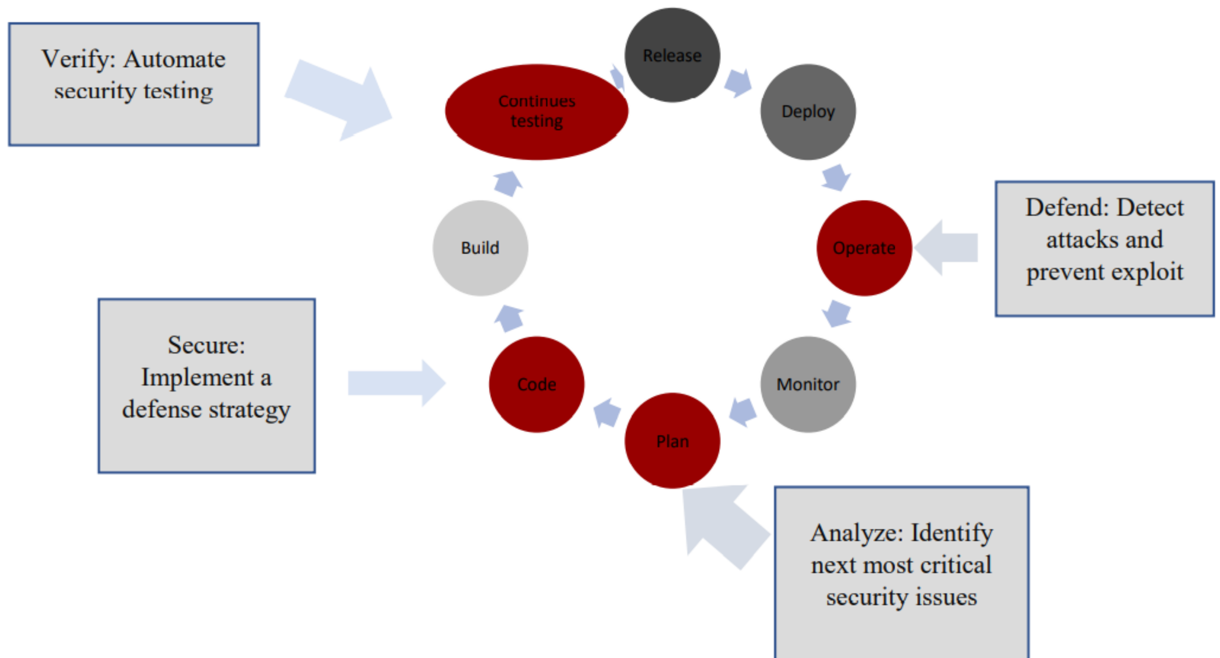


Рисунок 3.35 — Запропонована структурна схема CI/CD-циклу

Таким чином процес тестування безпеки стає невід'ємною частиною CI/CD-процесу.

4 ТЕСТУВАННЯ ТА ВЕРИФІКАЦІЯ ПРОГРАМНОГО ЗАСОБУ

4.1 Функції програмного засобу, які необхідно протестувати

Однією з основних функцій розробленого програмного засобу є перевірка коду додатку на вразливості в процесі безперервної інтеграції та розгортання. Перш за все необхідно перевірити, що в разі знаходження вразливості, код не пройде тестування та його не буде розгорнуто в контейнерному середовищі.

Також необхідною є перевірка таких функцій:

— відправлення звіту на поштову скриню розробнику у випадку не успішної збірки;

— доступ до розширеного звіту у Jenkins, на сторінці аналізатора SonarQube, або після відкриття посилання у листі;

— перевірка, що у випадку відсутності вразливостей, збірку буде здійснено успішно.

4.2 Верифікація програмного засобу

Верифікація — це процес оцінювання системи або її компонентів з метою визначення, чи задовольняють результати поточного етапу розробки умови, сформовані на початку цього етапу, тобто чи виконуються цілі, терміни, завдання з розробки проекту, визначені на початку поточної фази.

В ході верифікації перевіряється відповідність одних створених в ході розробки і супроводження ПЗ артефактів іншим, створеним раніше або використаним як вихідні дані, а також відповідність цих артефактів і процесів їх розробки правилам і стандартам. В цілому основною задачею верифікації, як і валідації, є контроль якості програмного забезпечення.

Згідно з технічним завданням, для реалізації поставленого завдання був розроблений програмний компонент, який дозволяє:

— проведення тестування безпеки програмного продукту;

— створення звіту на вкладці аналізатору у Jenkins;

— відправка звіту на поштову скриню у разі виявлення вразливостей;

— зупинка розгортання артефакту в середовищі у випадку виявлення вразливостей.

4.3 Тестування програмного засобу

Тестування програмного забезпечення включає дослідження ПЗ з метою отримання інформації про якість продукту, а також процес перевірки відповідності заявленим до продукту вимогам і реалізованої функціональності, що здійснюється шляхом спостереження за його роботою в створених ситуаціях і на обмеженому рівні тестів, які були обрані певним чином. Тестування — це одна з технік контролю якості (Quality Control), яка містить планування, складання тестів, безпосередньо виконане тестування і аналіз отриманих результатів.

Тестування ПЗ передбачає також інші дії, пов'язані з процесом забезпечення якості, зокрема:

— аналіз і планування стратегії тестування;

— розробка тестових сценаріїв;

— оцінка критеріїв закінчення тестування;

— написання звітів;

— рецензування документації (в тому числі і вихідного коду);

— проведення статичного аналізу.

Автоматизоване тестування додатків є процесом перевірки програмного забезпечення, що містить проведення таких основних функцій і кроків тесту, як запуск, ініціалізацію, виконання, аналіз і видачу результатів, автоматично за допомогою спеціалізованих інструментів.

Для проведення різних видів тестування використовується тестовий випадок (Test Case) — це артефакт, що описує сукупність кроків, конкретних умов і параметрів, необхідних для перевірки реалізації функції, що тестується, або її частини.

Опис різних тестових випадків та кроки проведення тестування програмного компоненту наведено у таблицях 4.1 — 4.3.

Таблиця 4.1 — Тест-кейс «Перевірка звіту у разі використання SQL-ін’єкції у коді»

Опис	Кроки для відтворення	Очікуваний результат
Відправка змін на віддалений репозиторій з використанням SQL-ін’єкції у коді (Рисунок 4.1- 4.2)	1) відкрити проект у середовищі розробки; 2) додати SQL-ін’єкцію до коду; 3) виконати коміт в системі контролю версій та відправити на віддалений репозиторій; 4) відкрити сторінку з початком збірки; 5) дочекатись проведення тестування; 6) перевірити результат проведення збірки.	Збірка була зупинена на кроці тестування безпеки.

З рисунків 4.1 — 4.3, видно, що тестування дійсно є не успішним, та подальша збірка проекту зупинена.

```
StoreApp | StoreApp.Services.ProductsService | DeleteProduct(int productId)
15
   Ссылок: 3
16 public async Task<IEnumerable<ProductModel>> GetProducts()
17 {
18     return await _storeDbContext.Products.Include(p=> p.Categories).ToListAsync();
19 }
   Ссылок: 1
20 public async Task<ProductModel> GetProduct(int productId)
21 {
22     return await _storeDbContext.Products.Include(p => p.Categories).FirstOrDefaultAsync(p=> p.Id == productId);
23 }
24
   Ссылок: 1
25 public async Task TestVulnerability(string sqlWithInjection)
26 {
27     var test = _storeDbContext.Categories.FromSqlRaw($"SELECT * FROM Categories WHERE Name = '{sqlWithInjection}'");
28 }
```

Рисунок 4.1 — Використання SQL-ін’єкції у коді під час реалізації тест-кейсу №1

Видно, що звіт на вкладці аналізатора SonarQube (Рисунок 4.3) містить у собі інформацію про причину не успішної збірки, у даному випадку про наявність SQL-ін’єкції.

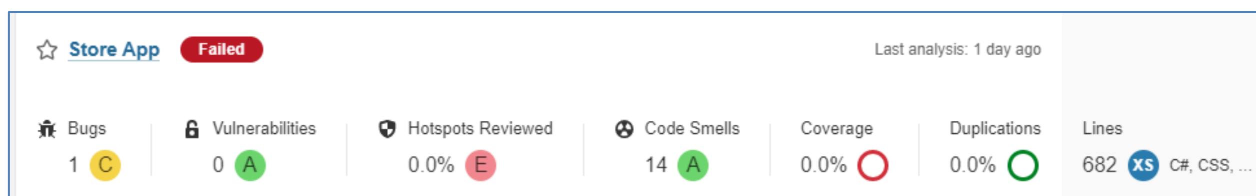


Рисунок 4.2 — Перевірка звіту на сторінці сканеру у разі знаходження вразливості під час реалізації тест-кейсу №1

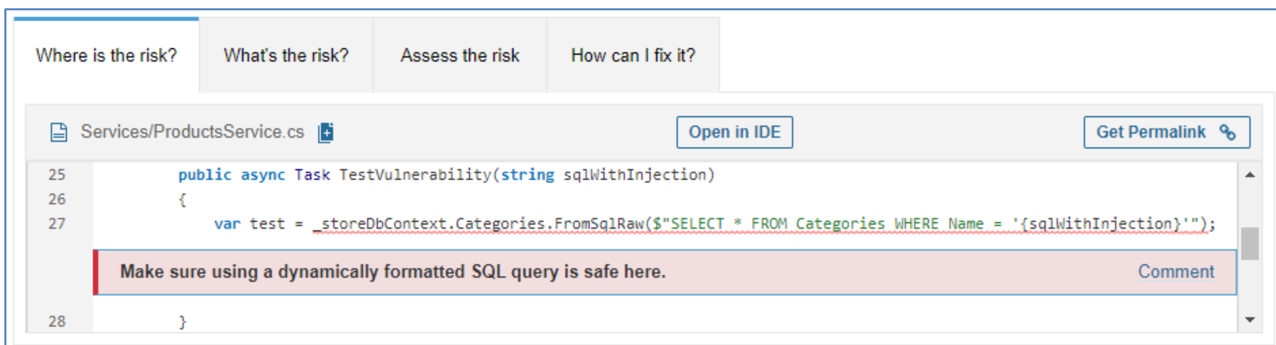


Рисунок 4.3 — Перевірка інформації про знайдену гарячу точку (Security Hotspot) на сторінці сканеру під час реалізації тест-кейсу №1

Таблиця 4.2 — Тест-кейс «Перевірка звіту на поштовій скрині у разі знаходження вразливості»

Опис	Передумови	Кроки для відтворення	Очікуваний результат
Перевірка звіту на поштовій скрині у разі знаходження вразливості (Рисунок 4.4)	Коміт, що містить у собі SQL-ін’єкцію, було відправлено на репозиторій.	1) відкрити сторінку з початком збірки; 2) дочекатись проведення тестування; 3) перевірити результат проведення збірки; 4) відкрити поштову скриню; 5) перевірити вміст листа.	На поштову скриню надійшов лист з інформацією щодо проваленої збірки.

З рисунку 4.4 видно, що до поштової скрині надійшов лист з інформацією щодо не пройденого аналізу.

Рисунок 4.4 — Перевірка звіту на поштовій скрині у разі знаходження вразливості під час реалізації тест-кейсу №2

Таблиця 4.3 — Тест-кейс «Виконання збірки без використання будь-яких вразливостей»

Опис	Кроки для відтворення	Очікуваний результат
Виконання збірки без використання будь-яких вразливостей (Рисунок 4.5)	1) відкрити проект у середовищі розробки; 2) додати будь-який валідний код; 3) виконати коміт за допомогою системи контролю версій та відправити на репозиторій; 4) відкрити сторінку з початком збірки; 5) дочекатись проведення тестування; 6) перевірити результат проведення збірки.	Збірка була пройдена успішно.

З рисунку 4.5 видно, що збірка була здійснена успішно, тестування було пройдено.

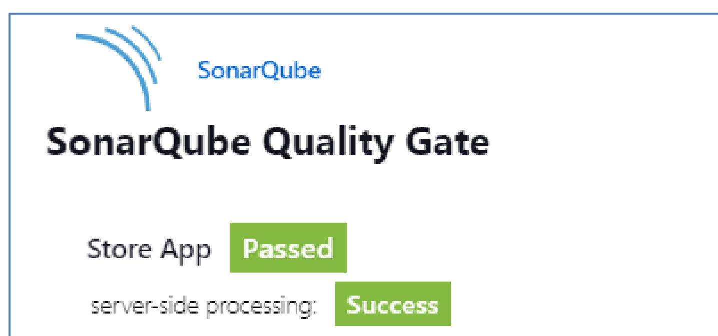


Рисунок 4.5 — Результат аналізу збірки з валідним кодом на вкладці аналізатора в Jenkins під час реалізації тест-кейсу №3

Відповідно до результатів тестування, можна зробити висновок, що розроблений програмний засіб працює та повноцінно виконує свою роботу. Проблема безпеки була знайдена одразу та не була допущена до вихідного артефакту збірки. Після аналізу було сформовано повний звіт, що робить аналіз проблем швидким та зручним.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Комерційний та технологічний аудит науково-технічної розробки

Метою даного розділу є проведення технологічного аудиту, в даному випадку розробки програмного засобу для автоматизації проведення тестування безпеки програмного продукту, який стане частиною CI/CD процесу та дозволить знизити ймовірність потрапляння неякісного коду до середовища.

Особливість програми полягає в розробці нового підходу до тестування безпеки продукту з впровадженням автоматизованого тестування у CI/CD процес, а також розробка засобу інтеграції ПЗ в ізольоване хмарне середовище (контейнеризації) з використанням розробленого компоненту кібербезпеки.

Аналогом може бути SonarCloud Data Center Plan — 8200 грн. або CloudBees Software Delivery Platform — 9500 грн.

Для проведення комерційного та технологічного аудиту залучають не менше 3-х незалежних експертів. Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням п’ятибальної системи оцінювання за 12-ма критеріями, у відповідності із табл. 5.1 [30].

Таблиця 5.1 — Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Бали (за 5-ти бальною шкалою)					
Кри-терій	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздат-ність продукту в реальних умовах
2	Багато аналогів на малому ринку	Ринкові п Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку

Продовження таблиці 5.1

Ринкові переваги					
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною .	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практик на здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так із комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві

Продовження таблиці 5.1

11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Усі дані по кожному параметру занесено в таблицю 5.2.

Таблиця 5.2 — Результати оцінювання комерційного потенціалу розробки

Критерії оцінювання	ПІБ експертів		
	Експерт 1	Експерт 2	Експерт 3
	Бали		
Технічна здійсненність концепції	3	4	4
Наявність аналогів на ринку	3	4	4
Цінова політика	4	4	3
Технічні та споживчі властивості виробу	3	3	4
Експлуатаційні витрати	3	4	3
Ринок збуту	4	3	4
Конкурентоспроможність	3	4	3
Фахівці з технічної і комерційної реалізації	4	4	4
Фінансування	4	4	3
Матеріально-технічна база	3	3	3
Термін реалізації ідеї	4	4	3
Супровідна документація	3	3	4
Сума	41	44	42
Середньоарифметична сума балів	(41+44+42) / 3 = 42,33		

За даними таблиці 5.2 можна зробити висновок щодо рівня комерційного потенціалу даної розробки. Для цього доцільно скористатись рекомендаціями, наведеними в таблиці 5.3 [30].

Таблиця 5.3 - Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ , розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

Як видно з таблиці, рівень комерційного потенціалу розроблюваного нового програмного продукту є високим, що досягається за рахунок того, що програмний продукт призначений для автоматизації проведення тестування безпеки програмного продукту, який стане частиною CI/CD процесу та дозволить знизити ймовірність потрапляння неякісного коду до середовища. Особливість програми полягає в розробці нового підходу до тестування безпеки продукту з впровадженням автоматизованого тестування у CI/CD процес, а також розробка засобу інтеграції ПЗ в ізольоване хмарне середовище (контейнеризації) з використанням розробленого компоненту кібербезпеки.

5.2 Прогнозування витрат на виконання науково-дослідної роботи

Основна заробітна плата розробників розраховується за формулою:

$$З_о = \frac{М}{Т_р} \cdot t, \qquad (5.1)$$

де М — місячний посадовий оклад конкретного розробника (дослідника), грн.;

$Т_р$ — число робочих днів в місяці, 23 днів;

t — число днів роботи розробника (дослідника).

Результати розрахунків зведемо до таблиці 5.4.

Таблиця 5.4 — Основна заробітна плата розробників

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник проекту	42000	1826,09	37	67565,217
Програміст	39000	1695,65	37	62739,130
Всього				130304,35

Так як в даному випадку розробляється програмний продукт, то розробник виступає одночасно і основним робітником, і тестувальником розроблюваного програмного продукту.

Додаткова заробітна плата прийнято розраховувати як 13,5 % від основної заробітної плати розробників та робітників:

$$З_д = З_о \cdot 13{,}5\ \% / 100\ \%. \qquad (5.2)$$

$$З_д = (130304{,}35 \cdot 13{,}5\ \% \ /\ 100\ \%\ ) = 17591{,}09\ \ (грн.)$$

Згідно діючого законодавства нарахування на заробітну плату складають 22 % від суми основної та додаткової заробітної плати.

$$Н_з = (З_о + З_д) \cdot 22\ \% / 100\%. \qquad (5.3)$$

$$Н_з = (130304{,}35 + 17591{,}09) \cdot 22\ \% \ /\ 100\ \% \ = 32537{,}00\ (грн.)$$

Оскільки для розроблювального пристрою не потрібно витрачати матеріали та комплектуючи, то витрати на матеріали і комплектуючі дорівнюють нулю.

Амортизація обладнання, що використовувалось для розробки в спрощеному вигляді амортизація обладнання, що використовувалась для розробки розраховується за формулою:

$$A = \frac{Ц}{Тв} \cdot \frac{t_{вик}}{12} \text{ [грн.]}, \quad (5.4)$$

де Ц — балансова вартість обладнання, грн.;

Т — термін корисного використання обладнання згідно податкового законодавства, років;

$t_{вик}$ — термін використання під час розробки, місяців.

Розрахуємо, для прикладу, амортизаційні витрати на комп’ютер балансова вартість якого становить 23522 грн., термін його корисного використання згідно податкового законодавства — 2 роки, а термін його фактичного використання — 1,61 міс.

$$A_{обл} = \frac{23522}{2} \times \frac{1{,}61}{12} = 1576{,}66 \text{грн.}$$

Аналогічно визначаємо амортизаційні витрати на інше обладнання та приміщення. Розрахунки заносимо до таблиці 5.5.

Таблиця 5.5 — Амортизаційні відрахування матеріальних і нематеріальних ресурсів для розробників

Найменування обладнання	Балансова вартість, грн.	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн.
Комп’ютер та комп’ютерна периферія (Комп’ютер Medion Akoya P6657, TP-LINK TL-WR841N, HP Laser 107a)	23522	2	1,61	1576,656
Офісне обладнення (меблі)	23000	4	1,61	770,833
Приміщення	1000000	20	1,61	6702,899
Всього				9050,39

Так як вартість ліцензійної ОС та спеціалізованих ліцензійних нематеріальних ресурсів менше 20000 грн.: ОС Windows 10 — 5700 грн., прикладний пакет Microsoft Office 2016 — 5100 грн., середовище розробки програмного забезпечення Visual Studio 2022 — 8046 грн., то даний нематеріальний актив не амортизується, а його вартість включається у вартість розробки повністю, $В_{нем.ак.1}$=18846 грн. А також ми маємо орендовані

нематеріальні ресурси (оренда серверів AWS EC2 — 1350 грн., оренда екземплярів баз даних AWS RDS — 520 грн., реєстр контейнерів AWS ECR — 120 грн.) і загальна вартість їх підписки склала 1990 грн.

Отже, загальна вартість використаних ліцензійних нематеріальних ресурсів склала $В_{нем.акзаг..}$ = 20836 грн.

Тарифи на електроенергію для непобутових споживачів (промислових підприємств) відрізняються від тарифів на електроенергію для населення. При цьому тарифи на розподіл електроенергії у різних постачальників (енергорозподільних компаній), будуть різними. Крім того, розмір тарифу залежить від класу напруги (1-й або 2-й клас). Тарифи на розподіл електроенергії для всіх енергорозподільних компаній встановлює Національна комісія з регулювання енергетики і комунальних послуг (НКРЕКП). Витрати на силову електроенергію розраховуються за формулою:

$$Е_{в} = \sqrt[Т_{ж}]{1 + \frac{Е_{абс}}{PV}} - 1, \tag{5.5}$$

де В — вартість 1 кВт-години електроенергії для 1 класу підприємства, В = 6,2 грн./кВт;

П — встановлена потужність обладнання, кВт. П = 0,5 кВт;

Ф — фактична кількість годин роботи обладнання, годин;

$К_{п}$ — коефіцієнт використання потужності, $К_{п}$ = 0,9.

$$В_{е} = 0{,}9 \cdot 0{,}5 \cdot 8 \cdot 37 \cdot 6{,}2 = 825{,}84 \text{ (грн.)}$$

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками. Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників [30]:

$$І_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (5.6)$$

де $Н_{ів}$ — норма нарахування за статтею «Інші витрати».

$І_в$ = 130304,35 * 85% / 100% = 110758,7 (грн.)

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін. Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…150% від суми основної заробітної плати дослідників:

$$Н_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.7)$$

де $Н_{нзв}$ — норма нарахування за статтею «Накладні (загальновиробничі) витрати».

$Н_{нзв}$ = 130304,35 * 130 % / 100 % = 169396 (грн.)

Сума всіх попередніх статей витрат дає загальні витрати на проведення науково-дослідної роботи:

$В_{заг}$ = 130304,35+17591,09+32537,00+9050,39+20836+825,84+110758,7+
+169396 = 491299,01 грн.

Загальні витрати на завершення науково-технічної роботи та оформлення її результатів розраховуються ЗВ, визначається за формулою:

$$ЗВ = \frac{В_{заг}}{\eta} \text{ (грн)}, \quad (5.8)$$

де η — коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи.

Так, якщо науково-технічна розробка знаходиться на стадії: науково-дослідних робіт, то η=0,1; технічного проектування, то η=0,2; розробки конструкторської документації, то η=0,3; розробки технологій, то η=0,4; розробки дослідного зразка, то η=0,5; розробки промислового зразка, то η=0,7; впровадження, то η=0,9. Оберемо η = 0,5, так як розробка, на даний момент, знаходиться на стадії дослідного зразка:

$$ЗВ = 491299{,}01 / 0{,}5 = 982598 \text{ грн.}$$

5.3 Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором

В ринкових умовах узагальнювальним позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку. Саме зростання чистого прибутку забезпечить потенційному інвестору надходження додаткових коштів, дозволить покращити фінансові результати його діяльності, підвищить конкурентоспроможність та може позитивно вплинути на ухвалення рішення щодо комерціалізації цієї розробки.

Для того, щоб розрахувати можливе зростання чистого прибутку у потенційного інвестора від можливого впровадження науково-технічної розробки необхідно:

— вказати, з якого часу можуть бути впроваджені результати науково-технічної розробки;

— зазначити, протягом скількох років після впровадження цієї науково-технічної розробки очікуються основні позитивні результати для потенційного інвестора (наприклад, протягом 3-х років після її впровадження);

— кількісно оцінити величину існуючого та майбутнього попиту на цю або аналогічні чи подібні науково-технічні розробки та назвати основних суб’єктів (зацікавлених осіб) цього попиту;

— визначити ціну реалізації на ринку науково-технічних розробок з аналогічними чи подібними функціями.

При розрахунку економічної ефективності потрібно обов’язково враховувати зміну вартості грошей у часі, оскільки від вкладення інвестицій до отримання прибутку минає чимало часу. При оцінюванні ефективності інноваційних проектів передбачається розрахунок таких важливих показників:

— абсолютного економічного ефекту (чистого дисконтованого доходу);

— внутрішньої економічної дохідності (внутрішньої норми дохідності);

— терміну окупності (дисконтованого терміну окупності).

Аналізуючи напрямки проведення науково-технічних розробок, розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором можна об’єднати, враховуючи визначені ситуації з відповідними умовами.

5.3.1 Розробка чи суттєве вдосконалення програмного засобу для використання масовим споживачем

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

$$\Delta П_i = (\pm\Delta Ц_0 \cdot N + Ц_0 \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (5.10)$$

де $\pm\Delta Ц_o$ — зміна вартості програмного продукту (зростання чи зниження) від впровадження результатів науково-технічної розробки в аналізовані періоди часу;

N — кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки;

$Ц_о$ — основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки, $Ц_о = Ц_б \pm \Delta Ц_о$;

Цб — вартість програмного продукту у році до впровадження результатів розробки;

ΔN — збільшення кількості споживачів продукту, в аналізовані періоди часу, від покращення його певних характеристик;

λ — коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт λ = 0,8333.

р — коефіцієнт, який враховує рентабельність продукту;

ϑ — ставка податку на прибуток, у 2022 році ϑ =18%.

Припустимо, що при прогнозованій ціні 4500 грн. за одиницю виробу, термін збільшення прибутку складе 3 роки. Після завершення розробки і її вдосконалення, можна буде підняти її ціну на 300 грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року — на 10000 шт., протягом другого року — на 8000 шт., протягом третього року на 5000 шт. До моменту впровадження результатів наукової розробки реалізації продукту не було:

$$\Delta П_1 = (0*300 + (4500 + 300)*10000)* 0,8333* 0,18) * (1 - 0,18) = 5534999,779 \text{ грн.}$$

$$\Delta П_2 = (0*300 + (4500 + 300)*(10000+8000)* 0,8333* 0,18) * (1 - 0,18) = 10627199,575 \text{ грн.}$$

$$\Delta П_3 = (0*300 + (4500 + 300)*(10000+8000+5000)* 0,8333* 0,18) * (1 - 0,18) = 13579199,457 \text{ грн.}$$

Отже, комерційний ефект від реалізації результатів розробки за три роки складе 29741398,81 грн.

5.3.2 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Розраховуємо приведену вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (5.11)$$

де $\Delta П_i$ — збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої науково-дослідної (науково-технічної) роботи, грн;

T — період часу, протягом якою виявляються результати впровадженої науково-дослідної (науково-технічної) роботи, роки;

τ — ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, $\tau = 0,05 \ldots 0,15$;

t — період часу (в роках).

Збільшення прибутку ми отримаємо починаючи з першого року:

$$ПП= (5534999,779/(1+0,1)^1)+(10627199,575/(1+0,1)^2)+(13579199,457/ /(1+0,1)^3) = 5031817,98 + 8782809,566 + 10202253,54 = 24016881,08 \text{ грн.}$$

Далі розраховують величину початкових інвестицій *PV*, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки. Для цього можна використати формулу:

$$PV = k_{інв} * ЗВ, \qquad (5.12)$$

де $k_{інв}$ — коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай $k_{інв}=2...5$, але може бути і більшим;

ЗВ — загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

$$PV = 2 * 982598 = 1965196,02 \text{ грн.}$$

Тоді абсолютний економічний ефект $E_{абс}$ або чистий приведений дохід (NPV, Net Present Value) для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV, \quad (5.13)$$

$$E_{абс} = 24016881{,}08 - 1965196{,}02 = 22051685{,}06 \text{ грн.}$$

Оскільки $E_{абс} > 0$ то вкладання коштів на виконання та впровадження результатів даної науково-дослідної (науково-технічної) роботи може бути доцільним.

Для остаточного прийняття рішення з цього питання необхідно розрахувати внутрішню економічну дохідність або показник внутрішньої норми дохідності (IRR, Internal Rate of Return) вкладених інвестицій та порівняти її з так званою бар'єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку науково-технічну розробку вкладати буде економічно недоцільно.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього використаємо формулу:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (5.14)$$

де $T_{ж}$ — життєвий цикл наукової розробки, роки.

$$E_в = \sqrt[3]{(1 + 22051685{,}06/1965196{,}02} - 1 = 1{,}303$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f, \quad (5.15)$$

де d — середньозважена ставка за депозитними операціями в комерційних банках; в 2022 році в Україні d = (0,09…0,14);

f — показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,5).

$$\tau_{min} = 0{,}14 + 0{,}05 = 0{,}19$$

Так як $Е_в > \tau_{min}$, то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$Т_{ок} = \frac{1}{Е_в}. \qquad (5.16)$$

$$Т_{ок} = 1 / 1{,}303 = 0{,}77 \text{ р.}$$

Оскільки $Т_{ок} < 3$-х років, а саме термін окупності рівний 0,77 роки, то фінансування даної наукової розробки є доцільним.

Висновки до розділу: економічна частина даної роботи містить розрахунок витрат на розробку нового програмного продукту, сума яких складає 982598 гривень. Було спрогнозовано орієнтовану величину витрат по кожній з статей витрат. Також розраховано чистий прибуток, який може отримати виробник від реалізації нового технічного рішення, розраховано період окупності витрат для інвестора та економічний ефект при використанні даної розробки. В результаті аналізу розрахунків можна зробити висновок, що розроблений програмний продукт за ціною дешевший за аналог і є висококонкурентоспроможним. Період окупності складе близько 0,77 роки.

ВИСНОВКИ

В магістерській кваліфікаційній роботі було створено програмний компонент для забезпечення аналізу на вразливості програмних продуктів, та виконана його інтеграція в CI/CD процес, що передбачає подальшу контейнеризацію.

Проведено аналіз сучасних технологій розробки веб-додатків, їх особливостей та вразливостей, в результаті якого визначено основні вимоги до програмного компоненту тестування безпеки для вдосконалення процесу доставки програмного продукту до кінцевого користувача та гарантування захисту персональної інформації користувача.

Проведено аналіз інструментів для розроблення та контейнерізації програмного засобу, в результаті якого вибрано контейнеризацію Docker та секьюріті-сканер SonarQube. Для створення повноцінного процесу доставки коду до середовища розгортання розроблений засіб інтегровано з системою Jenkins. Також використано сервіс AWS CodePipeline для компонування, тестування та розгортання програмного коду.

Розроблено метод тестування безпеки продукту, який полягає в створенні нового етапу у CI/CD процесі, що відповідає за проведення тестування вразливостей безпеки та гарантує зниження ймовірності потрапляння неякісного коду до середовища розгортання. Розроблено алгоритм процесу інтеграції програмного коду та структурну схему процесу розгортання програмного компоненту для забезпечення безперервної доставки контенту з подальшим розгортанням в хмарному контейнерному середовищі.

Розроблено засіб контейнеризації з використанням запропонованого компоненту кібербезпеки. Налаштовано процес інтеграції коду від відправлення змін на репозиторій до повної перевірки, створення артефакту та розгортанні в контейнерному середовищі.

Розроблений програмний компонент протестовано з використанням тестового веб-проекту з додаванням SQL-ін’єкції до коду. В процесі тестування

було знайдено помилку, згенеровано звіт, проблемний код не було розгорнуто в середовищі. Такий підхід передбачає наявність кваліфікованих технічних фахівців, здатних налагодити і підтримувати цей процес, а також дисципліна і відповідальність самих розробників.

Використання розробленого засобу розгортання додатку блокує можливість попадання неякісного коду до продукту, що значно знижує ймовірність виникнення вразливостей у програмі, суттєво скорочує час перевірки та фіксації знайдених проблем, а контейнеризація додатку забезпечує швидкість та незалежність від платформи розгортання. Його впровадження коштує дешевше, ніж виправлення помилок вже після проходження етапу перевірки, а сам компонент може бути налаштований без особливих витрат. Для покращення компоненту, є можливість додати створення баг-звіту у системі Jira (або інших подібних системах).

В кваліфікаційній роботі проведено розрахунок економічної ефективності розробленого рішення, який показав, що комерційний ефект від реалізації результатів розробки за три роки складає майже 30 млн.грн., а період окупності — близько 0,77 роки.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Довбиш А. С., Ободяк В. К., Шелехов І. В. Сучасні інформаційні технології в кібербезпеці: монографія / за ред. В.К. Ободяка, І.В. Шелехова. — Суми: СумДУ, 2021. — 348 с.

2. Ткаченко О., Ткаченко К. Кіберпростір і кібербезпека: проблеми, перспективи, технології. Цифрова платформа: інформаційні технології в соціокультурній сфері, 2018. — 75—86 с.

3. Кібербезпека та інформаційні технології. Монографія. / За заг. ред. В.С. Пономаренка. — Х.: ТОВ «ДІСА ПЛЮС», 2020. — 380 с.

4. Lietz S. What is DevSecOps [Електроний ресурс] / Lietz Shannon. — 2015. — Режим доступу до ресурсу: http://www.devsecops.org/blog/2015/2/15/what-is-devsecops (дата звернення 20.09.2022).

5. Jose F. Effective DevSecOps [Електроний ресурс] / Jose Fabio. — 2019. — Режим доступу до ресурсу: https://medium.com/@fabiojose/effectivedevsecops-f22dd023c5cd (дата звернення 23.09.2022).

6. Nikiforova E. What is the difference between DevOps and DevSecOps? [Електроний ресурс] / Nikiforova Ekaterina. — 2020. — Режим доступу до ресурсу: https://pvs-studio.com/en/blog/posts/0710/ (дата звернення 25.09.2022).

7. McKay J. How to use DevSecOps to smooth cloud deployment [Електроний ресурс] / McKay Jason. — 2016. — Режим доступу до ресурсу: https://www.networkworld.com/article/3041640/how-to-use-devsecops-to-smooth-cloud-deployment.html (дата звернення 26.09.2022).

8. Sunny Valley Networks What is DevSecOps? [Електроний ресурс] / Sunny Valley Networks. — 2022. — Режим доступу до ресурсу: https://www.sunnyvalley.io/docs/network-security-tutorials/what-is-devsecops (дата звернення 27.09.2022).

9. OWASP Software Assurance Maturity Model [Електронний ресурс] / OWASP. — 2022. — Режим доступу до ресурсу: https://owaspsamm.org (дата

звернення 27.09.2022).

10. Вісник студентського наукового товариства «Ватра» Вінницького торговельно-економічного інституту ДТЕУ. Вінниця: Редакційно-видавничий відділ ВТЕІ ДТЕУ, 2022. — 256 с.

11. Keary E. OWASP Testing Guide 4.0 [Електронний ресурс] / Keary Eoin. — 2013. — Режим доступу до ресурсу: https://owasp.org/www-pdf-archive/OTGv4.pdf (дата звернення 28.09.2022).

12. Kumar S. Static + Dynamic Code Analysis with SonarQube [Електронний ресурс] / Kumar Sandeep. — 2022. — Режим доступу до ресурсу: https://siddhivinayak-sk.medium.com/static-dynamic-code-analysis-with-sonarqube-af689124dab0 (дата звернення 01.10.2022).

13. Docker Inc. Docker [Електронний ресурс] / Docker Inc. — 2022. — Режим доступу до ресурсу: https://www.docker.com (дата звернення 02.10.2022).

14. Hwang J., Zeng S., Wu F. y., Wood T., A component-based performance comparison of four hypervisors," in 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), 2013, pp. 269—276.

15. Mulerickal F. J. P., Paul B., and Sastri Y., Evaluation of docker containers based on hardware utilization, in 2015 International Conference on Control Communication Computing India (ICCC), 2015, pp. 697—700.

16. Parida V., Digitalization, Editors Johan Frishammar Asa Ericson [Електронний ресурс] / Parida Vinit — 2018. — Режим доступу до ресурсу: http://urn.kb.se/resolve?urn=urn:nbn:se:ltu:diva-68008 (дата звернення 04.10.2022).

17. Аналіз та порівняння продуктивності віртуалізації та контейнеризації при розгортанні API на веб-сервері [Текст] / Л. І. Рудь, О. В. Войцеховська // Матеріали Всеукраїнської науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2023): Тез. доп. — Вінниця, 2023. Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2023/paper/viewFile/16890/14079

18. Nanath K., Pillai R., A model for cost-beneft analysis of cloud computing, Journal of International Technology and Information Management, vol. 22, no. 3, 2013, p. 6.

19. Barik R. K., Lenka R. K., Rao K. R., Ghose D., Performance analysis of virtual machines and containers in cloud computing, in 2016 international conference on computing, communication and automation (iccca). IEEE, 2016, pp. 1204—1210.

20. Canonical Ltd. Linux containers. [Електронний ресурс] / Canonical Ltd — 2022. — Режим доступу до ресурсу: https://linuxcontainers.org (дата звернення 08.10.2022).

21. Bernstein D., Containers and cloud: From lxc to docker to kubernetes, IEEE Cloud Computing, vol. 1, no. 3, 2014, pp. 81—84.

22. Використання хмарних технологій в методології DevOps та CI/CD процесі [Текст] / Л. І. Рудь, О. В. Войцеховська // Матеріали L наукової-технічної конференції підрозділів Вінницького національного технічного університету (2021) : Тез. доп. — Вінниця, 2021. Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2021/paper/view/11983/10003

23. Mell E. What is container management and why is it important? [Електронний ресурс] / Mell Emily. — 2020. — Режим доступу до ресурсу: https://www.techtarget.com/searchitoperations/definition/container-management-software (дата звернення 10.10.2022).

24. IBM Cloud Team, Containers vs. Virtual Machines (VMs): What's the Difference? [Електронний ресурс] / IBM Cloud. — 2021. — Режим доступу до ресурсу: https://www.ibm.com/cloud/blog/containers-vs-vms (дата звернення 11.10.2022).

25. Vrukshali T. What is Jenkins? [Електронний ресурс] / Vrukshali Torawane. — 2021. — Режим доступу до ресурсу: https://vrukshalitorawane.medium.com/jenkins-9f092475d4cc (дата звернення 12.10.2022).

26. OWASP Source Code Analysis Tools [Електронний ресурс] / OWASP. — 2022. — Режим доступу до ресурсу: https://owasp.org/www-community/Source_Code_Analysis_Tools (дата звернення 13.10.2022).

27. SonarQube Documentation Quality gates [Електронний ресурс] / SonarQube. — 2022. — Режим доступу до ресурсу: https://docs.sonarqube.org/latest/user-guide/quality-gates/ (дата звернення 13.10.2022).

28. Програмування: теорія та практика. Збірник матеріалів за результатами IT-проєкту міждисциплінарної інтеграції / За редакцією Омельчук Л.Л., Ткаченка О.М., Шишацької О.В. — Київ, 2022. — 155 с.

29. Microsoft Documentation Docker images for ASP.NET Core [Електронний ресурс] / Microsoft. — 2022. — Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/host-and-deploy/docker/building-net-docker-images?view=aspnetcore-7.0 (дата звернення 16.10.2022).

30. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Козловський В. О., Лесько О. Й., Кавецький В. В. — Вінниця : ВНТУ, 2021. — 42 с.

ДОДАТОК А

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп’ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

д.т.н., проф. О. Д. Азаров

“__” __________ 2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи на тему:

«Програмний засіб безпечної контейнеризації веб-додатку»

08-23.МКР.012.00.000 ТЗ

Науковий керівник к.т.н., доц. каф. ОТ

___________ Войцеховська О. В.

Студентка групи 1КІ-21м

______________ Рудь Л. І.

ВНТУ 2022

1 Підстава для використання МКР

1.1 Актуальність розробки полягає у вдосконаленні методу розгортання веб-додатку шляхом інтеграції тестування безпеки в CI/CD-процес з подальшою контейнеризацією.

1.2 Наказ про затвердження теми кваліфікаційної роботи.

2 Мета МКР і призначення розробки

2.1 Мета роботи — створення програмного засобу для автоматизації проведення тестування безпеки програмного продукту за допомогою інтеграції тестування безпеки та контейнеризації;

2.2 Призначення розробки — програмна реалізація покращеного методу розгортання веб-додатку, що включає аналіз коду та подальше розгортання в ізольованому хмарному середовищі.

3 Вихідні дані для виконання МКР

3.1 Проведення аналізу існуючих принципів та технологій тестування безпеки коду в CI/CD процесі;

3.2 Розробка структури програмного засобу безпечної контейнеризації веб-додатку;

3.3 Середовище розробки Visual Studio для платформи ASP.NET Core;

3.4 Хмарні засоби інтеграції та розгортання Amazon Web Services.

4 Вимоги до виконання МКР

4.1 Наявність веб-додатку для розгортання;

4.2 Наявність системи інтеграції та розгортання додатку з інтегрованим аналізом коду та контейнеризацією.

5 Етапи МКР та очікувані результати

Робота виконується за п’ять етапів, таблиця А.1.

Таблиця А.1 — Етапи виконання МКР

№ етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Аналіз сучасного стану досліджень в галузі розгортання додатків в хмарному середовищі	04.10.2022	15.10.2022	Аналітичний огляд літературних джерел
2	Побудова структурних моделей системи тестування та контейнеризації	16.10.2022	04.11.2022	Структурні моделі, 2 розділ
3	Практичне застосування та оцінка ефективності розроблених моделей	05.11.2022	30.11.2022	3 і 4 розділи
4	Підготовка економічної частини	30.11.2022	03.12.2022	5 розділ
5	Оформлення пояснювальної записки, графічного матеріалу і/або презентації	04.12.2022	18.12.2022	пояснювальна записка, графічний матеріал і/або презентація

6 Матеріали, що подаються до захисту МКР

До захисту подаються: пояснювальна записка МКР, графічні і ілюстративні матеріали, протокол попереднього захисту МКР на кафедрі, відзив наукового керівника, відзив опонента, протоколи проходження перевірки на плагіат, анотації до МКР українською та іноземною мовами, нормоконтроль про відповідність оформлення МКР діючим вимогам.

7 Порядок контролю виконання та захисту МКР

Виконання етапів графічної та розрахункової документації МКР контролюється науковим керівником згідно зі встановленими термінами. Захист МКР відбувається на засіданні екзаменаційної комісії, затверджено] наказом ректора.

8 Вимоги до оформлення МКР

8.1 При оформлювання МКР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— міждержавний ГОСТ 2.104-2006 «Едина система конструкторської документації. Основні написи»;

— Методичні вказівки до виконання магістерських кваліфікаційних робіт зі спеціальності 123 — «Комп’ютерна інженерія». Кафедра обчислювальної техніки ВНТУ 2022;

— документами на які посилаються у вище вказаних.

8.2 Порядок виконання МКР викладено в «Положення про кваліфікаційні роботи на другому (магістерському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21».

Технічне завдання отримала _______________ Рудь Л. І.

ДОДАТОК Б

Блок-схема процесу інтеграції програмного коду

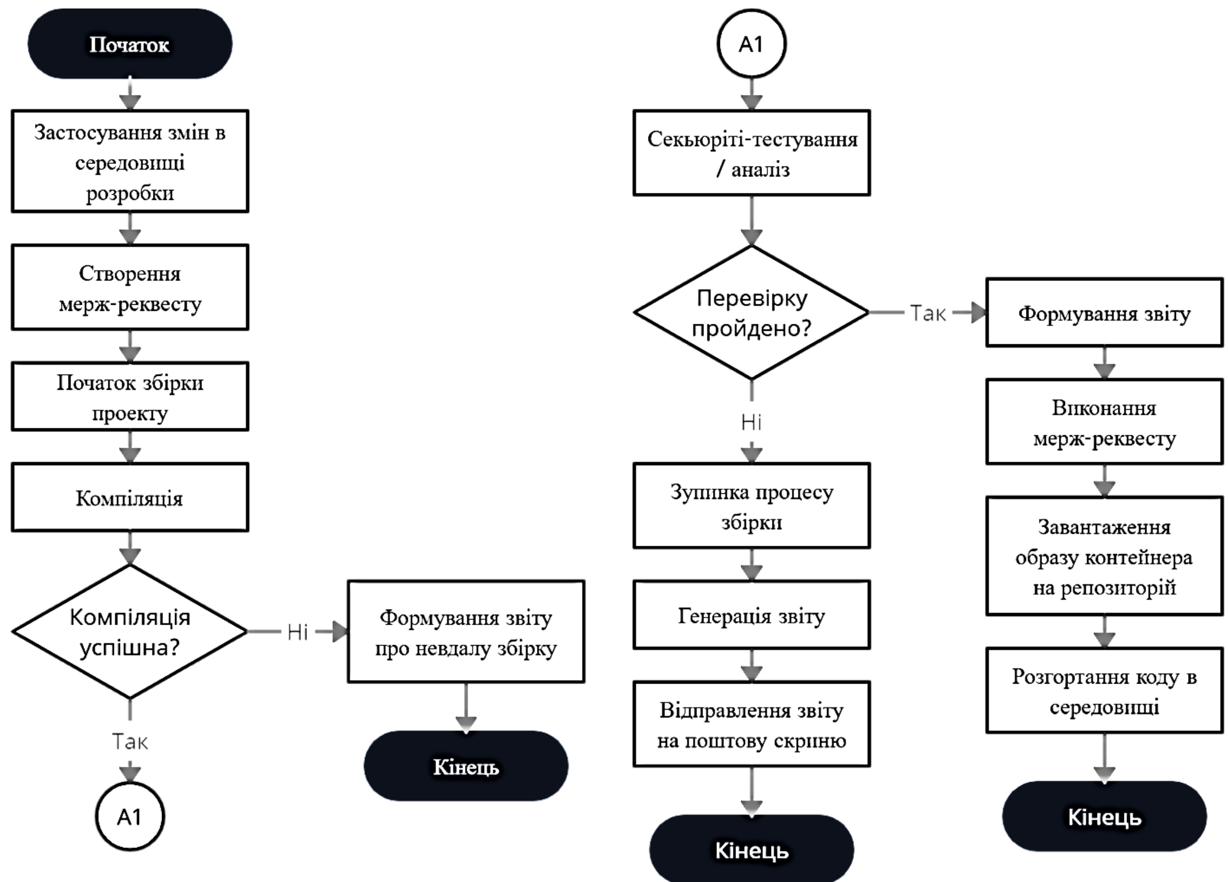


Рисунок Б.1 — Блок-схема процесу інтеграції програмного коду

ДОДАТОК В

Структурна схема CI/CD-ковеєру системи безпечної контейнеризації веб-додатку

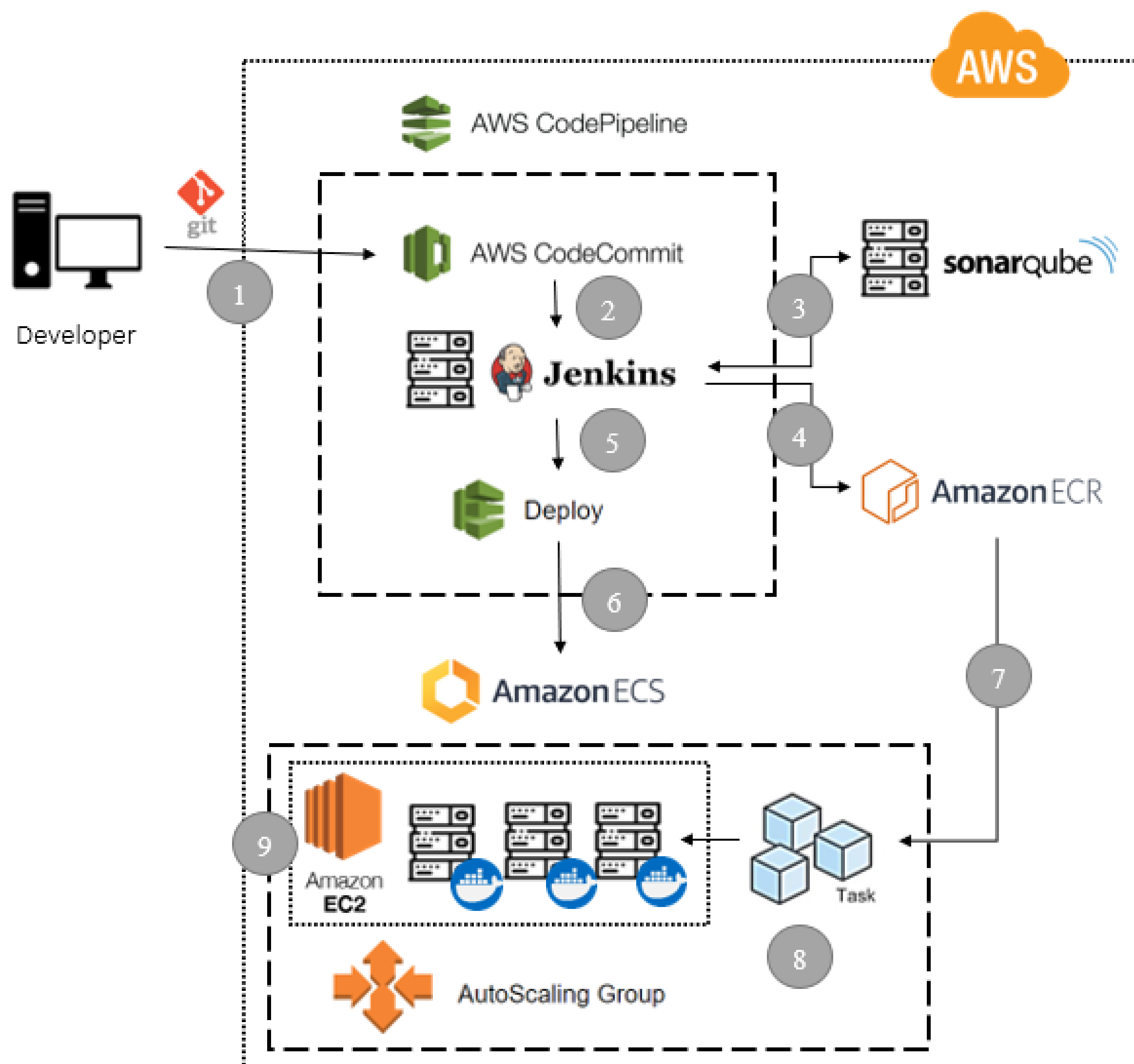


Рисунок В.1 — Структурна схема CI/CD-ковеєру системи безпечної контейнеризації веб-додатку

ДОДАТОК Г

Статистика знаходження вразливостей

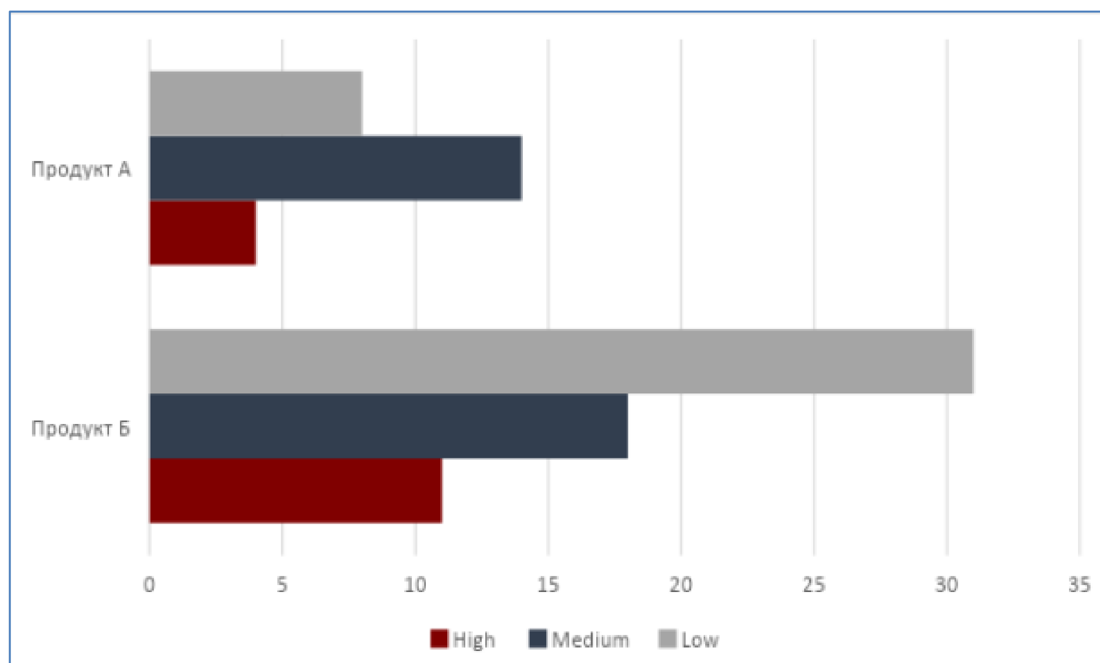


Рисунок Г.1 — Статистика знаходження вразливостей для проектів з запуском тестів на готовій збірці (Продукт А) та з впровадженим компонентом тестування (Продукт Б)

ДОДАТОК Д

Запропонована структурна схема CI/CD-циклу

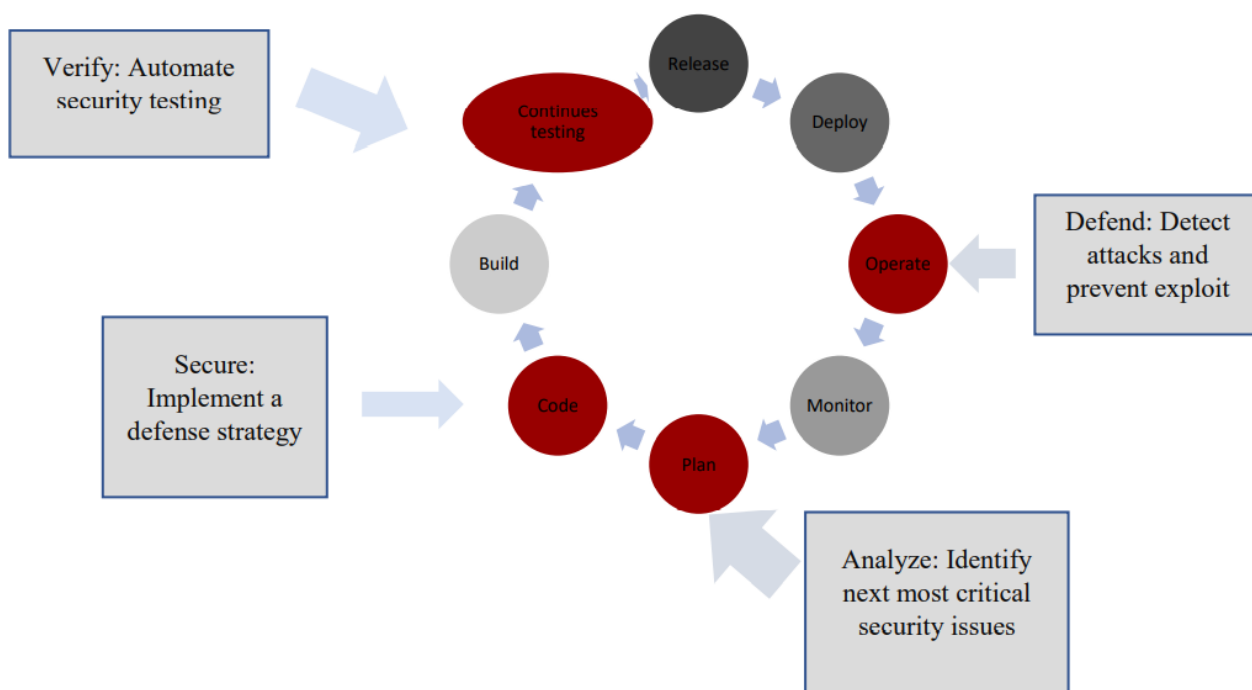


Рисунок Д.1 — Запропонована структурна схема CI/CD-циклу

ДОДАТОК Е

Вміст файлу Task Definition в форматі JSON


```
{
  "ipcMode": null,
  "executionRoleArn": null,
  "containerDefinitions": [
    {
      "dnsSearchDomains": null,
      "environmentFiles": null,
      "logConfiguration": null,
      "entryPoint": [],
      "portMappings": [
        {
          "hostPort": 80,
          "protocol": "tcp",
          "containerPort": 80
        }
      ],
      "command": [],
      "linuxParameters": null,
      "cpu": 0,
      "environment": [],
      "resourceRequirements": null,
      "ulimits": null,
      "dnsServers": null,
      "mountPoints": [],
      "workingDirectory": null,
      "secrets": null,
      "dockerSecurityOptions": null,
```

```
      "memory": 900,
      "memoryReservation": null,
      "volumesFrom": [],
      "stopTimeout": null,
      "image": "892461827874.dkr.ecr.us-east-1.amazonaws.com/store-app:latest",
      "startTimeout": null,
      "firelensConfiguration": null,
      "dependsOn": null,
      "disableNetworking": null,
      "interactive": null,
      "healthCheck": null,
      "essential": true,
      "links": null,
      "hostname": null,
      "extraHosts": null,
      "pseudoTerminal": null,
      "user": null,
      "readonlyRootFilesystem": null,
      "dockerLabels": null,
      "systemControls": null,
      "privileged": null,
      "name": "store-app"
    }
  ],
  "placementConstraints": [],
  "memory": "900",
  "taskRoleArn": null,
  "compatibilities": [
    "EXTERNAL",
    "EC2"
```

],
"taskDefinitionArn": "arn:aws:ecs:us-east-1:892461827874:task-definition/StoreAppTask:38",
"family": "StoreAppTask",
"requiresAttributes": [
{
"targetId": null,
"targetType": null,
"value": null,
"name": "com.amazonaws.ecs.capability.ecr-auth"
}
],
"pidMode": null,
"requiresCompatibilities": [
"EC2"
],
"networkMode": null,
"runtimePlatform": null,
"cpu": "1024",
"revision": 38,
"status": "ACTIVE",
"inferenceAccelerators": null,
"proxyConfiguration": null,
"volumes": []
}

ДОДАТОК Ж

Конфігураційний файл .yml для налаштування SonarQube

```yaml
version: "3"
services:
  sonarqube:
    image: sonarqube:community
    depends_on:
      - db
    environment:
      SONAR_JDBC_URL: jdbc:postgresql://db:5432/sonar
      SONAR_JDBC_USERNAME: sonar
      SONAR_JDBC_PASSWORD: sonar
    volumes:
      - sonarqube_data:/opt/sonarqube/data
      - sonarqube_extensions:/opt/sonarqube/extensions
      - sonarqube_logs:/opt/sonarqube/logs
    ports:
      - "9000:9000"
  db:
    image: postgres:12
    environment:
      POSTGRES_USER: sonar
      POSTGRES_PASSWORD: sonar
    volumes:
      - postgresql:/var/lib/postgresql
      - postgresql_data:/var/lib/postgresql/data
volumes:
  sonarqube_data:
  sonarqube_extensions:
```

postgresql:

postgresql_data:

ДОДАТОК И

Лістинг скрипта для керування об’єктами AWS з використанням бібліотеки Python SDK

```
import boto3
import socket
import paramiko
hostname=socket.gethostname()
IPAddr=socket.gethostbyname(hostname)
class JenkinsInitializer:
    ec2 = boto3.client('ec2')
    iam = boto3.client('iam')
    instance = None
    JENKINS_KEYPAIR_NAME = 'JenkinsKeyPair'
    JENKINS_SECURITY_GROUP_NAME = 'JenkinsSecurityGroup'
    JENKINS_ROLE_NAME = "JenkinsInstanceRole"
    def __init__(self):
        image_id = self.get_ami_id()
        self.create_key_pair()
        security_group_id = self.create_security_group()
        role = self.create_role()
        self.instance = self.create_jenkins_instance(security_group_id, image_id)
        self.connect_and_install()
    def create_role(self):
        policies = self.iam.list_policies(
            Scope="AWS"
        )["Policies"]
        aws_code_pipeline_policy = [policy for policy in policies if policy["Name"] ==
"AWSCodePipelineCustomActionAccess"]
```

```
        ecr_policy = [policy for policy in policies if policy["Name"] ==
"EC2InstanceProfileForImageBuilderECRContainerBuilds"]
        role = self.iam.create_role(
            RoleName=self.JENKINS_ROLE_NAME,
            Description='Jenkins role for CodeCommit and CodePipeline interaction',
        )["Roles"][0]
        self.iam.attach_role_policy(
            RoleName=self.JENKINS_ROLE_NAME,
            PolicyArn=aws_code_pipeline_policy["ARN"]
        )
        self.iam.attach_role_policy(
            RoleName=self.JENKINS_ROLE_NAME,
            PolicyArn=ecr_policy["ARN"]
        )
        return role
    def create_key_pair(self):
        key_pairs = self.ec2.describe_key_pairs()["KeyPairs"]
        if not any(element["KeyName"] in self.JENKINS_KEYPAIR_NAME for element
in key_pairs):
            response =
self.ec2.create_key_pair(KeyName=self.JENKINS_KEYPAIR_NAME)
            with open(f"{self.JENKINS_KEYPAIR_NAME}.ppk", "w") as f:
                print(response, file=f)
            return
        print(f"Key pair: {self.JENKINS_KEYPAIR_NAME} already exists")
    def create_security_group(self):
        security_groups = self.ec2.describe_security_groups()["SecurityGroups"]
        security_groups = [group for group in security_groups if group["GroupName"] ==
self.JENKINS_SECURITY_GROUP_NAME]
        jenkins_security_group = security_groups[0] if len(security_groups) else None
```

```
        if not jenkins_security_group:
            jenkins_security_group =
self.ec2.create_security_group(GroupName=self.JENKINS_SECURITY_GROUP_NA
ME, Description='Security group for jenkins web and ssh access')
            self.ec2.authorize_security_group_ingress(
            GroupId=jenkins_security_group['GroupId'],
            IpPermissions=[
                {
                'IpProtocol': 'tcp',
                'FromPort': 8080,
                'ToPort': 8080,
                'IpRanges': [{'CidrIp': '0.0.0.0/0'}]
                },
                {
                'IpProtocol': 'tcp',
                'FromPort': 22,
                'ToPort': 22,
                'IpRanges': [{'CidrIp': f'{IPAddr}/32'}]
                }
            ])
            print(f"Created security group -
{self.JENKINS_SECURITY_GROUP_NAME}, GroupId -
{jenkins_security_group['GroupId']}")
            return jenkins_security_group['GroupId']
        print(f"Security group {self.JENKINS_SECURITY_GROUP_NAME} already
exists")
        return jenkins_security_group['GroupId']
    def create_jenkins_instance(self, security_group_id, image_id, role_arn):
        instance = self.ec2.run_instances(
            BlockDeviceMappings=[
```

```
        {
            'DeviceName': '/dev/xvda',
            'Ebs': {
                'DeleteOnTermination': True,
                'VolumeSize': 16,
                'VolumeType': 'gp2',
            },
        },
    ],
    ImageId=image_id,
    MinCount=1,
    MaxCount=1,
    KeyName=self.JENKINS_KEYPAIR_NAME,
    SecurityGroupIds=[security_group_id],
    IamInstanceProfile={ 'Arn': role_arn, 'Name': self.JENKINS_ROLE_NAME },
    TagSpecifications=[
        {
            'ResourceType': "instance",
            'Tags': [
                {
                    'Key': 'Name',
                    'Value': 'Jenkins Instance Test'
                },
            ]
        },
    ]
)["Instances"][0]
waiter = self.ec2.get_waiter('instance_running')
waiter.wait(InstanceIds=[instance["InstanceId"]])
return instance
```

```
    def get_ami_id(self):
        image = self.ec2.describe_images(Owners=["amazon"], Filters=[
            {
                "Name": "name",
                "Values": ["amzn2-ami-kernel-5.10-hvm-2.0.20221004.0-x86_64-gp2"]
            }
        ])["Images"][0]
        return image["ImageId"]
    def connect_and_install(self):
        key =
paramiko.RSAKey.from_private_key_file(f".\{self.JENKINS_KEYPAIR_NAME}")
        connection = paramiko.SSHClient()
        connection.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        connection.connect(hostname=jenkins_initializer.instance["IPv4Address"],
username="ec2-user", pkey=key, allow_agent=False, look_for_keys=False)
        connection.exec_command("sudo yum install -y amazon-linux-extras") # add
amazon-linux-extras package repository for docker and dotnet installation
        connection.exec_command("sudo yum update -y") # update yum references
        connection.exec_command("sudo wget -O /etc/yum.repos.d/jenkins.repo \
https://pkg.jenkins.io/redhat-stable/jenkins.repo") # download jenkins installation
        connection.exec_command("sudo rpm --import https://pkg.jenkins.io/redhat-
stable/jenkins.io.key") # import a key file from Jenkins-CI to enable installation from
the package
        connection.exec_command("sudo yum upgrade")
        connection.exec_command("sudo amazon-linux-extras install java-openjdk11 -y")
# install Java for Jenkins
        connection.exec_command("sudo yum install jenkins -y") # install Jenkins
        connection.exec_command("sudo systemctl enable jenkins") # enable Jenkins
service to start automatically
        connection.exec_command("sudo systemctl start jenkins") # start Jenkins service
```

```
        sftp_client = connection.open_sftp()
        sftp_client.put("/jenkins", "/var/lib/jenkins") # copy Jenkins home directory with
plugins and configurations
        connection.exec_command("sudo yum -y install docker") # install docker
        connection.exec_command("sudo systemctl enable docker") # enable docker
automatic startup
        connection.exec_command("sudo systemctl start docker") # start docker service
        connection.exec_command("sudo yum install dotnet-sdk-6.0") # start docker
service
        connection.close()
class SonarqubeInitializer:
    ec2 = boto3.client('ec2')
    iam = boto3.client('iam')
    instance = None
    SONARQUBE_KEYPAIR_NAME = 'SonarqubeKeyPair'
    SONARQUBE_SECURITY_GROUP_NAME = 'SonarqubeSecurityGroup'
    def __init__(self):
        image_id = self.get_ami_id()
        self.create_key_pair()
        security_group_id = self.create_security_group()
        self.instance = self.create_instance(security_group_id, image_id)
        self.connect_and_install()
    def create_key_pair(self):
        key_pairs = self.ec2.describe_key_pairs()["KeyPairs"]
        if not any(element["KeyName"] in self.SONARQUBE_KEYPAIR_NAME for
element in key_pairs):
            response =
self.ec2.create_key_pair(KeyName=self.SONARQUBE_KEYPAIR_NAME)
            with open(f"{self.SONARQUBE_KEYPAIR_NAME}.ppk", "w") as f:
                print(response, file=f)
```

```
        return
      print(f"Key pair: {self.SONARQUBE_KEYPAIR_NAME} already exists")
    def create_security_group(self):
      security_groups = self.ec2.describe_security_groups()["SecurityGroups"]
      security_groups = [group for group in security_groups if group["GroupName"] ==
self.SONARQUBE_SECURITY_GROUP_NAME]
      sonarqube_security_group = security_groups[0] if len(security_groups) else None
      if not sonarqube_security_group:
        sonarqube_security_group =
self.ec2.create_security_group(GroupName=self.SONARQUBE_SECURITY_GROUP
_NAME, Description='Security group for sonarqube web and ssh access')
        self.ec2.authorize_security_group_ingress(
        GroupId=sonarqube_security_group['GroupId'],
        IpPermissions=[
          {
          'IpProtocol': 'tcp',
          'FromPort': 9000,
          'ToPort': 9000,
          'IpRanges': [{'CidrIp': '0.0.0.0/0'}]
          },
          {
          'IpProtocol': 'tcp',
          'FromPort': 22,
          'ToPort': 22,
          'IpRanges': [{'CidrIp': f'{IPAddr}/32'}]
          }
        ])
        print(f"Created security group -
{self.SONARQUBE_SECURITY_GROUP_NAME}, GroupId -
{sonarqube_security_group['GroupId']}")
```

```
            return sonarqube_security_group['GroupId']
        print(f"Security group {self.SONARQUBE_SECURITY_GROUP_NAME}
already exists")
        return sonarqube_security_group['GroupId']
    def create_instance(self, security_group_id, image_id):
        instance = self.ec2.run_instances(
            BlockDeviceMappings=[
                {
                    'DeviceName': '/dev/xvda',
                    'Ebs': {
                        'DeleteOnTermination': True,
                        'VolumeSize': 16,
                        'VolumeType': 'gp2',
                    },
                },
            ],
            ImageId=image_id,
            MinCount=1,
            MaxCount=1,
            KeyName=self.SONARQUBE_KEYPAIR_NAME,
            SecurityGroupIds=[security_group_id],
            TagSpecifications=[
                {
                    'ResourceType': "instance",
                    'Tags': [
                        {
                            'Key': 'Name',
                            'Value': 'Sonarqube Instance Test'
                        },
                    ]
```

```
            },
        ]
    )["Instances"][0]
    waiter = self.ec2.get_waiter('instance_running')
    waiter.wait(InstanceIds=[instance["InstanceId"]])
    return instance
  def get_ami_id(self):
    image = self.ec2.describe_images(Owners=["amazon"], Filters=[
        {
            "Name": "name",
            "Values": ["amzn2-ami-kernel-5.10-hvm-2.0.20221004.0-x86_64-gp2"]
        }
    ])["Images"][0]
    return image["ImageId"]
  def connect_and_install(self):
    key =
paramiko.RSAKey.from_private_key_file(f".\{self.SONARQUBE_KEYPAIR_NAME
}")
    connection = paramiko.SSHClient()
    connection.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    connection.connect(hostname=jenkins_initializer.instance["IPv4Address"],
username="ec2-user", pkey=key, allow_agent=False, look_for_keys=False)
    connection.exec_command("sudo yum install -y amazon-linux-extras") # add
amazon-linux-extras package repository for docker and dotnet installation
    connection.exec_command("sudo yum update -y") # update yum references
    connection.exec_command("sudo curl -L
https://github.com/docker/compose/releases/download/1.22.0/docker-compose-$(uname
-s)-$(uname -m) -o /usr/local/bin/docker-compose") # download jenkins installation
    connection.exec_command("sudo chmod +x /usr/local/bin/docker-compose") #
import a key file from Jenkins-CI to enable installation from the package
```

```
        connection.exec_command("sudo yum upgrade")

        connection.exec_command("sudo amazon-linux-extras install java-openjdk11 -y")
# install Java for Jenkins
        sftp = connection.open_sftp()
        sftp.put("sonarqube/postgresql", "/var/lib/postgresql") # load sonarqube data to aws
instance
        connection.exec_command("""sudo nano docker-compose.yml
                        version: "3"
                        services:
                        SonarQube:
                          image: SonarQube:community
                          depends_on:
                          - db
                          environment:
                          SONAR_JDBC_URL: jdbc:postgresql://db:5432/sonar
                          SONAR_JDBC_USERNAME: sonartest
                          SONAR_JDBC_PASSWORD: sonartest
                          volumes:
                          - SonarQube_data:/opt/SonarQube/data
                          - SonarQube_extensions:/opt/SonarQube/extensions
                          - SonarQube_logs:/opt/SonarQube/logs
                          ports:
                          - "9000:9000"
                        db:
                          image: postgres:12
                          environment:
                          POSTGRES_USER: sonartest
                          POSTGRES_PASSWORD: sonartesrt
                          volumes:
```

```
                    - postgresql:/var/lib/postgresql
                    - postgresql_data:/var/lib/postgresql/data
                  volumes:
                  SonarQube_data:
                  SonarQube_extensions:
                  SonarQube_logs:
                  postgresql:
                  postgresql_data:""") # create docker compose file for Sonarqube
        connection.exec_command("docker-compose up") # run sonarqube
        connection.close()
class ECSInitializer:
    ec2 = boto3.client('ec2')
    autoscaling = boto3.client('autoscaling')
    ecs = boto3.client('ecs')
    CLUSTER_NAME = "StoreAppCluster"
    SERVICE_NAME = "StoreAppService"
    TASK_DEFINITION_NAME = "StoreAppTask"
    INSTANCE_KEYPAIR_NAME = "StoreAppInstanceKeyPair"
    INSTANCE_SECURITY_GROUP_NAME = "InstanceSecurityGroup"
    INSTANCE_LAUNCH_CONFIGURATION_NAME =
"InstanceLaunchConfiguration"
    AUTOSCALING_GROUP_NAME = "AppStoreAutoscalingGroup"
    def __init__(self):
        self.create_key_pair()
        self.create_security_group()
        self.create_launch_configuration()
        self.create_autoscaling_group()
        self.create_cluster()
        self.create_task_definition()
        self.create_service()
```

```
def create_service(self):
    return self.ecs.create_service(
        cluster=self.CLUSTER_NAME,
        serviceName=self.SERVICE_NAME,
        taskDefinition=self.TASK_DEFINITION_NAME,
        launchType='EC2',
        deploymentConfiguration={
            'deploymentCircuitBreaker': {
                'enable': True,
                'rollback': False
            },
            'maximumPercent': 200,
            'minimumHealthyPercent': 50
        },
        placementConstraints=[
            {
                'type': 'distinctInstance'
            },
        ],
        placementStrategy=[
            {
                'type': 'spread',
                'field': 'instanceId'
            },
        ],
        schedulingStrategy='REPLICA',
        deploymentController={
            'type': 'ECS'
        },
        enableExecuteCommand=True,
```

```
    )["Services"][0]
def create_cluster(self):
    return self.ecs.create_cluster(
        clusterName=self.CLUSTER_NAME,
    )["Clusters"][0]
def create_task_definition(self):
    return self.ecs.register_task_definition(
        family=self.TASK_DEFINITION_NAME,
        containerDefinitions=[
            {
                "ipcMode": None,
                "executionRoleArn": None,
                "containerDefinitions": [
                    {
                    "dnsSearchDomains": None,
                    "environmentFiles": None,
                    "logConfiguration": None,
                    "entryPoint": [],
                    "portMappings": [
                        {
                        "hostPort": 80,
                        "protocol": "tcp",
                        "containerPort": 80
                        }
                    ],
                    "command": [],
                    "linuxParameters": None,
                    "cpu": 0,
                    "environment": [],
                    "resourceRequirements": None,
```

"ulimits": None,
"dnsServers": None,
"mountPoints": [],
"workingDirectory": None,
"secrets": None,
"dockerSecurityOptions": None,
"memory": 900,
"memoryReservation": None,
"volumesFrom": [],
"stopTimeout": None,
"image": "892461827874.dkr.ecr.us-east-1.amazonaws.com/store-app:latest",
"startTimeout": None,
"firelensConfiguration": None,
"dependsOn": None,
"disableNetworking": None,
"interactive": None,
"healthCheck": None,
"essential": True,
"links": None,
"hostname": None,
"extraHosts": None,
"pseudoTerminal": None,
"user": None,
"readonlyRootFilesystem": None,
"dockerLabels": None,
"systemControls": None,
"privileged": None,
"name": "store-app"
}

],
"placementConstraints": [],
"memory": "900",
"taskRoleArn": None,
"compatibilities": [
"EXTERNAL",
"EC2"
],
"taskDefinitionArn": "arn:aws:ecs:us-east-1:892461827874:task-definition/StoreAppTask:38",
"family": "StoreAppTask",
"requiresAttributes": [
{
"targetId": None,
"targetType": None,
"value": None,
"name": "com.amazonaws.ecs.capability.ecr-auth"
}
],
"pidMode": None,
"requiresCompatibilities": [
"EC2"
],
"networkMode": None,
"runtimePlatform": None,
"cpu": "1024",
"revision": 38,
"status": "ACTIVE",
"inferenceAccelerators": None,
"proxyConfiguration": None,

```
                "volumes": []
            }
        ]
    )["TaskDefinitions"][0]
def create_autoscaling_group(self):
    return self.autoscaling.create_auto_scaling_group(
        AutoScalingGroupName=self.AUTOSCALING_GROUP_NAME,

LaunchConfigurationName=self.INSTANCE_LAUNCH_CONFIGURATION_NAME,
        MinSize=1,
        MaxSize=5,
        DesiredCapacity=2,
        AvailabilityZones=[
            'us-east-1',
        ],
        HealthCheckType='EC2',
    )["AutoscalingGroups"][0]
def create_launch_configuration(self):
    return self.autoscaling.create_launch_configuration(
        LaunchConfigurationName='StoreAppClusterLaunchConfiguration',
        KeyName=self.INSTANCE_KEYPAIR_NAME,
        SecurityGroups=[
            self.INSTANCE_SECURITY_GROUP_NAME
        ],
        InstanceType='t2.micro',
        BlockDeviceMappings=[
            {
                'DeviceName': '/dev/xvda',
                'Ebs': {
                    'DeleteOnTermination': True,
```

```
                'VolumeSize': 16,
                'VolumeType': 'gp2',
            },
        },
    ],
    InstanceMonitoring={
        'Enabled': True
    },
    AssociatePublicIpAddress=True,
)["LaunchConfigurations"][0]
image = self.ec2.describe_images(Owners=["amazon"], Filters=[
    {
        "Name": "name",
        "Values": ["amzn2-ami-ecs-hvm-2.0.20221102-x86_64-ebs"]
    }
])["Images"][0]
return image["ImageId"]
def create_key_pair(self):
    key_pairs = self.ec2.describe_key_pairs()["KeyPairs"]
    if not any(element["KeyName"] in self.INSTANCE_KEYPAIR_NAME for
element in key_pairs):
        response =
self.ec2.create_key_pair(KeyName=self.INSTANCE_KEYPAIR_NAME)
        with open(f"{self.INSTANCE_KEYPAIR_NAME}.ppk", "w") as f:
            print(response, file=f)
        return
    print(f"Key pair: {self.INSTANCE_KEYPAIR_NAME} already exists")
def create_security_group(self):
    security_groups = self.ec2.describe_security_groups()["SecurityGroups"]
```

```
security_groups = [group for group in security_groups if group["GroupName"] ==
self.INSTANCE_SECURITY_GROUP_NAME]
sonarqube_security_group = security_groups[0] if len(security_groups) else None
if not sonarqube_security_group:
    sonarqube_security_group =
self.ec2.create_security_group(GroupName=self.INSTANCE_SECURITY_GROUP_N
AME, Description='Security group for sonarqube web and ssh access')
    self.ec2.authorize_security_group_ingress(
    GroupId=sonarqube_security_group['GroupId'],
    IpPermissions=[
        {
        'IpProtocol': 'tcp',
        'FromPort': 9000,
        'ToPort': 9000,
        'IpRanges': [{'CidrIp': '0.0.0.0/0'}]
        },
        {
        'IpProtocol': 'tcp',
        'FromPort': 22,
        'ToPort': 22,
        'IpRanges': [{'CidrIp': f'{IPAddr}/32'}]
        }
    ])
    print(f"Created security group -
{self.INSTANCE_SECURITY_GROUP_NAME}, GroupId -
{sonarqube_security_group['GroupId']}")
    return sonarqube_security_group['GroupId']
```

```
        print(f"Security group {self.INSTANCE_SECURITY_GROUP_NAME} already
exists")
        return sonarqube_security_group['GroupId']
class PipelineInitializer:
    s3 = boto3.client("s3")
    ecr = boto3.client("ecr")
    codecommit = boto3.client("codecommit")
    codepipeline = boto3.client("codepipeline")
    ECR_REPO_NAME = "store-app"
    PIPELINE_NAME = "StoreAppPipeline"
    PIPELINE_BUCKET_NAME = "PipelineBucket"
    def __init__(self):
        self.create_bucket()
        self.create_codecommit_repository()
        self.create_ecr_repository()
        self.create_pipeline()
    def create_ecr_repository(self):
        return self.ecr.create_repository(
            repositoryName=self.ECR_REPO_NAME,
            imageScanningConfiguration={
                'scanOnPush': True
            }
        )["Repositories"][0]
    def create_codecommit_repository(self):
        return self.codecommit.create_repository(
            repositoryName='StoreApp',
        )["Repositories"][0]
    def create_bucket(self):
        self.s3.create_bucket(Bucket=self.PIPELINE_BUCKET_NAME)
    def create_pipeline(self):
```

```
return self.codepipeline.create_pipeline(
    pipeline={
        'name': self.PIPELINE_NAME,
        'roleArn': 'string',
        'artifactStore': {
            'type': 'S3',
            'location': self.PIPELINE_BUCKET_NAME
        },
        'stages': [
            {
                'name': 'Source',
                'actions': [
                    {
                        'name': 'CodeCommit',
                        'actionTypeId': {
                            'category': 'Source',
                            'owner': 'AWS',
                            'provider': 'CodeCommit'
                        },
                        'runOrder': 1,
                        'configuration': {
                            'RepositoryName': 'StoreApp',
                            'BranchName': 'master'
                        },
                        'outputArtifacts': [
                            {
                                'name': 'SourceArtifact'
                            }
                        ],
                        'namespace': 'SourceVariables'
```

```
                    },
                ]
            },
            {
                'name': 'Build',
                'actions': [
                    {
                        'name': 'JenkinsBuild',
                        'actionTypeId': {
                            'category': 'Build',
                            'owner': 'AWS',
                            'provider': 'Add Jenkins'
                        },
                        'runOrder': 1,
                        'configuration': {
                            'Provider name': 'Jenkins Provider',
                            'Server URL':
f"{jenkins_initializer.instance['PrivateIPv4Address']}:8080",
                            'Project name': "StoreApp"
                        },
                        'outputArtifacts': [
                            {
                                'name': 'ImageDefinition'
                            }
                        ],
                        'inputArtifacts': [
                            {
                                'name': 'SourceArtifact'
                            }
                        ],
```

```
            },
        ]
    },
    {
        'name': 'Deploy',
        'actions': [
            {
                'name': 'Deploy',
                'actionTypeId': {
                    'category': 'Deploy',
                    'owner': 'AWS',
                    'provider': 'Amazon ECS'
                },
                'runOrder': 1,
                'configuration': {
                    'Cluster name': ecs_initializer.CLUSTER_NAME,
                    'Service name': ecs_initializer.SERVICE_NAME,
                    'Project name': "StoreApp"
                },
                'outputArtifacts': [
                    {
                        'name': 'ImageDefinition'
                    }
                ],
                'inputArtifacts': [
                    {
                        'name': 'ImageDefinition'
                    }
                ],
            },
```

```
                    ]
                }
            ],
        },
    )
jenkins_initializer: JenkinsInitializer = JenkinsInitializer()
sonarqube_initializer: SonarqubeInitializer = SonarqubeInitializer()
ecs_initializer: ECSInitializer = ECSInitializer()
```

ДОДАТОК К

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Програмний засіб безпечної контейнеризації веб-додатку

Тип роботи: магістерська кваліфікаційна робота
(БДР, МКР)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 97.5% Схожість 2.5%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.
- ☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- ☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку ____________ Захарченко С.М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи ____________ Рудь Л.І.
(підпис) (прізвище, ініціали)

Керівник роботи ____________ Войцеховська О.В.
(підпис) (прізвище, ініціали)