ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Кваліфікаційна робота

на тему: **«СИСТЕМА CONTINUOUS INTEGRATION/CONTINUOUS DELIVERY У КОРПОРАТИВНИХ СУБ'ЄКТАХ НА МОВІ ПРОГРАМУВАННІ JAVA»**

на здобуття освітнього ступеня магістра
зі спеціальності __122 Комп'ютерні науки_____
(код, найменування спеціальності)
освітньо-професійної програми __Комп'ютерні науки _____
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ ___Чистяков Богдан _____
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач вищої освіти гр. КНДМ-62_____

_____Чистяков Богдан_____
(Ім'я, ПРІЗВИЩЕ)

Керівник: __д.т.н., доцент Ольга Зінченко__
Науковий ступінь, вчене звання *(Ім'я, ПРІЗВИЩЕ)*

Рецензент: _____
Науковий ступінь, вчене звання *(Ім'я, ПРІЗВИЩЕ)*

Київ - 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально–науковий інститут інформаційних технологій

Кафедра Комп'ютерних наук
Ступінь вищої освіти − «Магістр»
Спеціальність підготовки − «122 Комп'ютерні науки»
Освітньо-професійна програма − «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерних наук
_____Віктор Вишнівський
"____" _____2024 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

_____Чистяков Богдан Олександрович
_____
(прізвище, ім'я, по батькові)

1.Тема кваліфікаційної роботи: «Система Continuous Integration/Continuous Delivery у корпоративних суб'єктах на мові програмування Java»
Керівник кваліфікаційної роботи __Ольга Зінченко, к.т.н., доцент._____,
(ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу № 320 від 15.10.2024р.

2. Строк подання студентом роботи 15.12.2024 р.
3. Вихідні дані до роботи: CI/CD, Jenkins, pipeline, DevOps, Науково-технічна література, пов'язана з методикою DevOps.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз концепцій Continuous Integration та Continuous Delivery у корпоративному середовищі
2. Розробка та налаштування CI/CD-процесу для корпоративного Java-додатка

3. Оптимізація та масштабування CI/CD-процесу для корпоративного середовища
4. Дослідження безпеки та ефективності CI/CD у корпоративних Java-проєктах

5. Перелік ілюстративного матеріалу: *презентація*
1. Мета роботи, об`єкт та предмет дослідження
2. Основні поняття та терміни
3. Огляд існуючих рішень для CI/CD
4. Технології та інструменти для роботи з Java
5. Архітектура системи CI/CD для корпоративних суб'єктів
6. Процес побудови CI/CD пайплайна
7. Автоматизація тестування в Java
8. Контейнеризація та оркестрація Java-додатків
9.Безпека в CI/CD процесах
10. Кейси впровадження CI/CD у корпоративних проектах
11. Результати впровадження CI/CD на базі Java
12. Перспективи розвитку CI/CD у корпоративному середовищі
13. Висновки

6. Дата видачі завдання 10.09.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	05.10 - 10.10.2024	вик.
2	Теоретичний огляд систем CI/CD	11.10 - 20.10.2024	вик.
3	Аналіз застосування CI/CD у корпоративних суб'єктах	21.10 - 31.10.2024	вик.
4	Інструменти та технології для CI/CD у Java-середовищі	01.11 - 10.11.2024	вик.
5	Розробка та впровадження CI/CD системи для Java-проекту	11.11 - 25.11.2024	вик.
6	Вступ, висновки, реферат	26.11 - 05.12.2024	вик.
7	Розробка демонстраційних креслень	21.12 - 24.12.2024	вик.

Здобувач вищої освіти _____ __Чистяков Богдан____
(підпис) (Ім'я, ПРІЗВИЩЕ)

Керівник
кваліфікаційної роботи _____ ___Ольга Зінченко_____
(підпис) (Ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 67 стор., 23 рис., 5 джерел

Мета роботи - Дослідити процеси Continuous Integration (CI) та Continuous Delivery (CD) у корпоративних середовищах, які розробляють програмні продукти на Java, виявити переваги, виклики та запропонувати практичні рекомендації щодо їх впровадження.

Об'єкт дослідження - Процеси CI/CD у корпоративних суб'єктах, що розробляють програмне забезпечення з використанням мови Java, та інструменти, які застосовуються для автоматизації цих процесів (Jenkins, Docker, SonarQube, Maven тощо).

Предмет дослідження - процеси автоматизації інтеграції, тестування та доставки програмного забезпечення у корпоративних середовищах на мові програмування Java.

Короткий зміст роботи:

У роботі розглядаються основи CI/CD, їх переваги для корпоративних розробників Java-програм, включаючи підвищення швидкості випуску продуктів та зменшення кількості помилок. Проаналізовано популярні інструменти для реалізації CI/CD, описано процес їх налаштування та інтеграції в корпоративне середовище. Наведено приклади успішного впровадження CI/CD, вказано на ключові виклики, зокрема початкові інвестиції у навчання персоналу та інфраструктуру. Запропоновано рекомендації щодо ефективного використання CI/CD для покращення якості та швидкості розробки.

КЛЮЧОВІ СЛОВА: CI/CD, Java, Jenkins, Docker, Spring Boot, Maven, тестування, розгортання, якість коду, програмне забезпечення.

ABSTRACT

Text part of the master's qualification work: 67 pages, 23 pictures, 5 sources

The purpose of the work - To investigate the processes of Continuous Integration (CI) and Continuous Delivery (CD) in corporate environments that develop Java software products, to identify the benefits, challenges and offer practical recommendations for their implementation.

Object of study - CI/CD processes in corporate entities that develop software using Java and the tools used to automate these processes (Jenkins, Docker, SonarQube, Maven, etc.).

The subject of the research - the processes of automating the integration, testing and delivery of software in corporate environments in the Java programming language.

Summary of the work:

The paper discusses the basics of CI/CD, their benefits for corporate Java developers, including increasing the speed of product release and reducing the number of errors. Popular tools for CI/CD implementation are analyzed, the process of their configuration and integration into the corporate environment is described. Examples of successful CI/CD implementation are provided, and key challenges, including initial investments in staff training and infrastructure, are identified. Recommendations for the effective use of CI/CD to improve the quality and speed of development are offered.

KEYWORDS: CI/CD, Java, Jenkins, Docker, Spring Boot, Maven, testing, deployment, code quality, software.

ЗМІСТ

ВСТУП

Сучасний розвиток інформаційних технологій та швидка еволюція програмного забезпечення вимагають від компаній постійного вдосконалення своїх процесів розробки. Однією з ключових практик, яка дозволяє знижувати час виходу продукту на ринок і підвищувати його якість, є впровадження систем Continuous Integration (CI) та Continuous Delivery (CD). Ці методології забезпечують автоматизацію процесів інтеграції, тестування та доставки програмного забезпечення, що є особливо важливим для корпоративних суб'єктів, які займаються розробкою складних систем і програмних забезпечень на мові програмування Java.

Java, як один з найпоширеніших мов програмування, забезпечує багатий набір інструментів та бібліотек, що поєднуються з CI/CD практиками. Від використання автоматичних тестів і зборок до безперервного деплою, Java середовище пропонує величезні можливості для оптимізації роботи команд розробників. У рамках даної роботи буде проведено аналіз сучасних підходів до реалізації CI/CD у корпоративному середовищі на прикладі програмного забезпечення Jenkins, виявлені переваги та недоліки їх впровадження, а також розроблені рекомендації щодо ефективного використання цих методологій.

Метою цієї магістерської роботи є дослідити актуальні практики Continuous Integration/Continuous Delivery у контексті розробки програмного забезпечення на мові Java, а також визначити, яким чином ці практики можуть позитивно вплинути на якість та швидкість впровадження продуктів у корпоративних суб'єктах. У межах роботи розглянуто як теоретичні аспекти, так і практичні кейси, що дозволить сформувати комплексне уявлення про цю важливу тему.

1 ОГЛЯД CI/CD ПРОЦЕСУ

1.1 Загальний опис CI/CD

Постійна інтеграція (CI) та безперервна доставка (CD) являють собою набір підходів і практик, які дають змогу командам розробників частіше та надійніше впроваджувати зміни в код. Цей процес часто називають CI/CD конвеєром (pipeline).

CI (Continuous Integration) — це підхід до кодування та набір практик, які заохочують розробників регулярно додавати невеликі зміни до системи контролю версій (наприклад, Git). Оскільки сучасні додатки зазвичай розробляються з використанням різних платформ і інструментів, командам потрібні механізми для інтеграції та перевірки цих змін.
Головна технічна мета CI — забезпечити послідовний і автоматизований процес створення, тестування та упаковки програм. Це дозволяє командам частіше вносити зміни в код, що підвищує якість програмного забезпечення і покращує співпрацю.

CD (Continuous Delivery) — це наступний етап після CI, який автоматизує доставку додатків до різних середовищ. Команди зазвичай працюють із декількома середовищами (наприклад, розробницьким, тестовим і виробничим), і CD забезпечує автоматизоване просування змін між ними.

Інструменти CI/CD також дозволяють налаштовувати параметри, специфічні для кожного середовища, і автоматизують процеси, пов’язані з оновленням серверів, баз даних та інших компонентів.

1.2 Переваги CI/CD

Постійна інтеграція є підходом до розробки, що базується на механізмах процесів і автоматизації. Розробники часто фіксують свій код у системі контролю версій (зазвичай не рідше одного разу на день). Такий підхід дозволяє швидше знаходити помилки та проблеми з якістю коду, оскільки вони виявляються на невеликих змінах, а не у великих оновленнях, що накопичуються протягом тривалого часу.

Крім того, регулярні фіксації знижують ризик конфліктів між змінами від різних розробників. Команди, які впроваджують CI, зазвичай починають із налаштування системи контролю версій і визначення практик внесення змін. Хоча зміни перевіряються регулярно, нові функції та виправлення можуть реалізовуватися як у короткі, так і в довші періоди часу.

1.3 Особливості CD

Безперервна доставка автоматизує процес передачі програм у цільові середовища. Зазвичай команди мають кілька середовищ для розробки та тестування, де зміни проходять перевірку перед розгортанням у виробниче середовище.

Типовий CD-конвеєр включає кілька етапів:

- Завантаження коду з системи контролю версій і збірка.
- Виконання автоматизованих інфраструктурних змін (наприклад, налаштування хмарного середовища).
- Перенесення коду в цільове середовище.
- Налаштування змінних середовища й конфігурацій для серверів, API та баз даних.

- Виконання необхідних дій для перезапуску служб або оновлення компонентів.
- Автоматичне тестування та відкат у разі помилок.

Прикладом може бути використання Jenkins, де етапи конвеєра (збірка, тестування, розгортання) описуються в спеціальному конфігураційному файлі. Секретні ключі, сертифікати й параметри середовища також визначаються у файлі та використовуються в різних етапах.

Більш складні CD-конвеєри можуть включати синхронізацію даних, архівування ресурсів або виправлення вразливостей. Інструменти CI/CD, такі як Jenkins, CircleCI, AWS CodeBuild, Azure DevOps, Travis CI, підтримують інтеграцію з безліччю платформ через плагіни. Наприклад, Jenkins пропонує понад 1500 плагінів для розширення функціональності.

2 ОГЛЯД ІНСТРУМЕНТАРІЇВ CI/CD

2.1 Jenkins

Jenkins – це сервер автоматизації з відкритим кодом, що забезпечує централізовану підтримку процесів CI. Це Java-додаток, доступний для Windows, macOS та Unix-подібних систем. Завдяки великій кількості доступних плагінів Jenkins дозволяє автоматизувати процеси створення, тестування та розгортання програмного забезпечення.

Основні функції:

- Просте встановлення та підтримка на різних операційних системах.
- Зручний інтерфейс для налаштування та управління.
- Широкі можливості розширення через плагіни.
- Інтуїтивне налаштування середовища через графічний інтерфейс.
- Підтримка розподілених збірок у форматі master-slave.
- Візуалізація процесів у вигляді графіків.
- Можливість виконання команд оболонки та Windows-скриптів.
- Підтримка сповіщень про статус збірок.

Ціна: повністю безкоштовний (рис. 2.1)

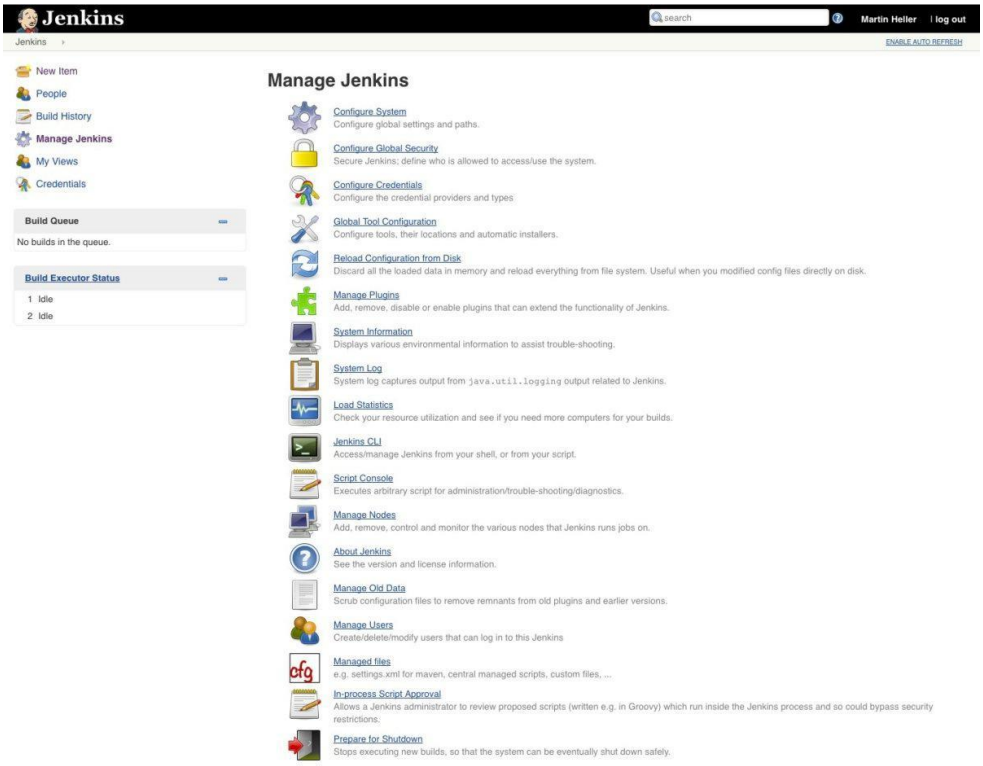


Рисунок 2.1 - Інтерфейс Jenkins

2.2 CircleCI

CircleCI – сучасна CI/CD-платформа, яку можна використовувати у хмарі чи локально. Платформа автоматизує збірку, тестування та розгортання програмного забезпечення. CircleCI інтегрується з GitHub, GitHub Enterprise та Bitbucket, запускаючи збірки при комітах.

Основні функції:

- Інтеграція з GitHub, GitHub Enterprise і Bitbucket.
- Використання контейнерів або віртуальних машин для збірок.
- Пряма налагодження збірок.
- Автоматичне розпаралелювання процесів.

- Швидке тестування і виконання.
- Інтеграція з чатами та сповіщеннями.
- Гнучке налаштування під різні потреби.
- Безперервне розгортання з урахуванням гілок репозиторію.

Ціна: базовий план для Linux дозволяє запускати одне завдання безкоштовно. Відкриті проєкти отримують додаткові безкоштовні контейнери. Деталі вартості доступні після реєстрації (рис. 2.2)

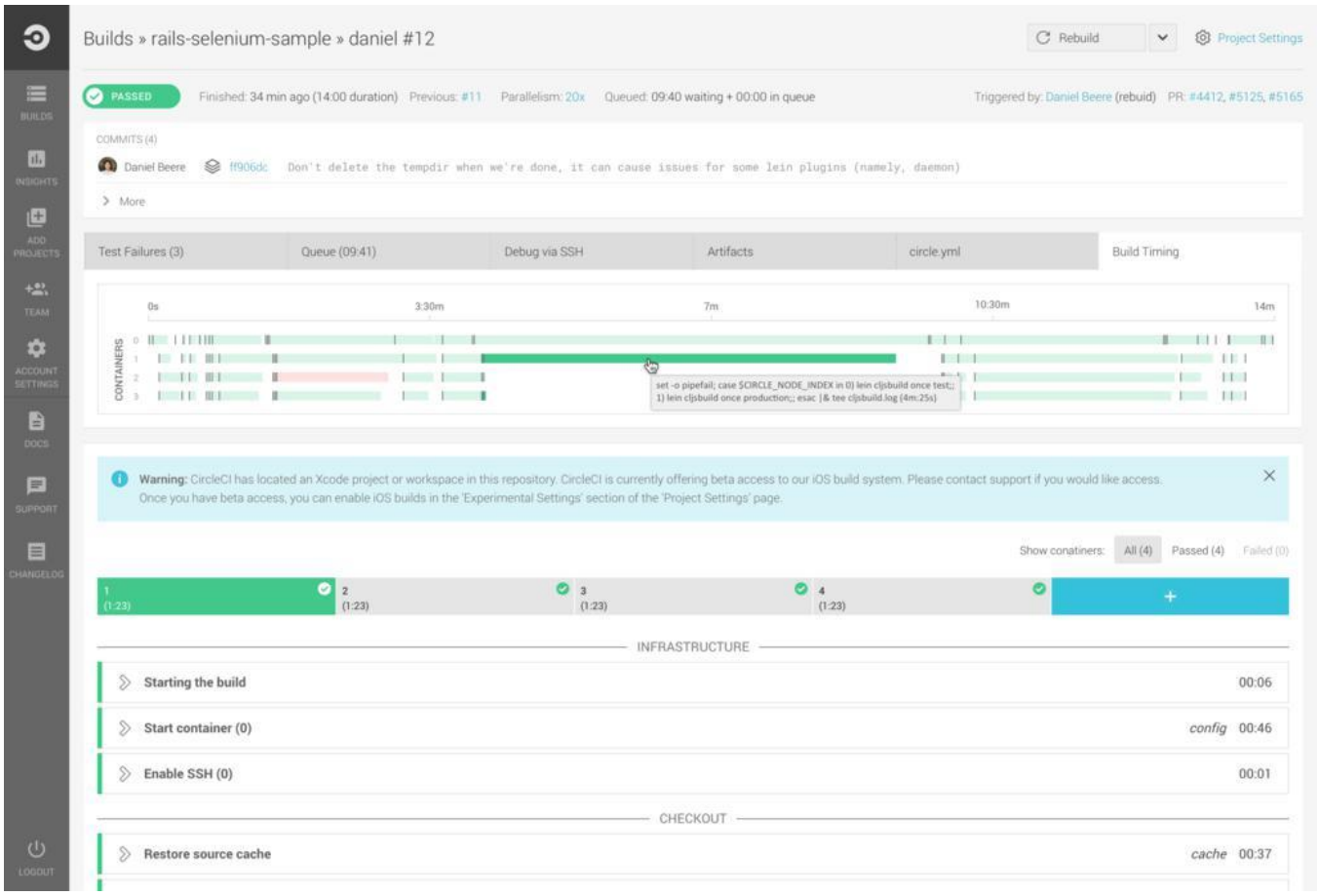


Рисунок 2.2 - Інтерфейс CircleCI

2.3 Bamboo

Bamboo – це сервер CI/CD, що автоматизує управління випусками програм, створюючи конвеєри безперервної доставки. Він об'єднує створення, тестування, маркування версій і розгортання програм.

Основні функції:

- Підтримка до 100 віддалених агентів.
- Паралельний запуск тестів для швидкого зворотного зв’язку.
- Інтеграція зі сховищами Git, Mercurial і SVN, автоматичне налаштування CI для нових гілок.
- Тригери збірок на основі змін у сховищах або подій у Bitbucket.
- Розширене управління дозволами для різних середовищ.

Ціна: залежить від кількості агентів. Більше агентів дозволяють виконувати більше процесів одночасно (рис. 2.3)

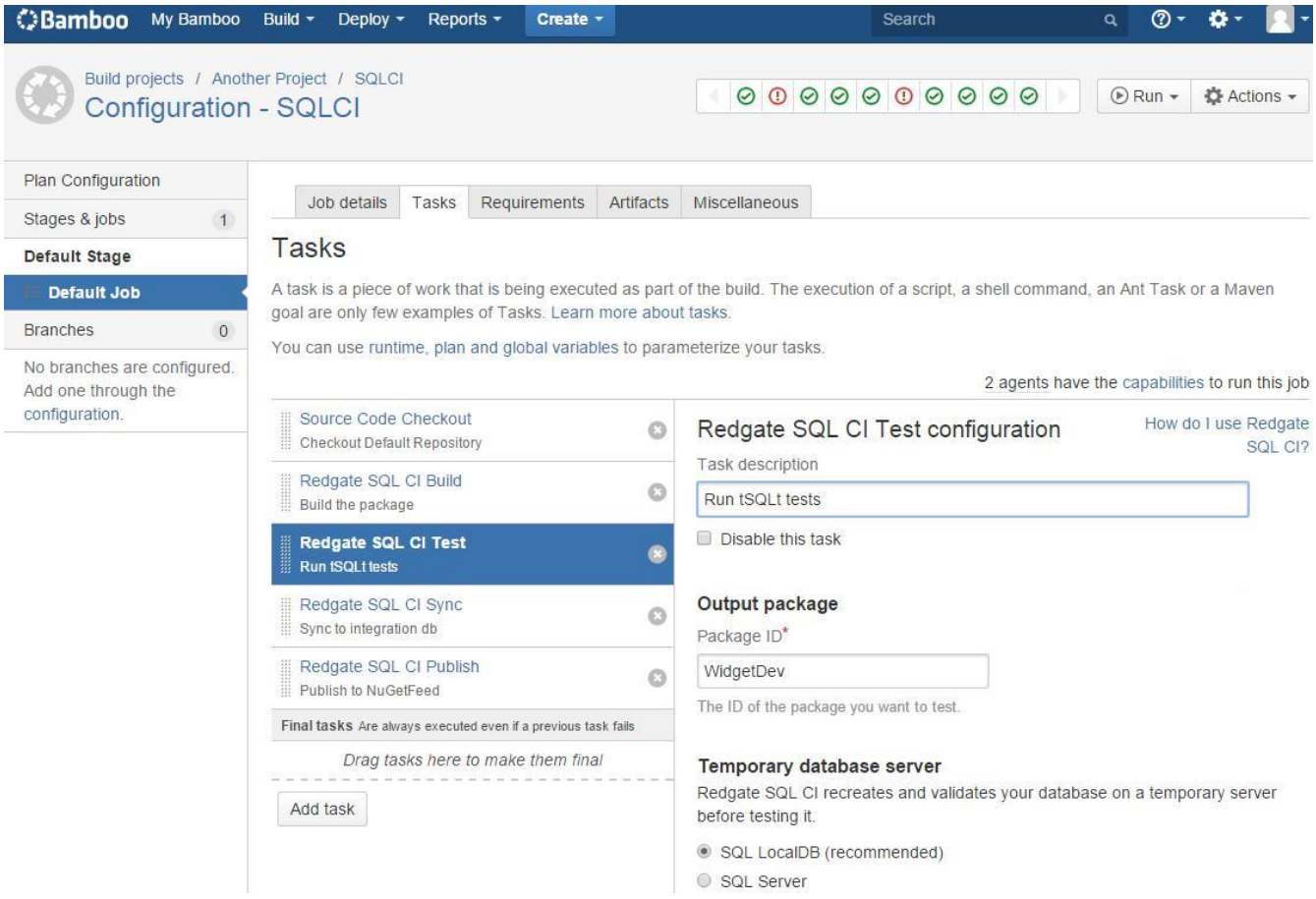


Рисунок 2.3 - Інтерфейс Bamboo

2.4 TeamCity

TeamCity – інструмент CI/CD від JetBrains, що підтримує розробку й розгортання різноманітних проєктів. Він працює у середовищі Java і підтримує інтеграцію з Visual Studio, .NET, Azure DevOps і Jira Software Cloud.

Основні функції:

- Повторне використання параметрів і конфігурацій з батьківського проєкту в підпроєктах.
- Паралельний запуск у різних середовищах.
- Збереження історії збірок і тестів.
- Гнучке управління користувачами з можливістю групування та різними методами автентифікації.
- Підтримка багатовузлової системи з додатковими серверами.

Ціна: комерційний інструмент із безкоштовною версією та ліцензіями (рис. 2.4)

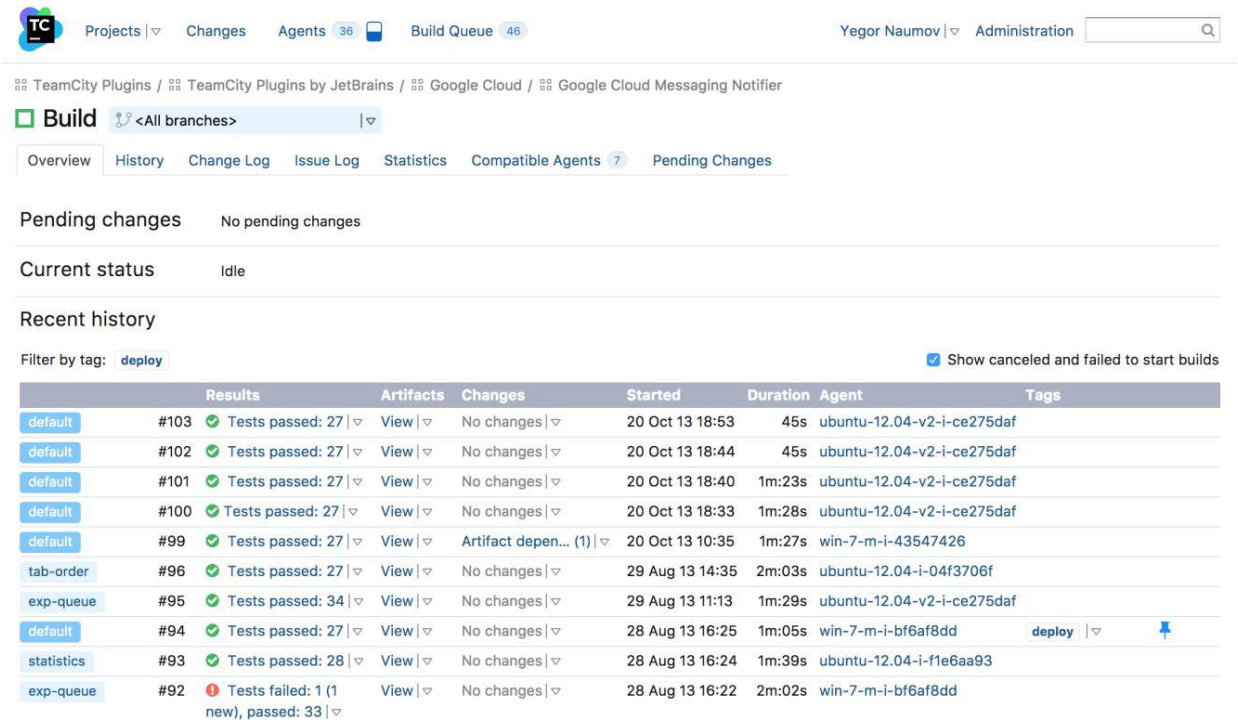


Рисунок 2.4 - Інтерфейс TeamCity

2.5 GitLab

GitLab – це комплексний набір інструментів для управління життєвим циклом розробки. Він дозволяє запускати збірки, тести та розгортання для кожного коміту або push.

Основні функції:

- Інструменти управління кодом і проєктами в єдиній системі.
- Підтримка повного циклу CI/CD, включно з тестуванням безпеки (SAST, DAST) та скануванням залежностей.
- Автоматизація випусків і розгортання.
- Сканування контейнерів для забезпечення безпеки.

Ціна: GitLab доступний безкоштовно або як комерційне рішення з хмарним чи локальним хостингом (рис. 2.5)

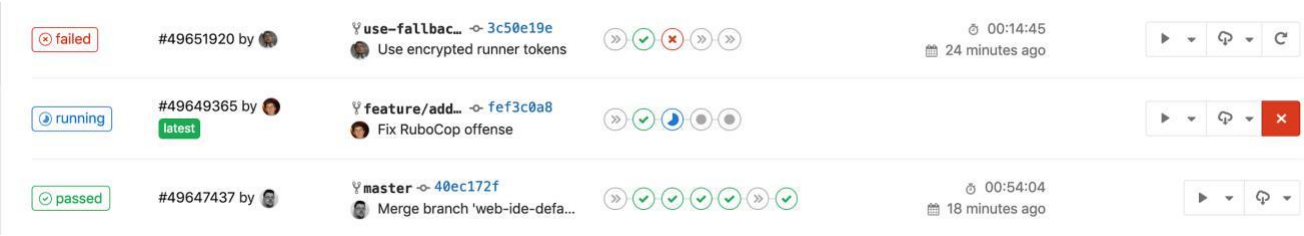


Рисунок 2.5 - Коміти та різні ступені тестів на GitLab CI

3 ГАЛУЗЕВІ ДОСЛІДЖЕННЯ ТА АНАЛІЗ ВПЛИВУ CI / CD ДЛЯ ВЕБ-ДОДАТКІВ

3.1 Співпраця та планування

Одним з ключових моментів підходу DevOps є співпраця всередині команди та постачальником. Сьогодні команда складається не тільки з розробників, це можуть бути і devs, і ops, тестувальники, дизайнери, менеджери релізів, менеджери продуктів та проектів тощо. Члени команди повинні тісно співпрацювати, розділяти спільні обов'язки та бути залучені до кожного етапу розробки

Мета інструментів управління вимогами полягає в тому, щоб забезпечити успішне досягнення цілей розробки продукту. забезпечити успішне досягнення цілей розробки продукту. Це набір методів для документування, аналізу, визначення пріоритетів та узгодження вимог, щоб інженерні команди завжди завжди мали актуальні та затверджені вимоги.

Всі учасники CI\CD повинні бути повідомлені про різні події CI/CD залежно від робочого процесу релізу в конкретній організації. Якщо в конвеєрі розгортання є якісь ручні кроки або затвердження в конвеєрі розгортання, команда повинна діяти швидко, щоб не затримати реліз. щоб не затримувати випуск. Вся комунікація повинна бути цільовою і суворо обмеженою щоб уникнути спаму та непорозумінь між членами команди.

Поширеними інструментами для використання є Trello, Jira, Clarizen, Bitbucket Server, Asana.

Ці програми є інструментами управління проектами, які підтримують стандартні функції для управління завданнями, відстеження часу, планування, виставлення рахунків, спілкування в чаті тощо.

Другий крок до ефективності процесу розробки - це IDE та редактори для проектної команди.

Звичайне середовище розробки включає в себе такі основні функції: Текстовий редактор, Компілятор або інтерпретатор, збирач та налагоджувач, підсвічування синтаксису, графічний інтерфейс користувача (GUI).

Згідно з даними GitHub, топ-3 IDE - це Visual Studio, Eclipse, Visual Studio Code (рис. 3.1).

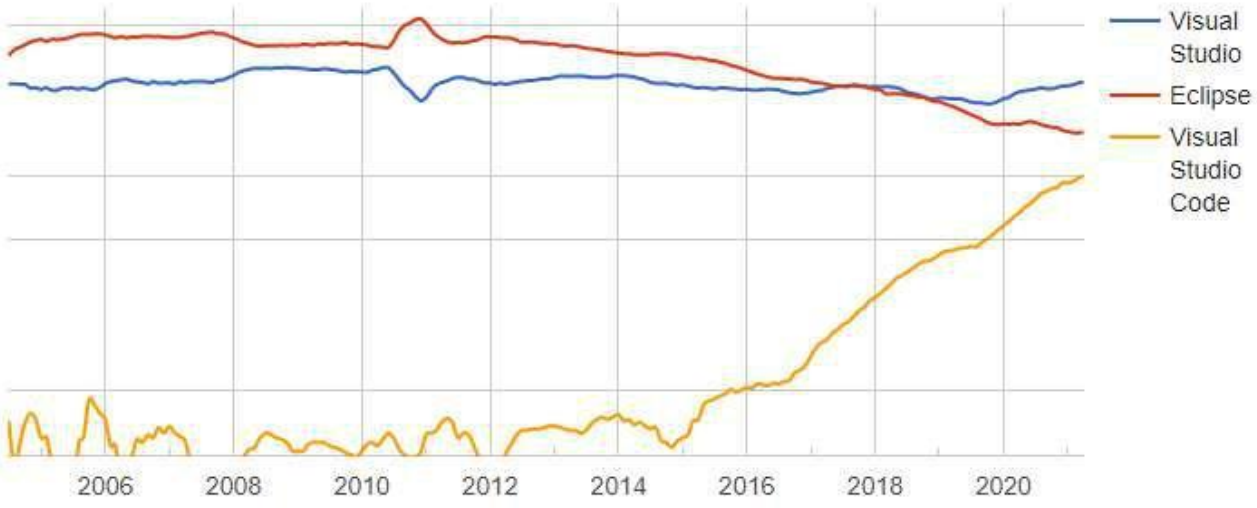


Рисунок 3.1 Світова тенденція для ТОП-3 IDE для веб-додатків

3.2 Конфігурація сервера та інфраструктура як код

3.2.1 Інфраструктура як код

Інфраструктура як код - це практика DevOps, така як CI\CD та хмарні технології і приносить більше користі від впровадження DevOps в організації Інфраструктура як код - це спосіб опису всієї програмної IT-інфраструктури, наприклад, віртуальних машин, правила безпеки, мережеві налаштування у вигляді простого коду, який можна зберігати та контролювати у вашій системі контролю версій (VCS) та оновлюватися на вимогу. Ці сховища записів називаються маніфестами. Вони використовуються інструментами DevOps, такими як Terraform, Kubernetes та інші пропрієтарні сервіси від хмарних провайдерів, такі як AWS CloudFormation, для автоматичного надання та

налаштування нових серверів та їхньої інфраструктуру для будь-якого типу середовища.

IaC забезпечує безперервність, оскільки всі середовища надаються та налаштовуються автоматично, без можливості людської помилки, що неймовірно прискорює і та спрощує просування продукту та завдання створення фундаменту. Підхід IaC забезпечує відстежуваність змін в інфраструктурі та легке відновлення інфраструктури розгортання інфраструктури. Це підвищує загальну стабільність та доступність IT-інфраструктури. Існує також багато переваг у проведенні операцій на основі принципу IaC. Відомими програмами, які дозволяють автоматизувати та впровадити інфраструктуру у вигляді коду в систему, є Puppet, Chef, Ansible, Terraform, SaltStack, AWS Cloudformation.

3.2.2 Хмарні обчислення

Хмарні обчислення - це доступність ресурсів комп'ютерної системи на вимогу, особливо сховища даних (хмарні сховища) та обчислювальних потужностей, без прямого активного управління з боку користувача. Цей термін зазвичай використовується для опису центрів обробки даних, доступних для багатьом користувачам через Інтернет.

Хмарні обчислення полегшують адаптацію DevOps, залучаючи кожну етап життєвого циклу розробки програмного забезпечення. Через хмару додатки можна створюватися і тестуватися в різних умовах, з різними середовищами, версіями ОС і емуляції пристроїв. Виключаючи фізичний сервер, вимога до тестування забезпечує економію часу економію часу і зниження витрат завдяки тому, що хмара працює на вимогу. Хмарні обчислення приносить нові можливості в процесах CI\CD та автоматизації. Хмара пропонує перенесення ризики та підвищити загальну надійність IT-інфраструктури завдяки зонам доступності та геолокаційної моделі, яку використовують більшість популярних провайдерів хмарних сервісів.

Найпоширенішими провайдерами хмарної інфраструктури є Amazon Web Services, Google Cloud Platform, Microsoft Azure, IBM Cloud та Oracle Cloud Platform.

П'ять основних характеристик хмарних обчислень включають:

● Самообслуговування та надання послуг на вимогу

● Широкий доступ до мережі

● Об'єднання ресурсів

● Швидка еластичність

● Вимірюваний сервіс

Три моделі (IaaS, PaaS, SaaS) вводять своєрідний рівень абстракції. Це рівень, про який ви не думаєте. Це рівень узагальнення. Якщо спростити наведену нижче картинку, можна сказати, що верхній рівень - це відповідальність клієнта, а нижній рівень це обов'язки постачальників хмарних послуг або IT-служби

3.3 Безперервна інтеграція та безперервна доставка

3.3.1 Базовий життєвий цикл розробки

Основні етапи життєвого циклу розробки (рис. 3.2), як правило, включають етап планування етап планування, за яким слідують кодування, збірка, інтеграція, тестування, випуск, а потім розгортання у розгортання у виробничому середовищі.

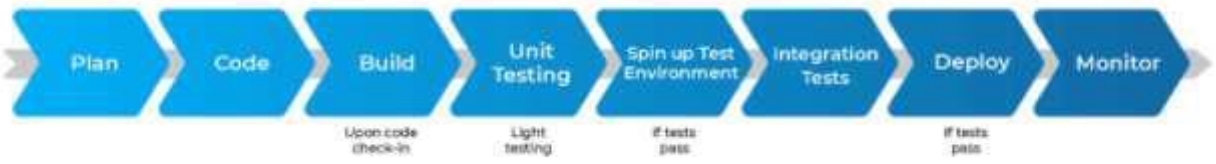


Рисунок 3.2 Життєвий цикл розробки

В рамках методології Agile, яка використовує ітеративний підхід до розробки програмного забезпечення, додаючи шар за шаром, будуючи, а точніше еволюціонуючи, додаток один спринт за одним час (рис. 3.3)

Рисунок 3.3 Життєвий цикл розробки vs середовище розробки

З точки зору DevOps, як правило, ця методологія може перетинатися з Agile (рис. 3.4)

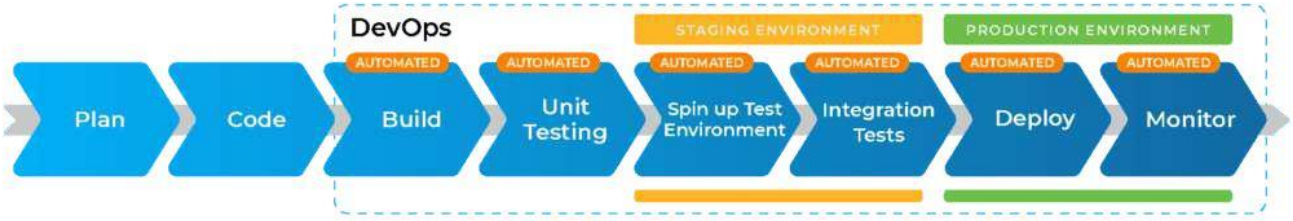


Рисунок 3.4 Життєвий цикл розробки в порівнянні з методологією DevOps

Але багато людей згадують DevOps та CI\CD разом. Дуже важливо розуміти, що ці дві концепції важливо розуміти, що ці дві концепції доповнюють одна одну, але не замінюють одне одного.

У той час як DevOps пропонує культуру/філософію командної співпраці та прозорості роботи, CI/CD - це стовпи, які уможливлюють DevOps. CI/CD фокусується на автоматизації побудови, тестування та розгортання шляхом належного налаштування конвеєра CI/CD за допомогою відповідними інструментами CI/CD.

3.3.2 Елементи конвеєра CI/CD для веб-додатків

Більшість релізів веб-додатків проходять через кілька типових етапів:

Етап вихідного коду: Загальним підходом є тригери, які налаштовуються залежно від потреб. У деяких випадках запуск конвеєра запускається сховищем вихідного коду або може бути автоматично запланованим або підтримуваним робочим процесом, ініційованим користувачем. Також він може бути

інтегрований з додатковими інструментами, такими як JIRA або Confluence, щоб організувати повний цикл для управління документами. На цьому етапі необхідно виконати кілька кроків, щоб покращити якість коду та швидкість роботи ринку. В основному це стосується операцій, що виконуються в Середовище місцевого розвитку. Кроки можуть бути наступними: компіляція, модульне тестування, статичний аналіз коду та ручна перевірка. Після того, як функція буде готова, або на щоденній основі зміни будуть перенесені у віддалену гілку функції і перейдуть на стадію збірки.

Docker контейнери. У віддалених гілках функцій буде виконуватися статичний аналіз коду, компіляція та модульне буде запущено статичний аналіз коду, компіляцію та модульне тестування для перевірки на стороні сервера. Якщо автоматичні тести пройдені, зміни будуть перенесені в середовище розробки, де будуть виконані наступні кроки - компіляція, автоматична інтеграція API, тестування, тестування нових функцій та розгортання. Якщо код схвалений власником продукту або відповідальною особою, створюються запити на злиття і відправляються на перевірку коду.

Етап тестування: Тут запускаються автоматизовані тести для перевірки коректності коду та поведінки веб-додатку. Основна мета етапу - запобігти появі легко відтворюваних помилок. легко відтворювані помилки від потрапляння до кінцевих користувачів. Після перегляду коду код потрапляє до релізну гілку для інтеграційного та модульного тестування. Беручи до уваги, що тестування є однією з важливих частин розробки та підвищення якості, тестування в середовищі UAT є дуже важливим. В середовищі UAT куріння і системне тестування виконуються і повинні бути схвалені бізнес-користувачами.

Етапи розгортання: Зазвичай існує декілька середовищ розгортання: Розробка, QA, Staging та Production. Команди, які прийняли гнучку модель розробки: керуючись тестами та моніторингом у реальному часі, зазвичай розгортають незавершене виробництво вручну в середовище для додаткового ручного тестування та перегляду, а також автоматично розгортають затверджені

зміни. перегляду, і автоматично розгортають затверджені зміни з основної гілки у виробництво.

Тестування продуктивності - це процес, під час якого оцінюється якість продукту або його здатність продукту або його здатність функціонувати в необхідному середовищі. Як нефункціональний метод тестування тестування продуктивності проводиться для оцінки здатності системи з точки зору швидкості реагування та стабільності роботи під навантаженням. Цей процес проводиться за трьома основними атрибутами, які включають масштабованість, надійність і використання ресурсів.

- Для перевірки продуктивності програмного або апаратного забезпечення в середовищі тестування продуктивності. До них відносяться:
- Навантажувальне тестування є найпростішою формою тестування і в основному проводиться для того, щоб зрозуміти поведінку системи під певним навантаженням.
- Стрес-тестування проводиться для перевірки здатності системи справлятися зі збільшеним навантаженням, якщо таке є. Воно проводиться для визначення максимальної продуктивності існуючої система’
- Занурювальне тестування, також відоме як тестування на витривалість, - це тип тестування, що проводиться для перевірки здатності системи працювати в умовах безперервного навантаження.
- Навантажувальне тестування проводиться шляхом різкого збільшення кількості користувачів системи і визначається, як система працює під таким навантаженням.

Існує кілька простих способів забезпечити точність і кращі точність і кращі результати тестів.

3.3.3 CI/CD як єдиний конвеєр для веб-додатків

Безперервна інтеграція та доставка/розгортання (CI/CD) виконується за допомогою єдиний конвеєр з високим рівнем автоматизації на кожному етапі процесу.

Для описаної вище мети можна використовувати наступні додатки: Jenkins, Gradle, GitLab CI, Travis CI, Bamboo CI, Teamcity, робочі процеси GitHub, Circle CI та Azure.

З точки зору категорій, додатки поділяються на кілька категорій. Програмне забезпечення, що встановлюється: Програми, які можна встановити та налаштувати для початкової мети. Вони можуть бути з відкритим вихідним кодом або пропрієтарними SaaS: Сервіси з веб-інтерфейсом, що надаються третьою стороною (наприклад, CircleCI, Azure DevOps) і можуть бути розміщені на власному хостингу або в хмарі.

Інтегроване програмне забезпечення: Більшість репозиторіїв (наприклад, GitLab, GitHub, BitBucket) підтримують власні веб-сервіси CI/CD, які інтегровані в їхні системи контролю вихідного коду.

Вибір категорії залежить від конкретної програми та вимог, що розглядаються.

Інструменти можна оцінювати за багатьма критеріями, такими як хостинг, безкоштовна версія, ціна та можливості інтеграції (рис. 3.5)

	Jenkins	circleci	TeamCity	Bamboo	GitLab
Open source	Yes	No	No	No	No
Ease of use & setup	Medium	Medium	Medium	Medium	Medium
Built-in features	3/5	4/5	4/5	4/5	4/5
Integration	★ ★ ★ ★ ★	★ ★ ★	★ ★ ★ ★	★ ★ ★	★ ★ ★ ★
Hosting	On premise & Cloud	On premise & Cloud	On premise	On premise & Bitbucket as Cloud	On premise & Cloud
Free version	Yes	Yes	Yes	Yes	Yes
Build Agent License Pricing	Free	From $39 per month	From $299 one-off payment	From $10 one-off payment	From $4 per month per user
Supported OSs	Windows, Linux, macOS, Unix-like OS	Linux or MacOS	Windows, Linux, macOS, Solaris, FreeBSD and more	Windows, Linux, macOS, Solaris	Linux distributions: Ubuntu, Debian, CentOS, Oracle Linux

Рисунок 3.5 Топ-5 інструментів CI/CD у 2020 році

3.4 Контейнеризація та оркестрування

Контейнери популярні серед DevOps, оскільки вони пропонують логічну упаковку механізм, за допомогою якого додатки можна абстрагуватися від середовища, в якому вони фактично працюють. Оркестратори - це інструменти, які використовуються для моніторингу та налаштування всіх існуючих контейнерів. Найчастіше вони вбудовуються в конвеєри CI/CD як інструменти за замовчуванням або можуть бути підключаються як розширення. У веб-додатках десятки контейнерів будуть пов'язані між собою, утворюючи додаток.

Широко відомими постачальниками контейнерів є Docker, Kubernetes, OpenShift.

Docker - це набір продуктів платформи як послуги (PaaS), які використовують віртуалізацію на рівні операційної системи віртуалізацію на рівні ОС для доставки програмного забезпечення в пакетах, які називаються контейнерами. Контейнери ізольовані один від одного і містять власне програмне забезпечення, бібліотеки та конфігураційні файли; вони можуть спілкуватися один з одним через чітко визначені канали. контейнери користуються послугами одного ядра операційної системи, вони використовують менше ресурсів, ніж віртуальні машини. Сервіс має безкоштовний та преміум-рівні. Програмне забезпечення програмне забезпечення, на якому розміщуються контейнери, називається Docker Engine. Його було вперше запущено у 2013 році і розробляється компанією Docker, Inc.

Kubernetes - це портативна, розширювана платформа з відкритим вихідним кодом для управління контейнерними робочими навантаженнями та сервісами, що полегшує як декларативну конфігурацію, так і автоматизація. Він має велику екосистему, що швидко зростає. Сервіси, підтримка та інструменти Kubernetes інструменти Kubernetes широко доступні. Назва Kubernetes походить з грецької мови, що означає керманич або пілот. K8s як абревіатура є результатом підрахунку восьми літер між «K» і «s». Google запустив проект Kubernetes з відкритим вихідним кодом у 2014 році.

Kubernetes поєднує в собі більш ніж 15-річний досвід Google в управлінні виробничими з найкращими у своєму роді ідеями та практиками від спільноти. Сервіс OpenShift побудований на основі контейнерів Docker, які організовуються та управляються Kubernetes на основі Red Hat Enterprise Linux. Інші продукти сімейства надають цю платформу через різні середовища: OKD слугує як висхідним потоком, керованим спільнотою (подібно до того, як Fedora є висхідним потоком для Red Hat Enterprise Linux), OpenShift Online - це платформа, що пропонується як програмне забезпечення як послуга, а OpenShift Dedicated - це платформа, що пропонується як керована послуга. Консоль OpenShift Console має режими, орієнтовані на розробника та адміністратора. Погляди адміністратора дозволяють відстежувати ресурси контейнера та стан контейнера, керувати користувачами, працювати з операторами тощо, тощо. Подання розробника орієнтовані на роботу з ресурсами програми в межах простору імен простору імен. OpenShift також надає CLI, який підтримує надмножину дій, які Kubernetes CLI.

3.5 Моніторинг та оповіщення

Якщо один з етапів CI\CD не пройшов або був знижений, всі залучені учасники повинні знати про цю проблему. Для цього існує цілий ряд інструментів та для моніторингу додатків через спеціальний інтерфейс.

Моніторинг продуктивності все більше розвивається як необхідна частина бізнес-процесів. бізнес-процесів. Доступність і надійність системи мають прямий вплив на кінцевий результат компанії. Бізнес хоче знати, як обробляються транзакції та що відбувається в певних компонентах програми.

Складні системи і додатки стикаються з безліччю питань і технічних проблем під час роботи у виробничих умовах. Час виявлення проблеми може варіюватися від 30 секунд до 30 годин і навіть більше. Але є способи скоротити цей час і покращити вирішення проблем командою IT-системи. Моніторинг застосовується для обробки та вирішення такого роду проблем.

Моніторинг - це систематичний процес збору, аналізу та використання інформації для відстеження прогресу програми на шляху до досягнення її цілей та прийняття управлінських рішень.

Компанії повинні використовувати методи моніторингу у своїх мережах, фізичних і віртуальних серверах та сервісах. та віртуальних серверах і сервісах. За допомогою моніторингу легше отримати більшу видимість, а сааме командам системної інженерії легше визначити, оцінити та виправити проблему в додатку.

Моніторинг також широко використовується в управлінні безпекою і продуктивністю додатків.

Компанії отримують багато користі від моніторингу, особливо для того, щоб

- Зменшити витрати на виявлення проблем.
- Зменшити час, необхідний для вирішення проблеми.
- Мінімізувати ризик витоку/витоку даних.

Скоротити час на вирішення проблеми.

Збільшити дохід, оскільки додаток буде доступний більше часу, а зменшиться час простою.

Чотири стовпи (вони ж джерела) моніторингу додатків:

- Бізнес KPI, угода про рівень обслуговування, дані веб-середовища
- Інфраструктура, технічне обладнання
- Бази даних, транзакції, запити користувачів
- Мережеві дані, поведінка користувачів

Процес моніторингу складається з кількох етапів. Для того, щоб отримати від нього користь,кожен з цих кроків має бути реалізований у наступному порядку:

- Зібрати інформацію від виробників даних.
- Засвоїти її.
- Перетворення даних.
- Шукати та аналізувати.

- Візуалізація даних.

Дані для моніторингу можна збирати з різних джерел. Інструменти моніторингу дозволяють приладам конфігурувати багато видів вхідних даних, в тому числі специфічних для специфічні для конкретних потреб програми. Деякі продукти надають користувачеві можливість конфігурувати будь-які довільні типи вхідних даних.

Найпоширенішими джерелами даних для моніторингу є

- Каталоги та файли
- Мережеві події.
- Джерела Windows.
- Метадані.

Моніторинг вимагає збору та забезпечення спостережливості системи. Для цього використовуються різні метрики. Моніторинг використовують розробники, IT-спеціалісти команда IT-операцій, команда безпеки, технічна підтримка, бізнес-лідери. Моніторинг може бути доповнений різними інструментами та методами. Одним з найпопулярніших зараз є AIOps. Він використовує штучний штучний інтелект для спрощення управління IT-операціями, прискорення та автоматизації вирішення проблем у складних сучасних IT-середовищах. вирішення проблем у складних сучасних IT-середовищах Prometheus - це інструментарій для моніторингу та оповіщення систем з відкритим вихідним кодом, спочатку створений на SoundCloud. З моменту заснування у 2012 році багато компаній та організацій взяли на озброєння Prometheus, і проект має дуже активну спільноту розробників і користувачів. спільнота розробників і користувачів. Зараз це окремий проект з відкритим вихідним кодом, який підтримується незалежно від від будь-якої компанії. Щоб підкреслити це, а також прояснити структуру управління проектом,

Prometheus приєднався до Cloud Native Computing Foundation у 2016 році як другий після Kubernetes.

Prometheus підтримує збір чотирьох основних типів метрик. Архітектура Prometheus має модульну архітектуру і має деякі вже доступні модулі, що

називаються експортерами, які отримують метрики з програмного забезпечення. Архітектура Prometheus є модульною і має деякі вже доступні модулі, які називаються експортерами, що збирають метрики з програмного забезпечення.

Як канал зв'язку можна використовувати наступні інструменти:

Slack - це платформа для обміну повідомленнями на основі каналів, яка підтримує ботів. Бот – це зручний спосіб виконання коду та автоматизації завдань. Бот може бути використаний для розміщення повідомлень у каналів та реагування на активність учасників. З точки зору CI/CD слак може надсилати повідомлення аудиторії, яка підписана на бота, та повідомляти про процеси CI/CD та оприлюднювати результати.

Telegram - це безкоштовна, крос-платформна, хмарна програма для обміну миттєвими повідомленнями (IM) програмне забезпечення та сервіс додатків. CI/CD може надсилати повідомлення безпосередньо людині або використовувати бота, який сповіщає про процеси або помилки, що відбуваються.

Електронна пошта є стандартним і поширеним способом комунікації, до якого звикли у цифровому світі звикає до цифрового світу. Електронні листи потенційно можуть спричинити інформаційне перевантаження. Деякі повідомлення можуть бути відхилені або залишені непрочитаними, особливо якщо їх надходить дуже багато.

Для моніторингу працездатності системи можна використовувати наступну програму з спеціальним інтерфейсом можна використовувати наступні додатки зі спеціальним інтерфейсом: Prometheus, Sensu GO, Nagios, Elastic Stack

3.6 Безпека та DevSecOps

3.6.1 Життєвий цикл розробки безпеки

З розвитком хмарних обчислень та автоматизованих конвеєрів з'являється все більше вразливостей у безпеці з'являється все більше вразливостей - те, над чим працюють розробники, є недоліками та вразливостями самого додатку.

Напрямок SDL або SDLC – Security життєвий цикл розробки - був розроблений компанією Microsoft (рис. 3.6)

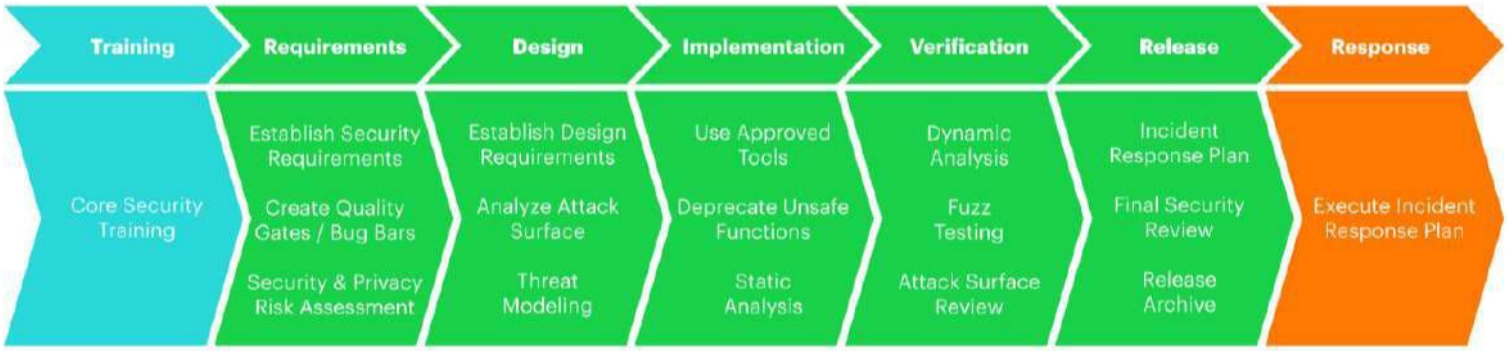


Рисунок 3.6 Безпека додатків

Безпека додатків та SDLC - це не про виявлення вразливостей, а про запобігання їх виникненню. SDLC детально описується в різних методологіях - OpenSAMM, BSIMM, OWASP

3.6.2 Побудова моделі безпеки на стадії зрілості (BSIMM)

Методологія базується на поділі процесу забезпечення безпеки додатків на 4 області: Управління, Розвідка, Точки дотику SSDL та Розгортання. Кожна область має 12 практик з 112 видами діяльності. Ця методологія може бути використана як основа для впровадження безпеки

DevSecOps - це підхід до виявлення безпеки інфраструктури та CI\CD конвеєра CI\CD. Для перевірки зазвичай використовуються Codacy, SonarQube та Logz.io.

3.6.3 Ризики безпеки веб-додатків (OWASP)

OWASP Top 10 - це стандартний інформаційний документ для розробників та безпеки веб-додатків. Він представляє широкий консенсус щодо найбільш критичних ризиків для безпеки веб-додатків. Стандарт включає в себе наступні ризики для безпеки веб-додатків:

1. Ін'єкції. Ін'єкції, такі як SQL, NoSQL, OS і LDAP ін'єкції, виникають коли ненадійні дані надсилаються інтерпретатору як частина команди або запиту.

2. Порушена автентифікація. Функції програми, пов'язані з аутентифікацією та управлінням сеансами, часто реалізовані некоректно, що дозволяє зловмисникам скомпрометувати скомпрометувати паролі, ключі або токени сеансів, або використати інші недоліки реалізації, щоб тимчасово або назавжди.

3. Вразливість чутливих даних. Багато веб-додатків та API не захищають належним чином захищають конфіденційні дані, такі як фінансові, медичні та персональні дані.

4. Зовнішні об'єкти XML (XXE). Багато старих або погано налаштованих XML-процесорів оцінюють посилання на зовнішні сутності в документах XML. Зовнішні сутності можуть використовуватися для розкриття внутрішніх файлів за допомогою обробника URI файлу, внутрішніх файлових ресурсів, сканування внутрішніх портів, віддаленого виконання коду та атак на відмову в обслуговуванні.

5. Порушення контролю доступу. Обмеження на те, що дозволено робити аутентифікованим користувачам часто не дотримуються належним чином.

6. Неправильна конфігурація безпеки. Неправильна конфігурація безпеки є найбільш поширеною проблема. Зазвичай це результат небезпечних конфігурацій за замовчуванням, неповних або спеціальних конфігурацій, відкритого хмарного сховища спеціальних конфігурацій, відкритих хмарних сховищ, неправильно налаштованих заголовків HTTP та багатослівних повідомлень про помилки, що містять конфіденційну інформацію.

7. Міжсайтовий скриптинг (XSS). Помилки XSS виникають щоразу, коли додаток включає ненадійні дані в нову веб-сторінку без належної перевірки або екранування, або оновлює існуючу веб-сторінку даними, наданими користувачем, використовуючи API браузера, який може створювати HTML або JavaScript.

8. Небезпечна десеріалізація. Небезпечна десеріалізація часто призводить до віддаленого виконання коду виконання коду.

9. Використання компонентів з відомими уразливостями. Компоненти, такі як бібліотеки, фреймворки та інші програмні модулі, запускаються з тими ж привілеями, що і додаток.

10. Недостатнє ведення журналів та моніторинг. Недостатнє ведення журналів і моніторингу в поєднанні з відсутністю або неефективною інтеграцією з системами реагування на інциденти з відсутністю або неефективною інтеграцією з реагуванням на інциденти дозволяє зловмисникам продовжувати атакувати системи, зберігати стійкість, переходити на інші системи та втручатися, вилучати або знищувати дані.

4 ЗБІРКА СЕРВЕРА АВТОМАТИЗАЦІЇ JENKINS

4.1 Етап збірки

Це відбувається, коли компілюється вихідний код і пов’язані з ним залежності побудувати виконуваний екземпляр прикладного продукту. Для побудови цієї сцени Дженкінгс можна використовувати pipeline.

Перш ніж використовувати Jenkins, програму потрібно налаштувати та налаштувати Terraform — це інфраструктура з відкритим вихідним кодом як програмний інструмент, створений HashiCorp.

Користувачі визначають і надають інфраструктуру центру обробки даних за допомогою декларативної конфігурації мову, відому як HashiCorp Configuration Language (HCL), або, за бажанням, JSON.

Terraform керує зовнішніми ресурсами (такими як публічна хмарна інфраструктура, приватна хмара інфраструктура, мережеві пристрої, програмне забезпечення як послуга та платформа як послуга).

"провайдери".

Для розгортання Jenkins на екземплярах AWS потрібно виконати кілька кроків. Провайдер конфігурації належать до кореневого модуля конфігурації Terraform.

Конфігурація провайдера створюється за допомогою блоку провайдера:

```
provider "aws" {
 profile    = "default"
 region     = "us-east-2"
access_key = "AKIXXXXWEZ37VXXC4R4R"
secret_key = "isaTYvCaLialt+k1I42XXuk37hgXsq97IYXXXIEA" }
```

Ім’я – це локальне ім’я постачальника, який потрібно налаштувати. Цей провайдер вже включено до блоку required_providers.

Тіло блоку (between { and }) містить аргументи конфігурації для провайдер.

Наступною частиною конфігурації є розділ ресурсів. Надає екземпляр EC2 ресурс. Це дозволяє створювати, оновлювати та видаляти екземпляри. Примірники також надання підтримки.

```
resource "aws_instance" "Ubuntu" {
  count                  = 1
  ami                    = "ami-01e7ca2ef94a0ae86"
  instance_type          = "t2.micro"
  vpc_security_group_ids = ["sg-05118f212a2ddd293"]
associate_public_ip_address = "true"
  key_name               = "newkey"
  provisioner "remote-exec" {
    inline = [
"wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key
add -",
"sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ >
/etc/apt/sources.list.d/jenkins.list'",
      "sudo apt update -qq",
      "sudo apt install -y default-jre",
      "sudo apt install -y jenkins",
      "sudo systemctl start jenkins",
        "sudo iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT --
to-port 8080",
      "sudo sh -c \"iptables-save > /etc/iptables.rules\"",
       "echo iptables-persistent iptables-persistent/autosave_v4 boolean true | sudo
debconf-set-selections",
```

```
"echo iptables-persistent iptables-persistent/autosave_v6 boolean true | sudo debconf-set-selections",
"sudo apt-get -y install iptables-persistent",
"sudo ufw allow 8080",
]
}
```

Провайдеру потрібен доступ до віддаленого ресурсу через SSH і очікується а вкладений блок підключення з докладною інформацією про підключення.

```
{ type = "ssh"
host = self.public_ip
user = "ubuntu"
private_key = file("newkey.pem")

}
```

Коли є багато ресурсів (наприклад, екземпляри, VCN, балансувальники навантаження, і блокові об’єми) у кількох відсіках під час оренди, може стати важко відстежувати ресурси, що використовуються для певних цілей, або для їх агрегування, звітування про них або взяття масові дії на них. Додавання тегів дозволяє визначати ключі та значення та пов’язувати їх із ними ресурси. Теги можна використовувати, щоб допомогти впорядкувати та створити список ресурсів на основі бізнесу потреби.

Існує два типи тегів:

Перший визначений теги, налаштовані в оренді адміністратором. Інший є що теги вільної форми можуть бути застосовані будь-яким користувачем з дозволом на ресурс.

```
tags = {
```

"Name" = "Jenkins_Server"

"Terraform" = "true"

}

Після завершення інсталяції Jenkins доступний за адресою локального хоста порт за замовчуванням 8080, наприклад http://jenkings_diplom:8080. У першому запуску Unlock Jenkins з'явиться вікно, у якому буде показано розташування початкового пароля. Щоб переглянути ключ команда cat, яка використовується для відображення пароля: $$ sudo cat початковий шлях до пароля адміністратора. В водночас журнал консолі Jenkins вказує розташування (у домашньому каталозі Jenkins) де також можна отримати цей пароль.

Pipeline — це набір плагінів, які підтримують впровадження та інтеграція безперервної доставки. У найпростіших і поширених використання, тепер ви можете змусити одне завдання виконуватися лише за наявності кількох інших, паралельних завдань завершено успішно. По суті, крок говорить Дженкінсу, що робити в конкретному випадку момент часу для мого випадку, щоб виконати команду оболонки та використати збірку номер. Коли плагін розширює Pipeline DSL, це зазвичай означає, що плагін має здійснив новий крок.

Jenkins пропонує встановити деякі плагіни. Є папки плагіни, плагіни інструментів збирання, такі як Ant і Gradle. Плагіни для роботи з Git і GitHub і деякі плагіни для конвеєра. Є плагіни конвеєра, перегляд етапів конвеєра, конвеєр GitHub Бібліотеки Groove і плагіни, що підтримують SSH.

Усі пристрої запущено, і з’явиться форма початку роботи для створення а досвідчений користувач. У формі необхідно вказати ім'я, пароль та інші дані. Після Jenkings потрібно надати URL-адресу, і після останньої кнопки «Почати використання Jenkings» можна торкнутися як на рис. 4.1

Рисунок 4.1 Головна сторінка Jenkins

Щоб налаштувати конвеєр, необхідно виконати кілька кроків. Конфігурація складається з налаштування облікових даних і конфігурації плагіна, включаючи базові властивості.

Плагіни є основним засобом покращення функціональності Jenkins середовище відповідно до потреб організації чи користувача. Їх понад тисячу різні плагіни, які можна встановити на майстер Jenkins і інтегрувати різні збірки інструменти, хмарні постачальники, інструменти аналізу та багато іншого. Плагіни можуть бути автоматично завантажені разом із залежностями з Центру оновлення. Центр оновлення — це а послуга, керована проектом Jenkins, яка надає інвентаризацію з відкритим кодом плагіни, розроблені та підтримувані різними членами спільноти Jenkins. The плагіни упаковані як самодостатні файли .hpi, які містять увесь необхідний код, зображення та інші ресурси, необхідні для успішної роботи плагіна.

Після встановлення через веб або командний рядок кожен плагін потрібно налаштувати окремо, в залежності від того, що і як цей плагін буде використовуватися для конвеєра.

Наступним кроком є створення вільних робочих місць для автоматизації конвеєрів з нуля. По-перше контрольну версію потрібно додати як джерело коду https://github.com/. Потрібні тригери створення другого кроку конфігурується залежно від необхідності та технічних вимог як на рис. 4.2

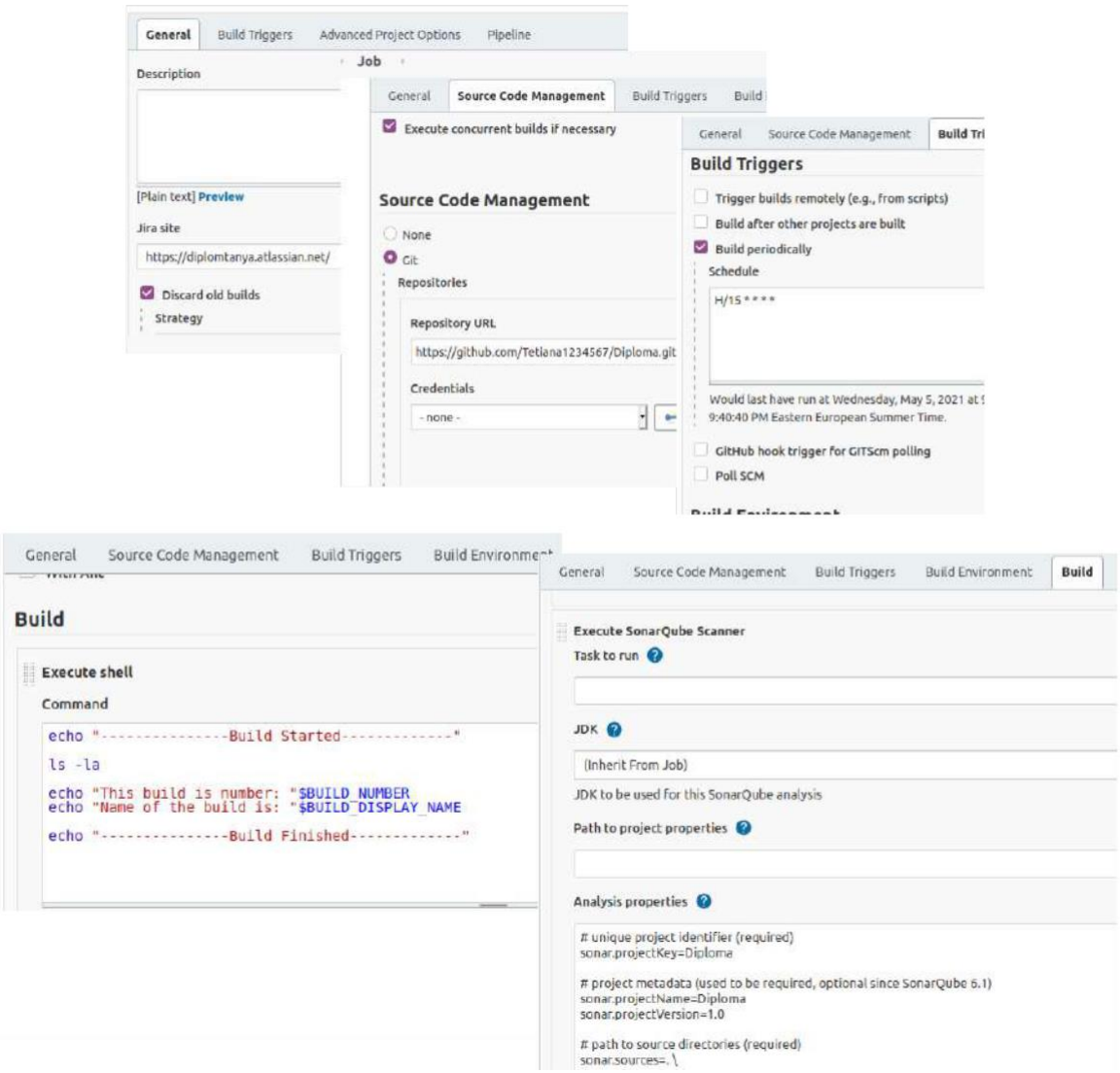


Рисунок 4.2 Головна сторінка Jenkins

Jobs logs:

Started by timer

Running as SYSTEM

Building in workspace
/var/lib/jenkins/workspace/Job The recommended git
tool is: NONE No credentials specified

> git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/Job/.git # timeout=10

Fetching changes from the remote Git repository

> git config remote.origin.url https://github.com/Tetiana1234567/Diploma.git
timeout=10

Fetching upstream changes from https://github.com/Tetiana1234567/Diploma.git

> git --version # timeout=10

> git --version # 'git version 2.25.1'

> git fetch --tags --force --progress -- https://github.com/Tetiana1234567/Diploma.git
+refs/heads/*:refs/remotes/origin/* # timeout=10
> git rev-parse refs/remotes/origin/main^{commit} # timeout=10

Checking out Revision eea08f02f2b0b3423d2180e5492af97b57a895e0
(refs/remotes/origin/main)

> git config core.sparsecheckout # timeout=10

> git checkout -f eea08f02f2b0b3423d2180e5492af97b57a895e0 #
timeout=10 Commit message: "Update README.md"
> git rev-list --no-walk 62a886567f6f3bea58fd835fc487e90667223cd7 #
timeout=10 [Job] $ /bin/sh -xe /tmp/jenkins14857247871819482941.sh
+ echo ----------Build Started----------

----------Build Started----------

+ ls -la
total 84
drwxr-xr-x 7 jenkins jenkins 4096 тра 27 00:25 . drwxr-xr-
x 4 jenkins jenkins 4096 тра 5 20:35 .. -rw-r--r-- 1 jenkins
jenkins 8244 тра 27 00:25 about.html -rw-r--r-- 1 jenkins
jenkins 9936 кві 29 22:29 contact.html drwxr-xr-x 2
jenkins jenkins 4096 тра 27 00:25 css -rw-r--r-- 1 jenkins
jenkins 11264 кві 29 22:29 gallery.html drwxr-xr-x 8
jenkins jenkins 4096 тра 27 00:25 .git drwxr-xr-x 2
jenkins jenkins 4096 кві 29 22:29 images -rw-r--r-- 1
jenkins jenkins 14575 кві 29 22:29 index.html drwxr-xr-x
2 jenkins jenkins 4096 кві 29 22:29 js

-rw-r--r-- 1 jenkins jenkins 440 тра 27 00:25 README.md

drwxr-xr-x 3 jenkins jenkins 4096 тра 7 01:55 .scannerwork

Git Build Data

Revision: eea08f02f2b0b3423d2180e5492af97b57a895e0

refs/remotes/origin/main

Changes:

Summary

Add Codacy badge (details)

Commit (details)

Commit (details)

Update README.md (details)

Commit 4968e9c044789734d34aa288b9c1dd623a4f0d6e by badger

Add Codacy badge

The file was modified README.md (diff)

Commit a41718ea4dc76352d04f88eadc162da64ab92d11 by skapustasertey

Commit

The file was modified about.html (diff)

Commit 2c94face3a04f6296dab6d030c0ec9c6508598a2 by tanyasoft

Commit

The file was modified css/nice-select.css (diff)

The file was modified about.html (diff)

Commit eea08f02f2b0b3423d2180e5492af97b57a895e0 by noreply

Update README.md

The file was modified README.md (diff)

Build result:

Build #175 (27 May 2021, 00:25:00)

add description

Changes

Add Codacy badge (details / githubweb)

Commit (details / githubweb)

Commit (details / githubweb)

Update README.md (details / githubweb)

Started by timer

Revision: eea08f02f2b0b3423d2180e5492af97b57a895e0

refs/remotes/origin/main

Built Branches

refs/remotes/origin/main: Build #175 of Revision eea08f02f2b0b3423d2180e5492af97b57a895e0 (refs/remotes/origin/main)

Складні системи та програми стикаються з багатьма проблемами та технічними проблемами під час роботи у виробничих умовах. Час для виявлення проблеми може бути різним від 30 секунд до 30 годин або навіть більше. Але є способи зменшити цей час покращити вирішення проблем командою IT-системи.

4.2 Етап тестування

Це один з найважливіших аспектів конвеєра CI/CD; використання автоматизованих тестів для перевірки правильності та життєздатності коду. Для цього існує багато інструментів, які можна використовувати. Для перегляду коду, аналізу якості коду та аналізу коду безпеки можна використовувати кодування та покриття.

Для інтеграції цих інструментів можна запропонувати локальну та локальну інсталяцію. Для перевірки коду можна скористатися сайтом codacy.com, який допоможе проаналізувати дані проекту. На першому місці стоїть реєстрація, базові властивості проєкту потрібно налаштувати і налаштувати основні властивості проекту. Мій Github-репозиторій було додано до codacy - https://github.com/Tetiana1234567/Diploma. Початкове налаштування складається з декількох кроків, включаючи клонування репозиторію, визначення мов та інших властивостей, запуск шаблонів коду і останній крок - завершення аналізу за допомогою детальних репостів.

Codacy запускає аналіз для кожного коміту сховища. Це дозволяє нам відстежувати еволюцію та прогрес у створенні сховища. Для застосування двох способів інтеграції веб-хуки повинні бути налаштовані з певним приватним ключем, https://app.codacy.com/events/github/3391b874f54043429740a0eadbe840db (рис. 4.3)

Рисунок 4.3 Налаштування веб-хуків

Після рецензування бейдж може бути присвоєний проекту і може автоматично оновлюватися на основі результатів рецензування, як показано на рис. 4.4.

На електронну пошту буде надіслано відповідне повідомлення:

Ви можете переглянути, прокоментувати або об'єднати цей запит онлайн за адресою:

https://github.com/Tetiana1234567/Diploma/pull/2

Commit Summary

- Add Codacy badge

File Changes

- **M** README.md

(1) Patch Links:

- https://github.com/Tetiana1234567/Diploma/pull/2.patch
- https://github.com/Tetiana1234567/Diploma/pull/2.diff

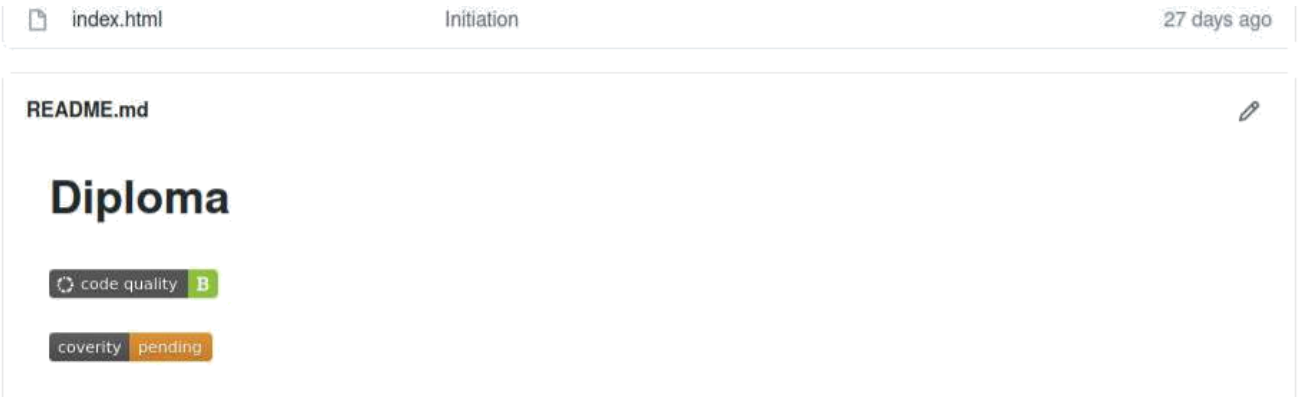


Рисунок 4.4 Позначка якості коду

Для сповіщень можна використовувати і налаштовувати слак (рис. 4.5). Вхідні веб-хуки можна додати до базової конфігурації за допомогою ключа Webhook https://hooks.slack.com/services/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

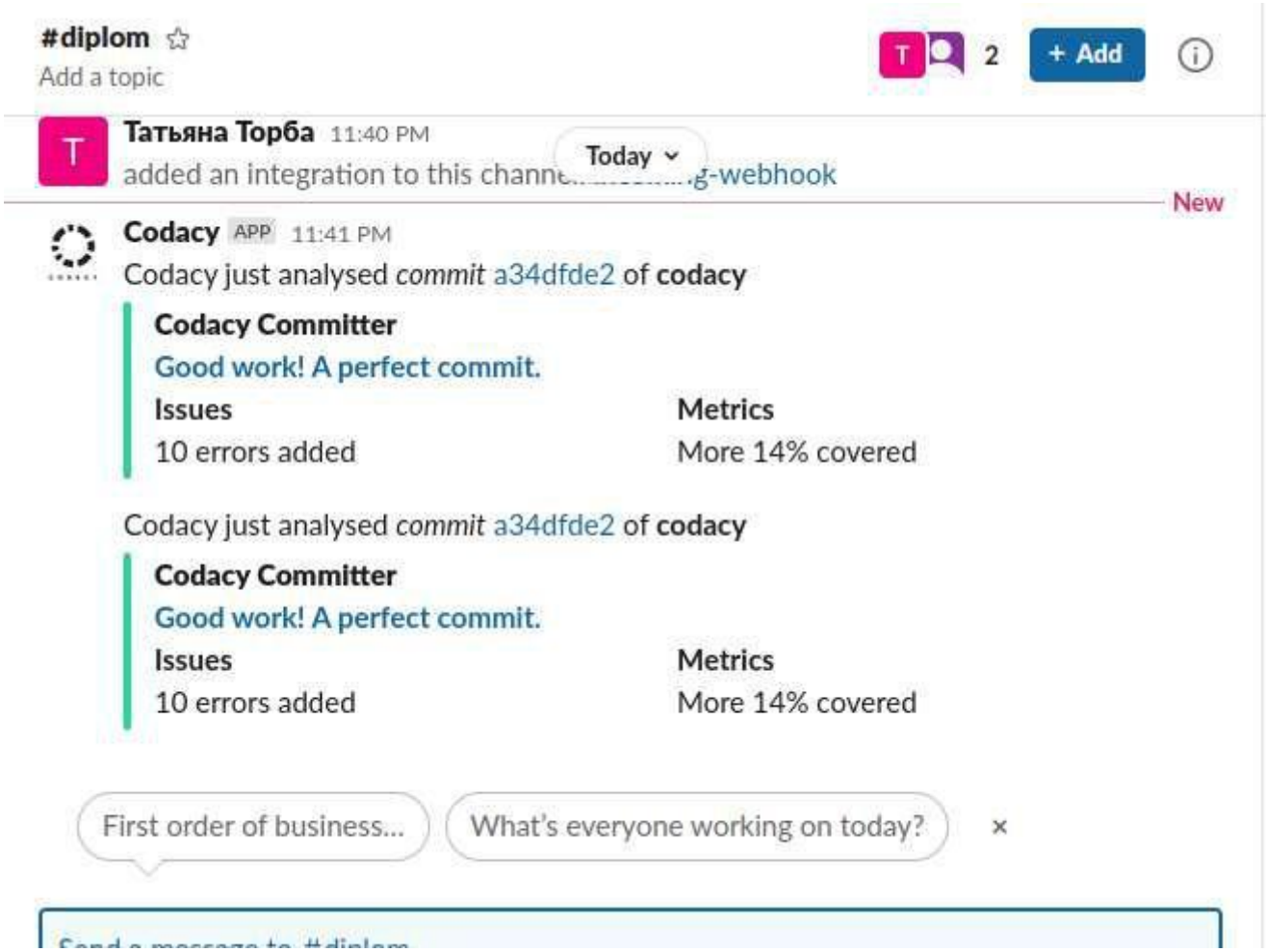


Рисунок 4.5 Slack сповіщення

4.3 Етап розгортання

Після того, як створений запущений екземпляр пройде всі тести, наступний етап у конвеєр розгортання. Етап розгортання є основним етапом конвеєра CI\CD, оскільки це останній етап забезпечує основну бізнес-цінність для бізнесу та клієнтів. Для Інтернету хостинг Я використовував AWS S3. Щоб створити етап S3, потрібно виконати кілька кроків заздалегідь:

- Створіть відро S3 у консолі AWS
- Налаштуйте Jenkins для розгортання артефактів у статичному локальному сховищі
- Завантажте ресурси у відро та налаштуйте його для статичного хостингу.

Щоб створити статичний веб-хостинг, увійдіть до консолі AWS як користувач root. Після в головному меню необхідно вибрати пункти S3. Коли Amazon S3 успішно створює вибраного відра, консоль відобразить порожнє відро на панелі Bucket (рис. 4.6)

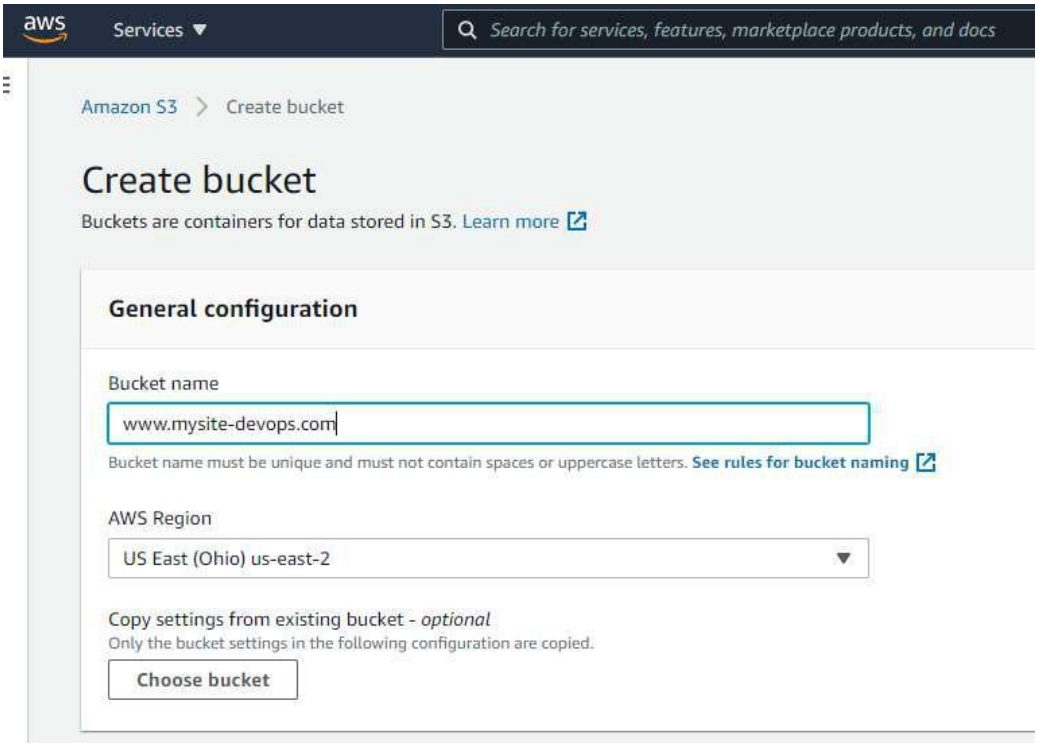


Рисунок 4.6 Конфігурація AWS S3

Основний статичний хостинг веб-сайтів починається з конфігураційної частини. Кілька параметри потрібно налаштувати під час налаштування сегмента,

деякі з них можна змінити під час експлуатації. Щоб увімкнути веб-хостинг у S3, потрібно вибрати пакетний хостинг і ввімкнено (рис. 4.7)

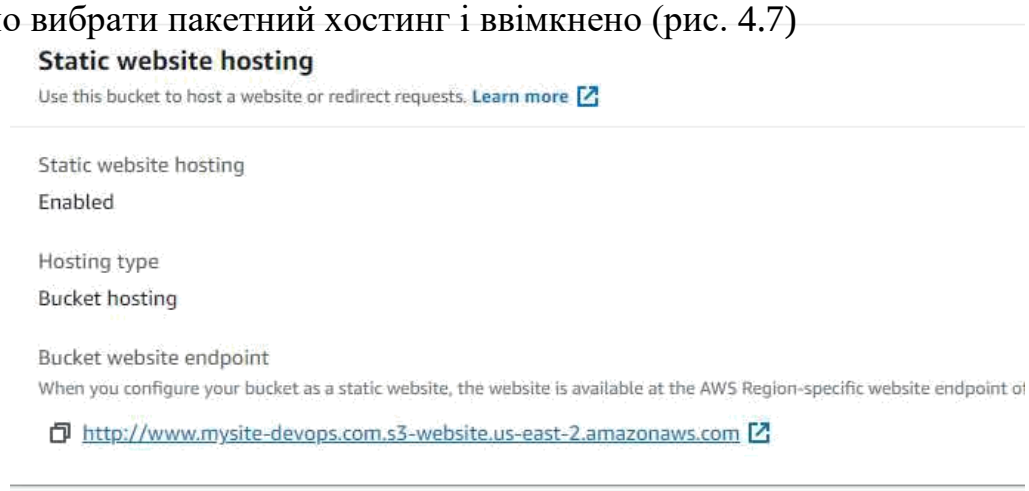


Рисунок 4.7 Налаштування хостингу в S3

Наступним кроком є налаштування статичного основного хостингу, а саме надання публічного доступу до сховища. За замовчуванням Amazon S3 блокує публічний доступ до статичного хостингу, і його потрібно надати додатково (рис. 4.8).

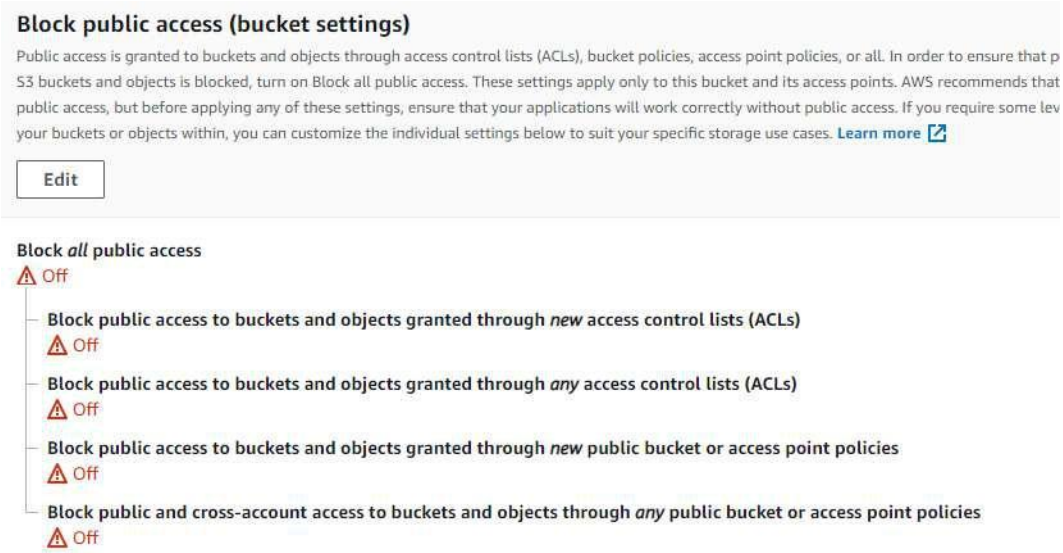


Рисунок 4.8 Налаштування прав доступу

Для того, щоб надати доступ до об'єктів у кошику, в політику потрібно додати наступний код:

```
{
   "Version": "2021-05-17",
```

```
    "Statement": [
      {
        "Sid": "PublicReadGetObject",
        "Effect": "Allow",
        "Principal": "*",
        "Action": "s3:GetObject",
        "Resource": "arn:aws:s3:::www.mysite-devops.com/*"
      }
    ]
}
```

З іншого боку, облікові записи AWS повинні бути додані до політики. Ось чому список контролю доступу (ACL) повинен управлятися належним чином, а операція CRUD повинна управлятися і контролюватися з боку різних користувачів, таких як власник бакетів, публічний доступ, група аутентифікованих користувачів з боку AWS і група доставки логів S3.

Для того, щоб реєструвати веб-трафік з боку AWS, необхідно створити окремі бакети, які будуть автоматично оновлюватися кожні дві години (рис. 4.9). Логи будуть записувати цілі запити, які виконуються до статичного хостингу.

```
{
  "Version": "2020-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::logs.mysite.com/*"
```

```
        }
    ]
}
```

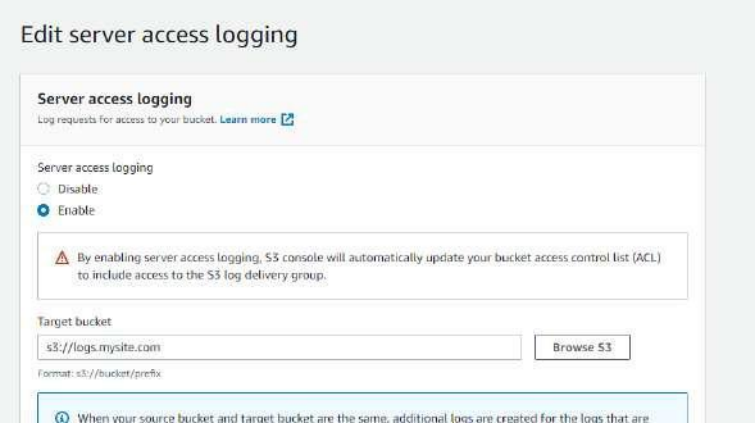


Рисунок 4.9 Журнал доступу до сервера AWS

Після створення та встановлення всіх необхідних налаштувань на стороні AWS, інструмент розгортання слід налаштувати відповідним чином за допомогою додаткового плагіна рис. 4.10

Рисунок 4.10 Плагін Jenkins S3

The log of execution can be following:

INFO: ----------

Publish artifacts to S3 Bucket Build is still running

Publish artifacts to S3 Bucket Using S3 profile: S3Deploy

Publish artifacts to S3 Bucket bucket=www.mysite-devops.com, file=.sonar_lock region=us-east-2, will be uploaded from slave=false managed=false , server encryption false

Finished: SUCCESS

В результаті, після публікації контенту до кошика, веб-сайт буде доступний за посиланням http://www.mysite-devops.com.s3-website.us-east-2.amazonaws.com/, а логи будуть доступні на окремому хостингу, як показано на рис. 4.11

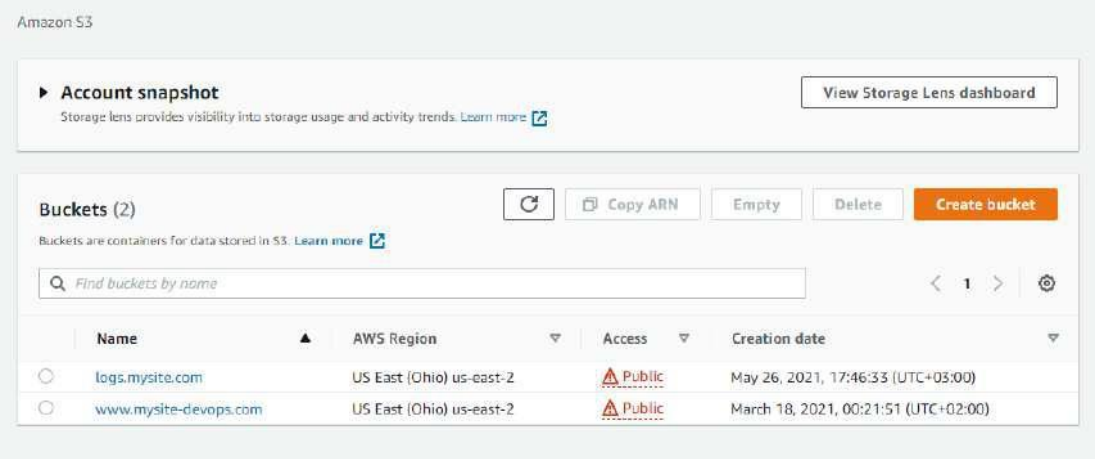


Рисунок 4.11 AWS S3

The example of code:

<Error>

<Code>AccessDenied</Code>

<Message>Access Denied</Message>

<RequestId>187MRE14TBFEBPNX</RequestId>

<HostId>

tNYlsF3mHvFvv+Afv2zqY8os1r1zXy/SyGDXq1aNMG8WFd9X3L8aglGKzLnlKCcc

4HL3Ty3R/+s=

</HostId>

</Error>

Натисніть на посилання нижче, щоб перейти на головну сторінку сайту (рис. 4.12).

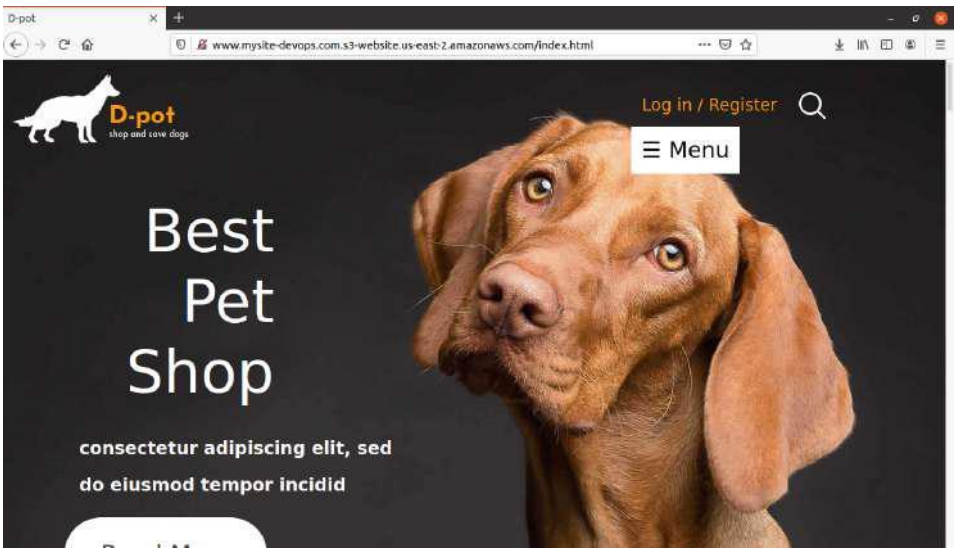


Рисунок 4.12 Веб-додаток, розміщений на S3 bucket

Завдяки подальшому розширенню можливостей веб-сайту, цей сайт може робити запити до інших систем для отримання додаткових даних, а також здійснювати додаткові дії, необхідні для роботи системи. Наприклад, географічне розташування або додаткові SEO-заходи, такі як реклама або вподобання клієнтів.

4.4 Моніторинг

Моніторинг буде підтримуватися Prometheus та Grafana. Для інтеграції наведених нижче інструментів необхідно встановити додаткові надбудови в Jenkins.

Для моніторингу я використовував докер-контейнери для запуску серверів. Після запуску сервісів, експортуйте дані з Jenkings, передаючи їх до Prometheus, а потім до Grafana.

Як я вже згадував, докер Prometheus працює після базової конфігурації докера. За замовчуванням пост для Prometeus - 9090. Встановлення було виконано за допомогою команди

docker run -d --name diplom_prometheus -p 9090:9090 prom/diplom_prometheus

До конвеєра було додано додатковий плагін. Плагін відкриває кінцеву точку з метрикою, з якої Prometheus Server може витягувати логи. Вивід плагіна має такий вигляд:

jvm_info{version="11.0.11+9-Ubuntu-0ubuntu2.20.04",vendor="Ubuntu",runtim e="OpenJDK Runtime Environment",} 1.0

HELP default_jenkins_builds_duration_milliseconds_summary Summary of Jenkins build times in milliseconds by Job

TYPE default_jenkins_builds_duration_milliseconds_summary summarydefault_jenkins_builds_duration_milliseconds_summary_count{jenkins_job="Diplom",repo="NA",} 1.0default_jenkins_builds_duration_milliseconds_summary_sum{jenkins_job="Diplom",repo="NA",} 5432.0default_jenkins_builds_duration_milliseconds_summary_count{jenkins_job="Job",repo="NA",} 2.0default_jenkins_builds_duration_milliseconds_summary_sum{jenkins_job="Job",repo="NA",} 674973.0

HELP default_jenkins_builds_success_build_count Successful build count

TYPE default_jenkins_builds_success_build_count counterdefault_jenkins_builds_success_build_count{jenkins_job="Diplom",repo="NA" ,} 1.0default_jenkins_builds_success_build_count{jenkins_job="Job",repo="NA",} 2.0

HELP default_jenkins_builds_health_score Health score of a job

TYPE default_jenkins_builds_health_score gaugedefault_jenkins_builds_health_score{jenkins_job="Diplom",repo="NA",} 100.0default_jenkins_builds_health_score{jenkins_job="Job",repo="NA",} 100.0

HELP default_jenkins_builds_last_build_result_ordinal Build status of a job.

TYPE default_jenkins_builds_last_build_result_ordinal gaugedefault_jenkins_builds_last_build_result_ordinal{jenkins_job="Diplom",repo="NA",}

0.0default_jenkins_builds_last_build_result_ordinal{jenkins_job="Job",repo="NA",}

0.0

HELP default_jenkins_builds_last_build_result Build status of a job as a boolean (0 or 1)

TYPE default_jenkins_builds_last_build_result gaugedefault_jenkins_builds_last_build_result{jenkins_job="Diplom",repo="NA",} 1.0default_jenkins_builds_last_build_result{jenkins_job="Job",repo="NA",} 1.0

HELP default_jenkins_builds_last_build_duration_milliseconds Build times in milliseconds of last build

TYPE default_jenkins_builds_last_build_duration_milliseconds gaugedefault_jenkins_builds_last_build_duration_milliseconds{jenkins_job="Diplom",repo="NA",}

5432.0default_jenkins_builds_last_build_duration_milliseconds{jenkins_job="Job",repo="NA",} 338198.0

HELP default_jenkins_builds_last_build_start_time_milliseconds Last build start timestamp in milliseconds

TYPE default_jenkins_builds_last_build_start_time_milliseconds gaugedefault_jenkins_builds_last_build_start_time_milliseconds{jenkins_job="Diplom",repo="NA",}

1.620147476553E12default_jenkins_builds_last_build_start_time_milliseconds{jenkins_job="Job",repo="NA",} 1.622409964268E12

HELP default_jenkins_builds_last_stage_duration_milliseconds_summary Summary of Jenkins build times by Job and Stage in the last build

TYPE default_jenkins_builds_last_stage_duration_milliseconds_summary summarydefault_jenkins_builds_last_stage_duration_milliseconds_summary_count{jenkins_job="Diplom",repo="NA",stage="Hello",} 1.0default_jenkins_builds_last_stage_duration_milliseconds_summary_sum{jenkins_job="Diplom",repo="NA",stage="Hello",} 219.0

To configure monitoring server, some yml file needs to be updated with next code:

- job_name: ‘jenkins’
  metrics_path: /prometheus
  static_configs:
  - targets: [‘http:\\jenkings\prometheus:9090’]

В результаті може бути створена комплексна інформаційна панель, яка буде доступна для всіх.

ВИСНОВКИ

У першому розділі було виявлено галузеві дослідження та аналіз впливу. Висвітлено переваги та недоліки різних підходів, що робить його можливість врахувати результати огляду в подальшому розвитку з метою розробити та створити вдосконалену архітектуру з урахуванням цих результатів. Друга частина першого розділу присвячена аналізу підходів до розробка архітектури, дослідження програмних продуктів, які можна використовувати як компоненти в архітектурі, зокрема їх можливості та методи інтеграція.

Другий розділ описує практичну реалізацію конвеєра CI\CD, в основі якого лежить розвинена архітектура. Jenkins Server використовувався для підтримки базових постулати, інструменти тестування для встановлення якісних шлюзів і хмара AWS для забезпечення продукту доступність широкому колу користувачів. При цьому всі ці компоненти інтегровані в один конвеєр і реалізована можливість роботи з конвеєрами на основі тригерів і розклади. За системою моніторингу налаштування та конфігурація для моніторингу та скоротити час вирішення проблем для бізнесу.

Отриману архітектуру можна розширити за допомогою нових компонентів необхідних. Зокрема, можуть бути додані компоненти повідомлень про етап конвеєра. Крім того, значний потенціал для розширення існує на етапі тестування, оскільки цей етап може включати різні види тестування та системи, які їх підтримують. Зокрема, підтримку для можна додати інтеграційні тести. Залежно від обраних мов програмування різні можна використовувати інструменти та підходи. Але основна увага завжди має бути на бізнесі вартість і швидкість ринку.

Загалом створений конвеєр CI\CD підтверджує актуальність та ефективність. Ця архітектура може слугувати прототипом для подальшого розширення та використовуватися для Інтернету розробка додатків. Розширити можливі підходи до зовнішніх систем і можна використовувати аддони.

Інформаційні технології є одними з найбільш динамічних і постійно розвиваються напрямків тому для будь-якого програмного рішення існують можливості для змін і розвитку. Через це можливо вдосконалення та додавання нових функцій до розробленої архітектури. бути частиною майбутньої роботи та може зосередитися на реалізації додаткових етапів і етапів pipeline.

Наприклад, Docker і Kubernetes можуть бути одним із напрямків вдосконалення. Такий підхід розширює конвеєр CI\CD і вносить нове бачення в безперервність інтеграційна сфера. З іншого боку, дослідження та покращення можуть розглянути розширення каналів зв'язку. Наприклад вайбер, телеграм чи будь-який інший корпоративні месенджери можна додати до конвеєра для надсилання сповіщень у відповідній формі спосіб привласнити людей.

ПЕРЕЛІК ПОСИЛАНЬ

1. **"Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation"**
Автори: Jez Humble, David Farley
Ця книга є класичним посібником з практик CI/CD і охоплює процеси автоматизації у корпоративному середовищі.
2. **"The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations"**
Автори: Gene Kim, Patrick Debois, John Willis, Jez Humble
Описує основи DevOps, у тому числі інтеграцію та доставку.
3. **"Java Performance: The Definitive Guide"**
Автор: Scott Oaks
Розглядає оптимізацію та продуктивність Java-додатків, що важливо для CI/CD у корпоративних проєктах.
4. **"Building Microservices: Designing Fine-Grained Systems"**
Автор: Sam Newman
Книга охоплює мікросервісну архітектуру, яка часто використовується разом із CI/CD.
5. **"Effective Java"**
Автор: Joshua Bloch
Рекомендована книга для розробників Java, щоб писати якісний код, що полегшує автоматизацію тестування та розгортання.

ДОДАТОК А

Started by user jenkins

Running as SYSTEM

Building in workspace
/var/lib/jenkins/workspace/Job The recommended git
tool is: NONE No credentials specified

> git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/Job/.git #
timeout=10 Fetching changes from the remote Git repository

> git config remote.origin.url https://github.com/Tetiana1234567/Diploma.git #
timeout=10
Fetching upstream changes from https://github.com/Tetiana1234567/Diploma.git

> git --version # timeout=10

> git --version # 'git version 2.25.1'

> git fetch --tags --force --progress -- https://github.com/Tetiana1234567/Diploma.git
+refs/heads/*:refs/remotes/origin/* # timeout=10

> git rev-parse refs/remotes/origin/main^{commit} # timeout=10
Checking out Revision 7d55a9e7f1e5c4024a2f5e590f5969462e9262c3
(refs/remotes/origin/main)

> git config core.sparsecheckout # timeout=10
> git checkout -f 7d55a9e7f1e5c4024a2f5e590f5969462e9262c3 #
timeout=10 Commit message: "Initiation"
> git rev-list --no-walk 7d55a9e7f1e5c4024a2f5e590f5969462e9262c3 #
timeout=10 [Job] $ /bin/sh -xe /tmp/jenkins15679617849493207118.sh

+ echo ----------Build Started----------

----------Build Started----------

+ ls -la
total 80
drwxr-xr-x 7 jenkins jenkins 4096 тра 3 20:50 .
drwxr-xr-x 4 jenkins jenkins 4096 тра 5 20:35 ..

-rw-r--r-- 1 jenkins jenkins 8292 кві 29 22:29 about.html

-rw-r--r-- 1 jenkins jenkins 9936 кві 29 22:29 contact.html

drwxr-xr-x 2 jenkins jenkins 4096 кві 29 22:29 css

-rw-r--r-- 1 jenkins jenkins 11264 кві 29 22:29 gallery.html

drwxr-xr-x 8 jenkins jenkins 4096 тра 5 21:38 .git

drwxr-xr-x 2 jenkins jenkins 4096 кві 29 22:29 images

-rw-r--r-- 1 jenkins jenkins 14575 кві 29 22:29 index.html

drwxr-xr-x 2 jenkins jenkins 4096 кві 29 22:29 js

drwxr-xr-x 3 jenkins jenkins 4096 тра 5 21:25 .scannerwork

+ echo This build is number: 86
This build is number: 86

+ echo Name of the build is: #86
Name of the build is: #86
+ echo ----------Build Finished----------

----------Build Finished----------

[Job] $ /var/lib/jenkins/tools/hudson.plugins.sonar.SonarRunnerInstallation/Sonarscanner/bin/so

nar-scanner -Dsonar.host.url=http://localhost:9000 ********

-Dsonar.projectKey=Diploma -Dsonar.projectName=Diploma

-Dsonar.projectVersion=1.0 -Dsonar.sources=.

-Dsonar.projectBaseDir=/var/lib/jenkins/workspace/Job

INFO: Scanner configuration file: /var/lib/jenkins/tools/hudson.plugins.sonar.SonarRunnerInstallation/Sonarscanner/conf/s

onar-scanner.properties

INFO: Project root configuration file: NONE

INFO: SonarScanner 4.6.1.2450

INFO: Java 11.0.11 Ubuntu (64-bit)

INFO: Linux 5.8.0-50-generic amd64

INFO: User cache: /var/lib/jenkins/.sonar/cache

INFO: Scanner configuration file:

/var/lib/jenkins/tools/hudson.plugins.sonar.SonarRunnerInstallation/Sonarscanner/conf/sonar-scanner.properties

INFO: Project root configuration file: NONE

INFO: Analyzing on SonarQube server 8.8.0

INFO: Default locale: "en_US", source code encoding: "UTF-8" (analysis is platform dependent)

INFO: Load global settings

INFO: Load global settings (done) | time=72ms

INFO: Server id: BF41A1F2-AXkyEzeA7vbvplIfljj7

INFO: User cache: /var/lib/jenkins/.sonar/cache

INFO: Load/download plugins

INFO: Load plugins index

INFO: Load plugins index (done) | time=42ms

INFO: Load/download plugins (done) | time=127ms

INFO: Process project properties

INFO: Process project properties (done) | time=8ms

INFO: Execute project builders

INFO: Execute project builders (done) | time=1ms

INFO: Project key: Diploma

INFO: Base dir: /var/lib/jenkins/workspace/Job

INFO: Working dir: /var/lib/jenkins/workspace/Job/.scannerwork

INFO: Load project settings for component key: 'Diploma'

INFO: Load project settings for component key: 'Diploma' (done) | time=14ms

INFO: Auto-configuring with CI 'Jenkins'

INFO: Load quality profiles

INFO: Load quality profiles (done) | time=48ms

INFO: Auto-configuring with CI 'Jenkins'

INFO: Load active rules

INFO: Load active rules (done) | time=1390ms

INFO: Indexing files...

INFO: Project configuration:

INFO: Load project repositories

INFO: Load project repositories (done) | time=14ms

WARN: Invalid character encountered in file /var/lib/jenkins/workspace/Job/js/plugin.js at line 128 for encoding UTF-8. Please fix file content or configure the encoding to be used using property 'sonar.sourceEncoding'.

INFO: 37 files indexed

INFO: 0 files ignored because of scm ignore settings

INFO: Quality profile for css: Sonar way

INFO: Quality profile for js: Sonar way

INFO: Quality profile for web: Sonar way

INFO: ---------- Run sensors on module Diploma

INFO: Load metrics repository

INFO: Load metrics repository (done) | time=31ms

INFO: Sensor CSS Metrics [cssfamily]

INFO: Sensor CSS Metrics [cssfamily] (done) | time=358ms

INFO: Sensor CSS Rules [cssfamily]

INFO: Sensor CSS Rules [cssfamily] (done) | time=661ms

INFO: Sensor JaCoCo XML Report Importer [jacoco]

INFO: 'sonar.coverage.jacoco.xmlReportPaths' is not defined. Using default locations: target/site/jacoco/jacoco.xml,target/site/jacoco-it/jacoco.xml,build/reports/jacoco/test/jacocoTestReport.xml

INFO: No report imported, no coverage information will be imported by JaCoCo XML Report Importer

INFO: Sensor JaCoCo XML Report Importer [jacoco] (done) | time=4ms

INFO: Sensor JavaScript analysis [javascript]

INFO: Sensor JavaScript analysis [javascript] (done) | time=1192ms

INFO: Sensor C# Project Type Information [csharp]

INFO: Sensor C# Project Type Information [csharp] (done) | time=2ms

INFO: Sensor C# Properties [csharp]

INFO: Sensor C# Properties [csharp] (done) | time=1ms

INFO: Sensor JavaXmlSensor [java]

INFO: Sensor JavaXmlSensor [java] (done) | time=3ms

INFO: Sensor HTML [web]

INFO: Sensor HTML [web] (done) | time=472ms

INFO: Sensor VB.NET Project Type Information [vbnet]

INFO: Sensor VB.NET Project Type Information [vbnet] (done) | time=1ms

INFO: Sensor VB.NET Properties [vbnet]

INFO: Sensor VB.NET Properties [vbnet] (done) | time=1ms

INFO: ---------- Run sensors on project

INFO: Sensor Zero Coverage Sensor

INFO: Sensor Zero Coverage Sensor (done) | time=1ms

INFO: CPD Executor Calculating CPD for 4 files

INFO: CPD Executor CPD calculation finished (done) | time=41ms

INFO: Analysis report generated in 94ms, dir size=386 KB

INFO: Analysis report compressed in 53ms, zip size=101 KB

INFO: Analysis report uploaded in 72ms

INFO: ANALYSIS SUCCESSFUL, you can browse
http://localhost:9000/dashboard?id=Diploma

INFO: Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report

INFO: More about the report processing at
http://localhost:9000/api/ce/task?id=AXk90rc47vbvplIfloms INFO: Analysis total time: 7.042 s

INFO: --

INFO: EXECUTION SUCCESS

INFO: --

INFO: Total time: 8.272s

INFO: Final Memory: 7M/37M

INFO: --

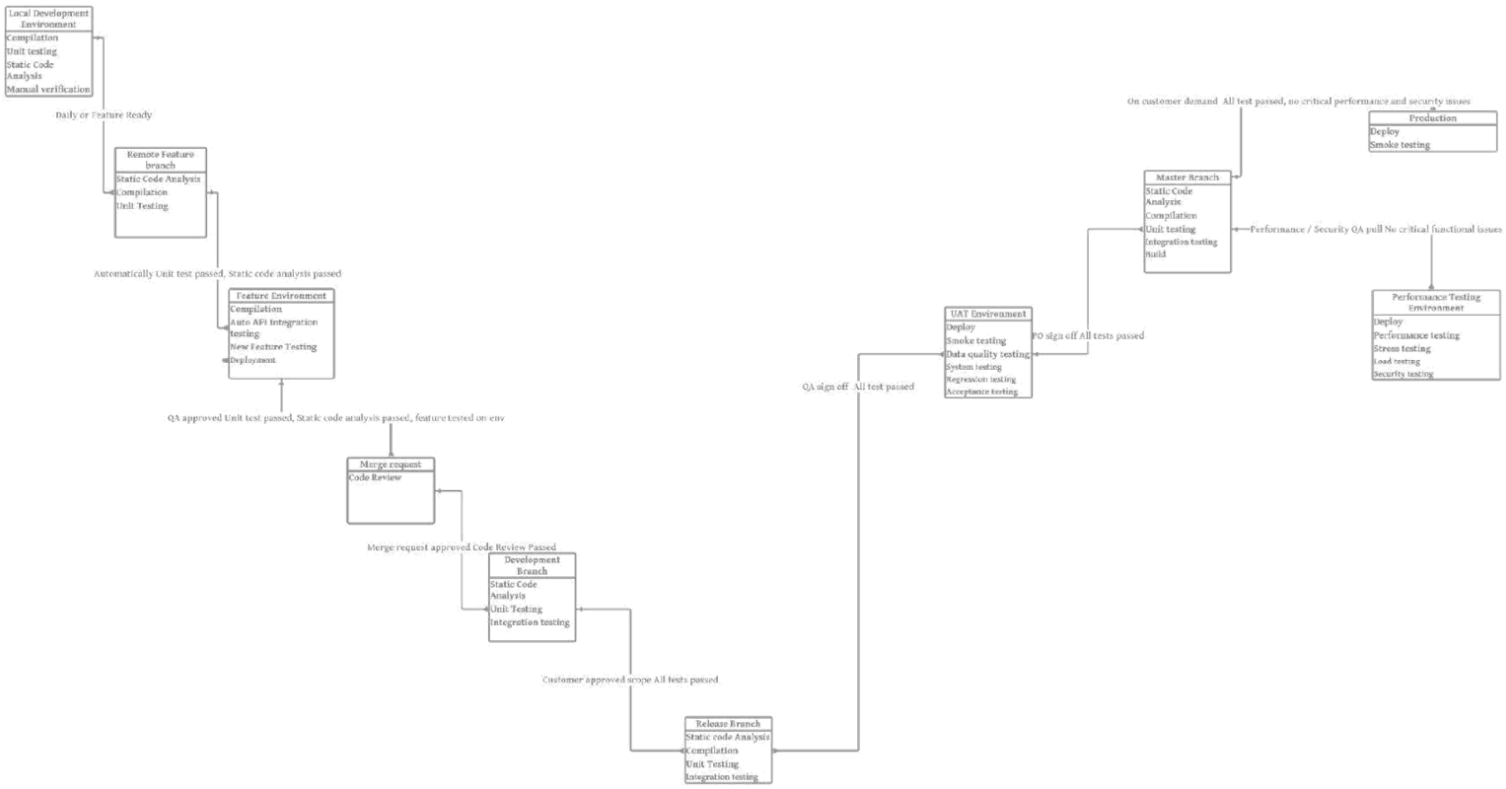
Publish artifacts to S3 Bucket Build is still
running FINISHED: SUCCESS

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (ПРЕЗЕНТАЦІЯ)

```
provider "aws" {

  profile    = "default"

  region     = "us-east-2"

  access_key = "AKIAYQUWEZ37V2VC4R4R"

  secret_key = "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"


}



resource "aws_instance" "Ubuntu" {

  count  = 1

  ami  = "ami-01e7ca2ef94a0ae86"

  instance_type = "t2.micro"

  vpc_security_group_ids = ["sg-05118f212a2ddd293"]
  associate_public_ip_address = "true"

  key_name = "newkey"

  provisioner "remote-exec" {

    inline = [

      "wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add-",
```

```
      "sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'",

      "sudo apt update -qq",

      "sudo apt install -y default-jre",

      "sudo apt install -y jenkins",

      "sudo systemctl start jenkins",

      "sudo iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT --to-port 8080",
      "sudo sh -c \"iptables-save > /etc/iptables.rules\"",

      "echo iptables-persistent iptables-persistent/autosave_v4 boolean true | sudo debconf-set-selections",

      "echo iptables-persistent iptables-persistent/autosave_v6 boolean true | sudo debconf-set-selections",

      "sudo apt-get -y install iptables-persistent",

      "sudo ufw allow 8080",

    ]

  }
  connection {
    type        = "ssh"
    host        = self.public_ip
    user        = "ubuntu"
    private_key = file("newkey.pem")
```

```
  }
  tags = {
    "Name"      = "Jenkins_Server"
    "Terraform" = "true"
  }
}
resource "aws_instance" "Jira" {
  count                  = 1
  ami                    = "ami-01e7ca2ef94a0ae86"
  instance_type          = "t2.micro"
  vpc_security_group_ids = ["sg-05118f212a2ddd293"]
  associate_public_ip_address = "true"
  key_name= "newkey"}
```


Local Development Environment
Compilation
Unit testing
Static Code Analysis
Manual verification
Daily or Feature Ready
Remote Feature branch
Static Code Analysis
Compilation
Unit Testing
Automatically Unit test passed, Static code analysis passed
Feature Environment
Compilation
Auto API Integration testing
New Feature Testing
Deployment
QA approved Unit test passed, Static code analysis passed, feature tested on env
Merge request
Code Review
Merge request approved Code Review Passed
Development Branch
Static Code Analysis
Unit Testing
Integration testing
Customer approved scope All tests passed
Release Branch
Static code Analysis
Compilation
Unit Testing
Integration testing
QA sign off All test passed
UAT Environment
Deploy
Smoke testing
Data quality testing
System testing
Regression testing
Acceptance testing
PO sign off All tests passed
Master Branch
Static Code Analysis
Compilation
Unit testing
Integration testing
Build
On customer demand All test passed, no critical performance and security issues
Production
Deploy
Smoke testing
Performance / Security QA pull No critical functional issues
Performance Testing Environment
Deploy
Performance testing
Stress testing
Load testing
Security testing

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

ДИПЛОМНА РОБОТА
на ступінь вищої освіти магістр
із спеціальності 122 Комп'ютерні науки

СИСТЕМА CONTINUOUS INTEGRATION/CONTINUOUS DELIVERY У КОРПОРАТИВНИХ СУБ'ЄКТАХ НА МОВІ ПРОГРАМУВАННІ JAVA

Виконав: студент 6 курсу, групи КНДМ-62,
Чистяков Богдан Олександрович
Керівник: Завідувач кафедри Штучного інтелекту д.т.н, доцент ЗІНЧЕНКО О. В.

Київ - 2024

ЗАГАЛЬНА ХАРАКТЕРИСТИКА ДИПЛОМНОЇ РОБОТИ

• Тема	Система Continuous Integration/ContinuousDelivery у корпоративних суб'єктах на мові програмування Java.
• Мета дослідження	Розробка та впровадження системи CI/CD для корпоративних суб'єктів на основі мови програмування Java, що забезпечує автоматизацію процесів розробки, тестування та доставки програмного забезпечення.
• Наукове завдання	Дослідження інструментів і технологій для побудови CI/CD, сумісних із Java.
• Об'єкт дослідження	Процеси автоматизації розробки, тестування та доставки програмного забезпечення у корпоративному середовищі.
• Предмет дослідження	Методи, підходи та технології побудови систем CI/CD на основі мови програмування Java.

Актуальні проблеми в галузі

- **Складність інтеграції з існуючими системами**
  Багато корпоративних мереж мають застарілі системи, які важко адаптувати до сучасних CI/CD-підходів.
  Необхідність модернізації інфраструктури може бути дорогою та трудомісткою.

- **Проблеми з конфіденційністю та безпекою**
  Інтеграція CI/CD вимагає роботи з конфіденційними даними, наприклад, обліковими записами, токенами чи ключами доступу. Неправильне налаштування може призвести до витоків даних або вразливостей.

- **Продуктивність і масштабованість**
  Виконання автоматизованих тестів та побудови артефактів у великих Java-проєктах може бути ресурсомістким.
  Обмеження обчислювальних потужностей корпоративних серверів.

- **Залежність від інструментів і технологій**
  Велика кількість доступних рішень для CI/CD ускладнює вибір оптимального інструменту для конкретного корпоративного середовища. Залежність від сторонніх платформ може створювати додаткові ризики.

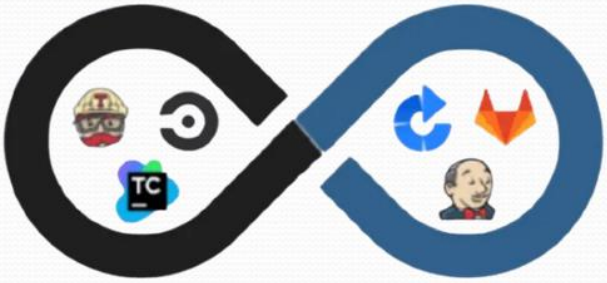
- **Брак кваліфікованих спеціалістів**
  Нестача досвідчених DevOps-інженерів із глибокими знаннями Java та сучасних CI/CD-технологій.
  Труднощі з навчанням існуючих кадрів для роботи з новими процесами.

- **Складність підтримки великих монолітних Java-додатків**
  Багато корпоративних проєктів досі побудовані як моноліти, що ускладнює їх розбиття на мікросервіси для ефективного CI/CD.

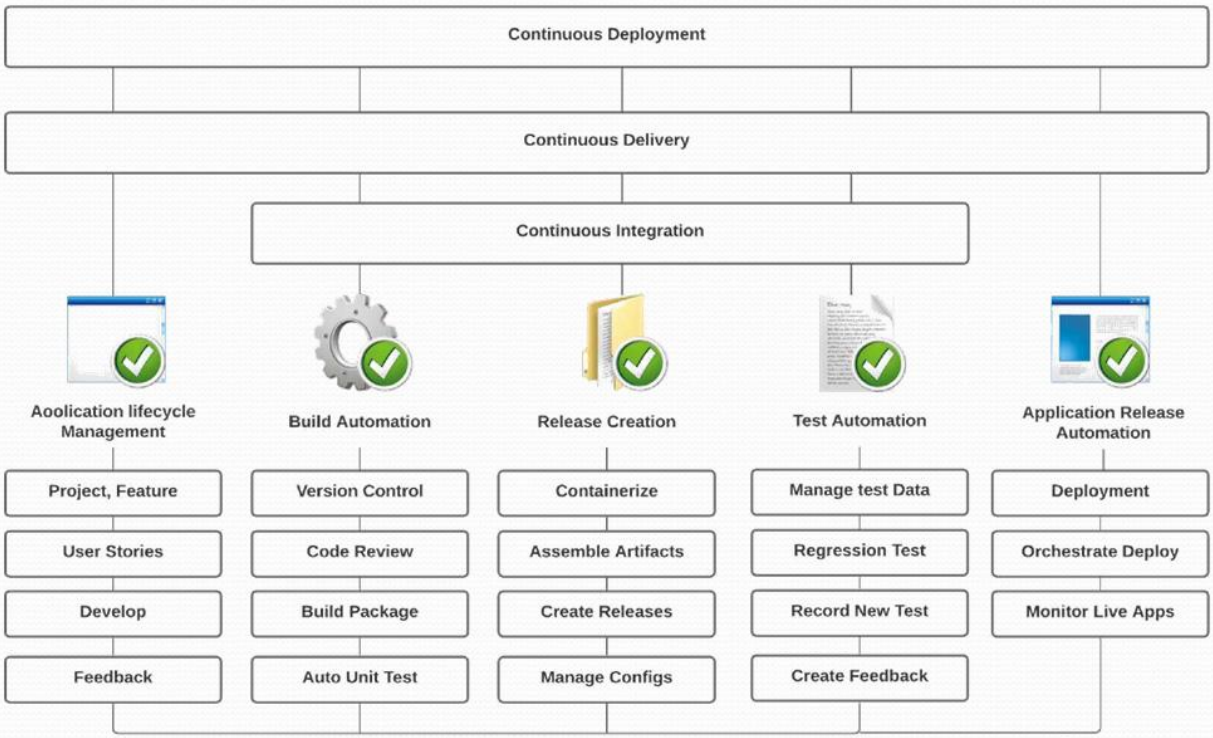
- **Тестування та забезпечення якості**
  Потреба в автоматизації складних інтеграційних тестів для Java-додатків.
  Висока тривалість виконання тестів у великих проєктах.

Огляд існуючих рішень для CI/CD

- **Jenkins**
  - Популярний інструмент із відкритим кодом.
  - Підтримує безліч плагінів для автоматизації побудови, тестування та розгортання.
- **GitLab CI/CD**
  - Інтегрований у платформу GitLab.
  - Забезпечує повний цикл DevOps із підтримкою pipeline-конвеєрів.
- **GitHub Actions**
  - CI/CD-інструмент, вбудований у GitHub.
  - Простий у налаштуванні для репозиторіїв GitHub.
- **CircleCI**
  - Хмарне рішення з гнучкою конфігурацією.
  - Добре підходить для масштабованих проєктів.
- **Azure DevOps**
  - Платформа від Microsoft для повного циклу розробки.
  - Інтегрується з екосистемою Microsoft та іншими сервісами.
- **TeamCity**
  - Інструмент від JetBrains із широкими можливостями конфігурації та інтеграції.
  - Підтримує великі та складні проєкти.



Схема, компоненти та їх взаємодія



Приклад взаємодії