МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ
ФАКУЛЬТЕТ МЕХАТРОНІКИ ТА КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Автоматизація процесів розроблення та впровадження програмного забезпечення за допомогою DevOps

Рівень вищої освіти другий (магістерський)
Спеціальність 122 Комп'ютерні науки
Освітня програма Комп'ютерні науки

Виконав: студент групи МгІТ-1-22
Сергій ОЛІЙНИК
Науковий керівник: к.т.н., доцент
Тетяна ДЕМКІВСЬКА
Рецензент: д.т.н., професор
Віктор ЧУПРИНКА

Київ 2023

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

Факультет	мехатроніки та комп'ютерних технологій
Кафедра	комп'ютерних наук
Рівень вищої освіти	другий (магістерський)
Спеціальність	122 Комп'ютерні науки
Освітня програма	Комп'ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри КН

проф. Володимир ЩЕРБАНЬ

"____" _____ 2023 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Олійнику Сергію Сергійовичу

1. Тема роботи: Автоматизація процесів розроблення та впровадження програмного забезпечення за допомогою DevOps.

Науковий керівник роботи Демківська Тетяна Іванівна, к.т.н., доцент,

затверджені наказом КНУТД від 12 вересня 2023 року № 210-уч

2. Вихідні дані до кваліфікаційної роботи: розробки кафедри комп'ютерних наук, рекомендована література, офіційна та неофіційна документація для інструментів, застосованих в рамках роботи.

3. Зміст кваліфікаційної роботи: Перелік умовних позначень; Вступ; Розділ 1. Методології розробки програмного забезпечення. концепції CI, CD. постановка задачі; Розділ 2. Аналіз доступних на ринку інструментів побудови CI/CD конвеєру; Розділ 3. Практична імплементація CI/CD конвеєру на основі обраного інструменту; Висновки; Список використаних джерел; Додатки.

4. Дата видачі завдання: 08.2023

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної магістерської роботи	Терміни виконання етапів	Примітка про виконання
1	Вступ	30.08.2023	
2	Розділ 1. Методології розробки програмного забезпечення. концепцій CI, CD. постановка задачі	06.09.2023	
3	Розділ 2. Аналіз доступних на ринку інструментів побудови CI/CD конвеєру	28.09.2023	
4	Розділ 3. Практична імплементація CI/CD конвеєру на основі обраного інструменту	21.10.2023	
5	Висновки	29.10.2023	
6	Оформлення кваліфікаційної роботи (чистовий варіант)	06.11.2023	
7	Подача кваліфікаційної роботи науковому керівнику для відгуку (за 14 днів до захисту)	08.11.2023	
8	Подача кваліфікаційної роботи для рецензування	08.11.2023	
9	Перевірка кваліфікаційної роботи на наявність ознак плагіату	09.11.2023	
10	Подання кваліфікаційної роботи на затвердження завідувачу кафедри	10.11.2023	

З завданням ознайомлений:

Студент _____ Сергій ОЛІЙНИК

Науковий керівник роботи _____ Тетяна ДЕМКІВСЬКА

АНОТАЦІЯ

Олійник С.С. Автоматизація процесів розроблення та впровадження програмного забезпечення за допомогою DevOps

Кваліфікаційна робота за спеціальністю 122 – «Комп'ютерні науки». – Київський національний університет технологій та дизайну, Київ, 2023 рік.

Кваліфікаційна робота присвячена дослідженню методологій розробки програмного забезпечення та аналізу концепцій безперервної інтеграції (CI) та безперервної доставки (CD). Основна увага була зосереджена на розробці та практичній імплементації CI/CD конвеєру з використанням інструменту Jenkins.

У ході кваліфікаційної роботи були вивчені та проаналізовані різні методології розробки ПЗ, включаючи традиційні та сучасні підходи, такі як Waterfall, V-Model, Agile та DevOps. Детально розглянуто переваги та недоліки впровадження практик CI/CD у процес розробки ПЗ. Також було проведено порівняльний аналіз доступних на ринку інструментів для побудови CI/CD конвеєрів, включаючи Jenkins, GitLab, Travis CI, та Go CD.

Практична частина роботи включала розгортання Jenkins, побудову та налаштування CI/CD конвеєру, включаючи конфігурацію плагінів, агентів та налаштування робочих процесів для автоматизації збирання, тестування та розгортання програмного забезпечення.

Результати роботи показали ефективність застосування CI/CD методологій та інструментів у сучасній розробці ПЗ, підкреслюючи їх здатність підвищувати швидкість розробки, знижувати ризики помилок та покращувати якість кінцевих продуктів.

Ключові слова: методології розробки ПЗ, безперервна інтеграція, безперервна доставка, CI/CD, Jenkins, автоматизація розробки.

ANNOTATION

Serhii Oliinyk, Automation of software development and deployment processes using DevOps.

Qualification Work in Computer Science, Kyiv National University of Technologies and Design, Kyiv, 2022.

This work is devoted to the study of software development methodologies and the analysis of Continuous Integration (CI) and Continuous Delivery (CD) concepts. The focus is on the development and practical implementation of a CI/CD pipeline using the Jenkins tool.

Throughout the work, various software development methodologies were examined and analyzed, including both traditional and modern approaches such as Waterfall, V-Model, Agile, and DevOps. The advantages and disadvantages of implementing CI/CD practices in the software development process were thoroughly considered. A comparative analysis of the tools available in the market for building CI/CD pipelines, including Jenkins, GitLab, Travis CI, and Go CD, was also conducted.

The practical part of the work involved the deployment of Jenkins, construction, and configuration of a CI/CD pipeline, including the setup of plugins, agents, and the configuration of workflows for automating the assembly, testing, and deployment of software.

The results of the work demonstrated the effectiveness of applying CI/CD methodologies and tools in modern software development, highlighting their ability to increase development speed, reduce the risk of errors, and improve the quality of the final products.

Keywords: software development methodologies, continuous integration, continuous delivery, CI/CD, Jenkins, software development automation.

ЗМІСТ

ВСТУП

Актуальність роботи. У сучасному світі, де технології швидко розвиваються, забезпечення ефективності процесів розробки та впровадження програмного забезпечення є ключовим аспектом для IT-компаній. Впровадження концепцій безперервної інтеграції (CI) та безперервної доставки (CD) та методологій Agile та DevOps відіграють важливу роль у підвищенні швидкості, гнучкості та якості розробки ПЗ. Це, у свою чергу, вимагає глибокого розуміння та аналізу доступних інструментів та методологій та практичне освоєння їх на навчальних на реальних проектах.

Мета дослідження. Метою цього дослідження є розробка та аналіз ефективного CI/CD конвеєру з використанням методології DevOps, що забезпечує оптимізацію процесів розробки та впровадження програмного забезпечення.

Об'єкт дослідження. Об'єктом дослідження є процес розробки та впровадження програмного забезпечення в рамках DevOps-підходу.

Предмет дослідження. Предметом дослідження є методи та інструменти безперервної інтеграції та доставки в контексті DevOps.

Методи дослідження. Дослідження базується на аналізі наукових публікацій, практичному впровадженні та тестуванні CI/CD конвеєрів, а також порівняльному аналізі інструментів та підходів, що використовуються в DevOps.

Наукова новизна. Наукова новизна полягає у розробці інтегрованого підходу до створення та оптимізації CI/CD конвеєру, який враховує специфіку DevOps і забезпечує гнучкість та ефективність процесів розробки та доставки ПЗ.

Практична значимість отриманих результатів. Отримані результати мають велике практичне значення для IT-компаній, оскільки забезпечують зниження витрат, підвищення швидкості розробки та поліпшення якості кінцевого продукту, що є важливим у конкурентному технологічному середовищі.

РОЗДІЛ 1. МЕТОДОЛОГІЇ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ. КОНЦЕПЦІЇ CI, CD. ПОСТАНОВКА ЗАДАЧІ

1.1. Етапи життєвого циклу програмного забезпечення

Життєвий цикл програмного забезпечення (ПЗ) представляє собою модель його створення та використання. Вона відображає різні стадії ПЗ, починаючи від моменту виявлення потреби в ньому і завершуючи моментом його випуску в публічний доступ. Різні моделі життєвого циклу відрізняються способами взаємодії між етапами, але кожен з цих етапів присутній у кожній з моделей.

Цикл розробки, розгортання та тестування ПЗ може варіюватися залежно від обраної методології розробки, наприклад, Waterfall, Agile чи Scrum. Проте, загальний цикл включає такі фази[15]:

- **планування:** етап, що включає визначення обсягу, цілей, досягнень і графіка проекту, ідентифікацію зацікавлених сторін, ресурсів, ризиків та обмежень;

- **аналіз:** етап, що включає збір та аналіз вимог і специфікацій проекту, розробку архітектури та структури ПЗ або системи;

- **розробка:** етап, що охоплює реалізацію ПЗ або системи відповідно до дизайну та специфікацій, проведення одиничних та інтеграційних тестів;

- **доставка:** етап, що передбачає розгортання ПЗ на цільову платформу або середовище, управління конфігурацією та релізами;

- **тестування:** етап, що включає проведення комплексного тестування ПЗ, звітування та усунення виявлених дефектів або проблем;

- **підтримка:** етап, що передбачає надання підтримки та оновлень для ПЗ після розгортання, включаючи коригувальне, адаптивне та профілактичне обслуговування.

Процес розробки, впровадження та тестування програмного забезпечення представляє собою ітеративний та безперервний процес, який вимагає постійного зв'язку та співпраці між усіма заінтересованими сторонами, включаючи розробників, тестувальників та кінцевих користувачів. Цей процес спрямований на створення програмних продуктів, які задовольняють потреби та очікування клієнтів, одночасно забезпечуючи високу якість, ефективність та надійність[15].

Кожен етап у цьому циклі має значний вплив на успішність загального процесу розробки. Ефективне та надійне виконання цих етапів є ключовим фактором у створенні конкурентоспроможних продуктів на сучасному ринку. У контексті швидкого розвитку технологій компаніям необхідно прискорювати процеси розробки програмного забезпечення, а відсутність впровадження сучасних методологій може призвести до негативних наслідків та зниження конкурентоспроможності на ринку.

Серед викликів, пов'язаних з традиційними підходами до розробки програмного забезпечення, можна виділити наступні[4]:

- брак ефективної комунікації, що веде до непорозумінь і затримок у роботі.

- ручне тестування та розгортання, затратні по часу і схильні до помилок через відмінності в середовищах розробки.

- неоптимальне управління версіями, яке призводить до рідкісних, але масштабних змін, ускладнюючи відстеження помилок.

- ручне розгортання оновлень ПЗ, яке є складним і схильним до помилок через ризики, пов'язані з конфігурацією та сумісністю.

- розбіжності між середовищами розробки, тестування, стейджингу та продакшну, що ускладнює точне відтворення та тестування реальних умов і збільшує ризик помилок.

• відсутність ефективного зворотного зв'язку між командами розробників і операційних спеціалістів, що позбавляє можливості обміну важливою інформацією про продукт, сповільнює ітерації, покращення якості коду та ефективне усунення можливих збоїв.

1.2. Огляд методологій розробки ПЗ

Методологія розробки ПЗ - це комплексний підхід, що встановлює послідовність виконання задач, а також методи оцінювання та управління процесом. Вибір конкретної моделі розробки залежить від специфіки проекту, включно з бюджетом, термінами розробки продукту, а також від особливостей керівника та команди розробників[15]. Різні підходи до розробки програмного забезпечення

відрізняються за способом взаємодії етапів життєвого циклу програмного продукту у межах процесу розробки. Розглянемо популярні методології розробки ПЗ.

1.2.1. Водоспадна модель (Waterfall)

Ця модель базується на послідовному проходженні етапів життєвого циклу ПЗ: від аналізу і дизайну до розробки, тестування, випуску та підтримки. Кожен етап повинен бути повністю завершений перед переходом до наступного. Однією з переваг цієї моделі є можливість оцінювати якість на кожному етапі. Проте, реалізувати проект за такою строго послідовною схемою часто є нереалістичним, через що модель може вважатись застарілою[15].

Однак, водоспадна модель обмежує можливість внесення змін на пізніших етапах, що ускладнює тестування, яке найчастіше проводиться наприкінці. Це може призвести до збільшення бюджету та затримок у випуску проекту. Цю модель рекомендується застосовувати лише у невеликих проектах, де немає ризику розбіжностей з вимогами. Водоспадна модель не завжди ефективна для розробки ПЗ.

Як видно на рис. 1.1, у водоспадній моделі наступна фаза розпочинається тільки після завершення попередньої. У цій моделі кожна фаза чітко обмежена, а їх послідовне виконання нагадує рух води у водоспаді, звідки й пішла назва.

Ця методологія підходить для таких умов:

- стабільні вимоги, які не схильні до частих змін;
- невеликі проекти;
- чітко визначені та зрозумілі вимоги;
- стабільне середовище;
- незмінні інструменти та методики;
- кваліфіковані та доступні ресурси.

Переваги водоспадної моделі включають:

- простоту та зрозумілість у використанні;
- ефективність для менших проектів з відповідними результатами;
- легкість у підтримці завдяки чітко визначеним фазам;
- чітко визначені критерії входу та виходу, які сприяють забезпеченню якості;

- добре документування результатів.

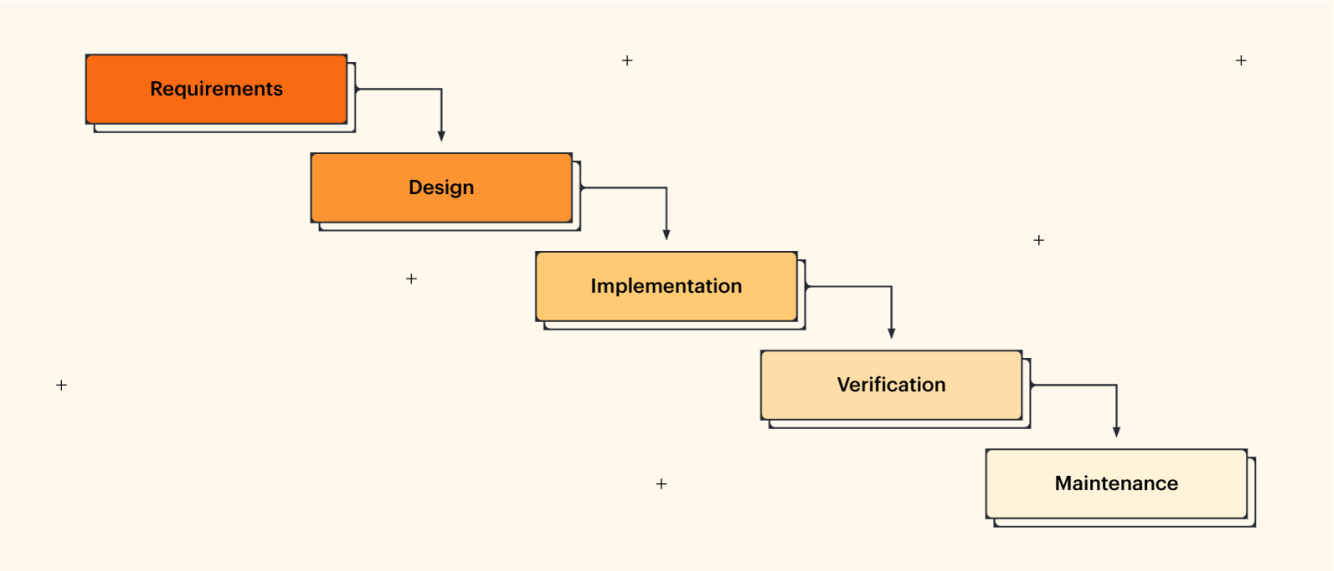


Рис. 1.1. Методологія Waterfall

Авжеж, ця застаріла методологія розробки ПЗ має численні недоліки, які обмежують її застосування у сучасних проектах:

- вимоги не можна змінити після їх схвалення;

- повернутися до попередньої фази важко, наприклад, якщо виявляється потреба в зміні на етапі тестування;

- затримки у доставці кінцевого продукту, оскільки немає проміжного прототипу;

- не підходить для великих та складних проектів через високий ризик;

- неефективна для проектів з частими змінами вимог;

- непридатна для тривалих та постійних проектів;

- оскільки тестування відбувається на пізніх етапах, це ускладнює виявлення проблем та ризиків на ранніх стадіях, що ускладнює планування стратегій зменшення ризиків.

1.2.2. «V-Model» (Крок за кроком або validation and verification)

Ключовим недоліком водоспадної моделі є пізнє виявлення дефектів, оскільки тестування відбувається лише на завершальній стадії розробки. Це робить виправлення помилок складним та затратним процесом. В якості відповіді на цю

проблему була розроблена V-модель, яка отримала свою назву через характерну форму структурної схеми, що нагадує літеру «V» (див. рис. 1.3).

Рис. 1.3. Схема V-моделі

Ця модель програмної інженерії зосереджена на ранній перевірці та тестуванні продукту. З самого початку розробки коду тестувальники розпочинають писати модульні тести, тобто вони працюють паралельно з розробниками. V-модель рекомендується у ситуаціях, де критично важливо безперебійне функціонування продукту і вимоги чітко визначені. Цей підхід також є частиною практик екстремального програмування[15].

Для розуміння V-моделі, важливо знати про «верифікацію та валідацію» у ПЗ. Верифікація - це статичний аналіз без виконання коду, включаючи огляди та перевірки. Валідація ж є динамічним аналізом, який включає виконання коду і включає функціональні та нефункціональні тести.

У V-моделі, контроль якості ведеться паралельно з розробкою, без виокремленої фази тестування. Тестування розпочинається з моменту встановлення вимог, а дії з верифікації та валідації виконуються паралельно.

1.2.3. Agile модель

Модель розробки програмного забезпечення **Agile** – це гнучкий підхід, який акцентує на швидкій адаптації до змін і неперервному покращенні. Головні особливості Agile включають[1]:

1. Ітеративний та інкрементний процес. Agile розробка відбувається в коротких циклах (ітераціях), що дозволяє швидко реагувати на зміни вимог та умов ринку.

2. Гнучкість та адаптивність. Замість жорсткого дотримання первісного плану, Agile дозволяє змінювати вимоги та пріоритети протягом усього процесу розробки.

3. Співпраця з замовником. Agile підкреслює важливість безпосередньої та частої взаємодії з замовником або користувачем для забезпечення більш точного відображення їх потреб у кінцевому продукті.

4. Крос-функціональні команди. Команди в Agile зазвичай багатофункціональні та самоорганізовані, з спільною відповідальністю за кінцевий продукт.

5. Неперервна зворотній зв'язок та самопокращення. Agile наголошує на регулярному перегляді продукту та процесу розробки, що сприяє неперервному покращенню.

6. Принципи, викладені в Agile Manifesto. Цей документ визначає ключові цінності та принципи Agile, такі як цінність індивідів та взаємодії, функціональний продукт, співпраця з клієнтами, та гнучкість перед обличчям змін.

Agile підходи, такі як Scrum, Kanban, та Extreme Programming (XP), пропонують конкретні методики та практики в рамках загальної філософії Agile. На рис. 1.3 зображена типова схема Agile-моделі, а нижче викладений загальний перелік етапів моделі.

Рис. 1.3. Узагальнена схема етапів розробки ПЗ у Agile моделі

Agile-модель розробки програмного забезпечення не має строго визначених етапів, як традиційні водоспадні або лінійні моделі. Однак, існують певні ключові фази, які зазвичай притаманні Agile-проектам[1]:

1. Аналіз вимог. Збір та аналіз детальних вимог від замовників чи користувачів. Цей етап часто переплітається з іншими фазами і продовжується протягом усього проекту.

2. Дизайн та розробка. Розробка продукту в невеликих інкрементах. Команда працює над завданнями, розбитими на короткі цикли (спринти), ітеративно створюює та поліпшує продукт.

3. Тестування. Проведення тестування паралельно з розробкою. Агільні команди часто використовують автоматизоване та ручне тестування для перевірки функціональності та якості продукту на кожній ітерації.

4. Впровадження. Після кожної ітерації (або декількох) продукт або його частина може бути випущена та розгорнута в середовищі користувача або на ринку.

5. Підтримка та розвиток. Під час кожної ітерації безперервно відбувається поточна підтримка роботоздатності продукту, а також формуються плани по розвитку та покращенню.

6. Зворотній зв'язок з замовником. Після кожної ітерації замовник має змогу оцінити поточну версію продукту і дати зворотній зв'язок команді розробників.

При такому підході кожна ітерація охоплює всі стадії життєвого циклу і готова до внесення змін. Проект поділяється на спринти - періоди часу, під час яких очікується певний результат. Зазвичай тривалість спринту становить від одного до чотирьох тижнів. В кожному спринті виконується визначений список завдань, причому кожне завдання має свою оцінку. Прогрес в реалізації завдань обговорюється на щоденних зустрічах, де команда ділиться інформацією про виконану роботу, плани на майбутнє та поточні проблеми. На початку спринту відбувається планувальна зустріч, де обговорюються завдання на ітерацію, а в кінці проводиться ретроспектива для аналізу результатів.

1.2.4. DevOps методологія

DevOps (скорочено від "Development" та "Operations") – це методологія в області розробки програмного забезпечення та інформаційних систем, яка прагне до зближення та оптимізації процесів розробки (Development) та експлуатації (Operations)[1][21].

Основна ідея DevOps полягає у створенні високоефективного середовища, де розробники програмного забезпечення та IT-спеціалісти, які займаються його експлуатацією та підтримкою, тісно співпрацюють, щоб забезпечити швидке, надійне та безперервне випускання продуктів[14].

Основні аспекти DevOps включають:

1. Культура співпраці. Сприяння більш тісній взаємодії між розробниками, IT-операціями та іншими відділами.

2. Автоматизація. Використання інструментів для автоматизації рутинних завдань, таких як розгортання, тестування, інтеграція тощо.

3. Неперервний розвиток (Continuous Development). Швидка та постійна розробка продуктів.

4. Неперервна інтеграція (Continuous Integration). Регулярне об'єднання змін у код, що дозволяє виявляти помилки на ранніх етапах.

5. Неперервна доставка або Неперервне розгортання (Continuous Delivery/Deployment). Автоматична доставка і розгортання програмного забезпечення у тестове (для розробників/тестувальників/членів команди) або користувацьке (для клієнта/користувачів) середовище.

6. Неперервне тестування (Continuous Testing). Постійне тестування продукту для гарантування якості.

7. Моніторинг та логування. Відстеження та аналіз роботи систем для оперативного виявлення та вирішення проблем.

8. Швидке виправлення помилок. Ефективне управління помилками та їх швидке усунення.

Метою DevOps є забезпечення більш швидкого та ефективного циклу розробки та випуску продуктів, підвищення якості та задоволення потреб клієнтів. DevOps використовує автоматизацію та інноваційні підходи для оптимізації процесів та забезпечення безперервної доставки високоякісних продуктів[21].

DevOps — це рух, що виник з метою забезпечити спільну роботу команд розробників та операційних спеціалістів для більш ефективного випуску якісного програмного забезпечення у коротші строки. Ідея DevOps бере свій початок від методології Agile, яка має на меті задовольняти потреби клієнта через ранню та неперервну доставку програмного забезпечення. Поняття DevOps було вперше представлено Патріком Дебоєм на Agile конференції в 2008 році, тоді як перша спеціалізована конференція DevOps відбулась в Бельгії у 2009 році. DevOps зорієнтований на застосування підходів і практик Agile до управління інфраструктурними процесами, автоматизуючи процеси побудови, тестування, інтеграції та доставки програмного забезпечення.

DevOps, що означає "Розробка та Операції", представляє собою підхід до розробки програмного забезпечення, націлений на покращення взаємодії та інтеграції між командами розробників і операційних спеціалістів. Основою методології є переконання, що тісна координація між цими групами є ключем до створення високоякісних програмних продуктів.

DevOps акцентує увагу на автоматизації процесів, культурних змінах та використанні сучасних інструментів та технологій, які спрощують процес розробки, тестування, розгортання та обслуговування програмного забезпечення.

Неперервна інтеграція та неперервна доставка (розгортання) – комплекс практик, інтегрованих у підхід DevOps, з акцентом на автоматизацію процесів розгортання програмного забезпечення.

Неперервна Інтеграція стосується процесу регулярної інтеграції змін у коді в спільний репозиторій. У межах CI розробники постійно об'єднують свої зміни, що автоматично збираються, тестуються і інтегруються з основною базою коду. Це сприяє виявленню проблем інтеграції на ранніх етапах, а також покращує співпрацю в команді.

Неперервна Доставка (Розгортання) є розвитком CI, автоматизуючи доставку або розгортання змін у коді в середовища розробки та виробництва. У рамках CD весь процес від внесення змін до їх кінцевого розгортання є автоматизованим, що забезпечує можливість швидкої та надійної доставки оновлень ПЗ. Ця автоматизація знижує ризик помилок, скорочує час на розгортання та забезпечує більш часті та послідовні оновлення.

На рисунку 1.4 представлено цикл розробки в командах, орієнтованих на DevOps, і показано, які практики застосовуються на кожному етапі. Важливо зазначити, що протягом усього циклу команда здійснює постійне спілкування і отримує зворотній зв'язок щодо завдань.

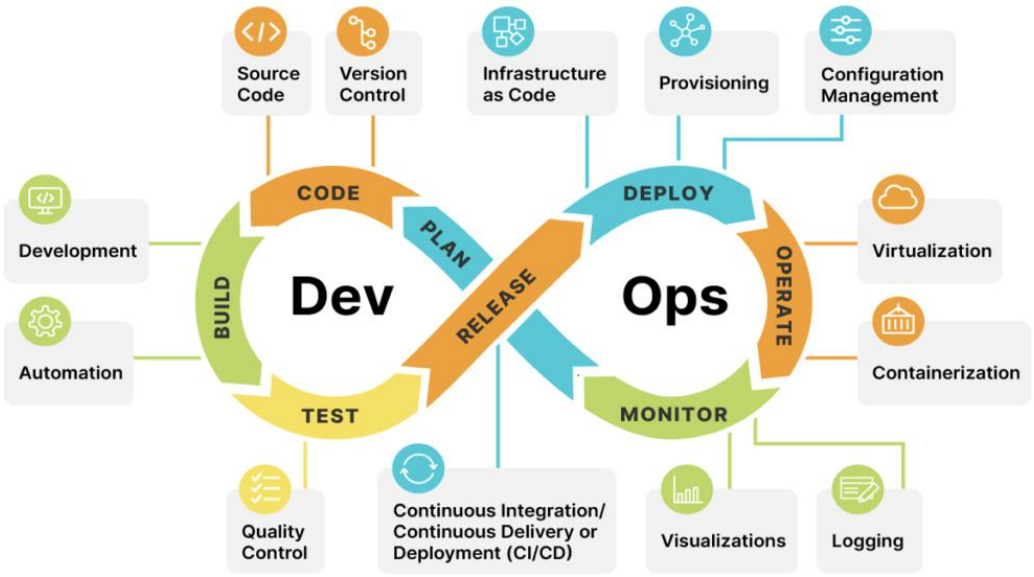


Рис. 1.4. Цикл розробки у методології DevOps.

Методологія DevOps і практики CI/CD значно вплинула на процес розробки програмного забезпечення.

DevOps сприяє укріпленню зв'язку та співпраці між розробниками, тестувальниками, операційними командами та іншими учасниками процесу. Це призводить до зниження взаємних бар’єрів, стимулює спільну відповідальність і культивує культуру взаємодії, що прискорює отримання зворотного зв'язку та забезпечує відповідність продукту вимогам на всіх етапах його створення.

CI/CD, автоматизуючи процеси створення, тестування та розгортання, сприяє зменшенню часу, потрібного для розробки та доставки оновлень ПЗ. Це дозволяє швидше випускати нові функції, ефективно виправляти помилки і більш оперативно реагувати на вимоги ринку.

Завдяки автоматизації та неперервному тестуванню, що використовуються в CI/CD, помилки і проблеми ідентифікуються на ранніх етапах розробки. Це сприяє підвищенню якості коду, надійності ПЗ і систем. Здатність своєчасно виявляти і усувати неполадки знижує ризик важливих збоїв у виробничих середовищах.

Методології DevOps і CI/CD полегшують процес масштабування розробки програмного забезпечення. Завдяки автоматизації рутинних завдань та використанню хмарних технологій команди можуть ефективно розширювати свої здібності у

розробці, тестуванні та розгортанні, адаптуючись до мінливих потреб та зростаючого обсягу роботи.

DevOps і CI/CD підтримують постійний зворотний зв'язок, дозволяючи розробникам отримувати важливі відгуки від користувачів і операційних команд, а також аналізувати показники продуктивності. Цей неперервний процес зворотного зв'язку сприяє ітеративним поліпшенням, дозволяючи командам швидко реагувати на відгуки, ідентифікувати перешкоди, оптимізувати процеси та постійно удосконалювати продукт.

Враховуючи все вищесказане, методики DevOps та CI/CD розроблені для вирішення завдань сучасної розробки ПЗ. Вони сприяють співпраці, автоматизації та неперервному розгортанню, щоб покращити ефективність, швидкість, якість і надійність на всіх етапах життєвого циклу розробки ПЗ. Завдяки цим практикам організації можуть швидше виходити на ринок, мати кращу продуктивність та створювати високоякісні програмні продукти.

1.3. Огляд концепцій безперервної інтеграції, розгортання

У цьому підрозділі ми детально розглядаємо три ключові концепції сучасної розробки програмного забезпечення: безперервну інтеграцію, розгортання та тестування. Ці методології є важливими елементами Agile та DevOps підходів і спрямовані на оптимізацію процесів розробки, забезпечення високої якості продуктів та швидкого реагування на зміни.

Безперервна Інтеграція (Continuous Integration, CI). Полягає у регулярному об'єднанні коду від усіх розробників у спільну базу. Це дозволяє швидше виявляти конфлікти коду, забезпечує їхнє швидке вирішення та підвищує якість продукту. В розділі обговорюється використання автоматизованих засобів для збірки та тестування коду, які сприяють ефективності CI[9].

Безперервне Розгортання (Continuous Deployment, CD). CD включає автоматичне розгортання всіх змін коду в продуктивному середовищі після успішного проходження тестів. Цей підхід забезпечує швидке впровадження нових

функцій та поліпшень, зменшуючи час між написанням коду та його використанням кінцевими користувачами.[9]

Безперервне Тестування (Continuous Testing, CT). CT – це процес постійного тестування продукту на кожному етапі розробки. Воно включає в себе автоматизацію тестів для забезпечення високої якості та оперативного виявлення помилок. У цьому розділі розглядаються різні стратегії та інструменти для ефективного CT.

В кінці розділу надається аналіз переваг та викликів, пов'язаних з впровадженням цих підходів, а також обговорюються кращі практики їх використання у різноманітних проектах розробки ПЗ.

1.3.1. Безперервна інтеграція коду

Безперервна інтеграція (CI) – це метод розробки ПЗ, при якому розробники регулярно додають зміни коду до основного репозиторію. Після цього автоматично здійснюється збірка, тестування та розгортання. CI зазвичай використовується на етапах складання та інтеграції в процесі випуску ПЗ, включаючи елементи автоматизації (як-от служби CI) і культуру розробки (наприклад, практику частої інтеграції). Основна мета CI – швидко виявляти та виправляти помилки, підвищувати якість ПЗ та зменшувати час на тестування і випуск оновлень.

CI спрямована на мінімізацію витрат на інтеграцію, забезпечуючи її на ранніх етапах. Розробники можуть виявляти конфлікти між новим та існуючим кодом раніше, коли їх легше виправляти. Після вирішення конфліктів робота може продовжуватися з упевненістю в сумісності нового коду з існуючою кодовою базою.

Інтеграція коду сама по собі не гарантує якості або функціональності. У багатьох випадках інтеграція вимагає значних ресурсів, оскільки вона включає ручні процеси для переконання, що код відповідає стандартам і не порушує існуючу функціональність. Часта інтеграція може створювати виклики, якщо рівень автоматизації не відповідає потребам контролю якості.

CI долає ці перешкоди, спираючись на автоматизацію та надійні набори тестів. Коли розробник вносить зміни до основного репозиторію, автоматизовані процеси ініціюють збірку та тестування, виявляючи помилки на ранніх етапах. Це робить

інтеграцію простішою та повторюваною частиною повсякденної роботи, зменшуючи витрати на інтеграцію та дозволяючи швидко виявляти дефекти коду. Основа успіху CI полягає в розвитку надійної та швидкої автоматизованої системи, а також у формуванні культури, що сприяє частим ітераціям і швидкому реагуванню на проблеми.

У рамках CI розробники часто вносять зміни у спільний репозиторій, використовуючи системи контролю версій, як Git. Перед внесенням змін у репозиторій, вони можуть виконувати локальні модульні тести для додаткової перевірки. Служба CI автоматично здійснює складання та тестування, що дозволяє вчасно виявляти помилки.

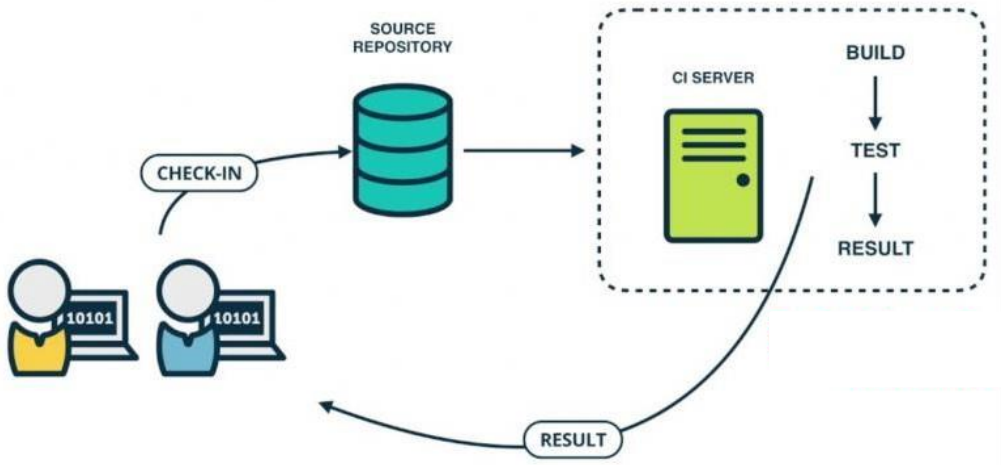


Рис. 2.1. Схема роботи сервісу безперервної інтеграції

Безперервна інтеграція (CI) є ключовою частиною процесу розробки та випуску ПЗ, що охоплює етапи складання та модульного тестування. Кожне підтвердження змін у коді ініціює автоматизований процес збірки та тестування.

Безперервна доставка (CD) є розширенням CI, де зміни коду після збірки автоматично розгортаються у тестовому та/або продуктивному середовищі. CD забезпечує безперервний потік розробки, тестування та випуску оновлень ПЗ.

Застосування CI підвищує ефективність розробки, прискорюючи процеси тестування та розробки, що приносить наступні переваги:

1. Збільшення продуктивності команди. CI звільняє розробників від ручної інтеграції, сприяючи підходам, які зменшують помилки та дефекти у програмному продукті.

2. Швидке виявлення та усунення Помилок. Часте та всебічне тестування дозволяє команді своєчасно виявляти та виправляти помилки, не допускаючи їх переростання у серйозні проблеми.

3. Швидке випускання оновлень. CI дозволяє командам швидше та частіше доставляти оновлення кінцевим користувачам, підтримуючи неперервний розвиток продукту.

Таким чином, безперервна інтеграція є фундаментальним елементом у сучасних практиках розробки ПЗ, забезпечуючи ефективність, якість та швидкість процесу розробки.

1.3.2. Безперервна доставка та розгортання коду

Безперервна доставка (CD), яка включає безперервну інтеграцію та доставку, є важливим аспектом розробки ПЗ. У цьому процесі, будь-які зміни в коді автоматично проходять етапи збірки, тестування, та підготовки до випуску. CD значно розширює можливості безперервної інтеграції, автоматично розгортаючи зміни в коді в тестовому та/або продуктивному середовищі. Це дозволяє розробникам мати версію програмного продукту, готову до розгортання, що пройшла стандартні процедури тестування.

Завдяки безперервній доставці, розробники можуть автоматизувати не тільки модульне тестування, а й проводити комплексне тестування оновлень, включаючи UI тести, навантажувальні тести, інтеграційне тестування, перевірку надійності API та інше. Це дозволяє ретельно перевірити оновлення та виявити потенційні проблеми на ранніх стадіях. Хмарні середовища сприяють легкій та економічно ефективний автоматизації багатократного створення та відтворення тестових середовищ.

В контексті безперервної доставки, кожна зміна коду проходить через збірку, тестування, і потім рухається до підготовчої середи, як ілюструється на рис. 2.2. Різні стадії тестування можуть бути виконані паралельно перед розгортанням у

продуктивному середовищі. Важливою особливістю безперервної доставки є те, що вона вимагає ручного підтвердження для розгортання оновлень у середовищі, на відміну від безперервного розгортання, де процес відбувається автоматично.

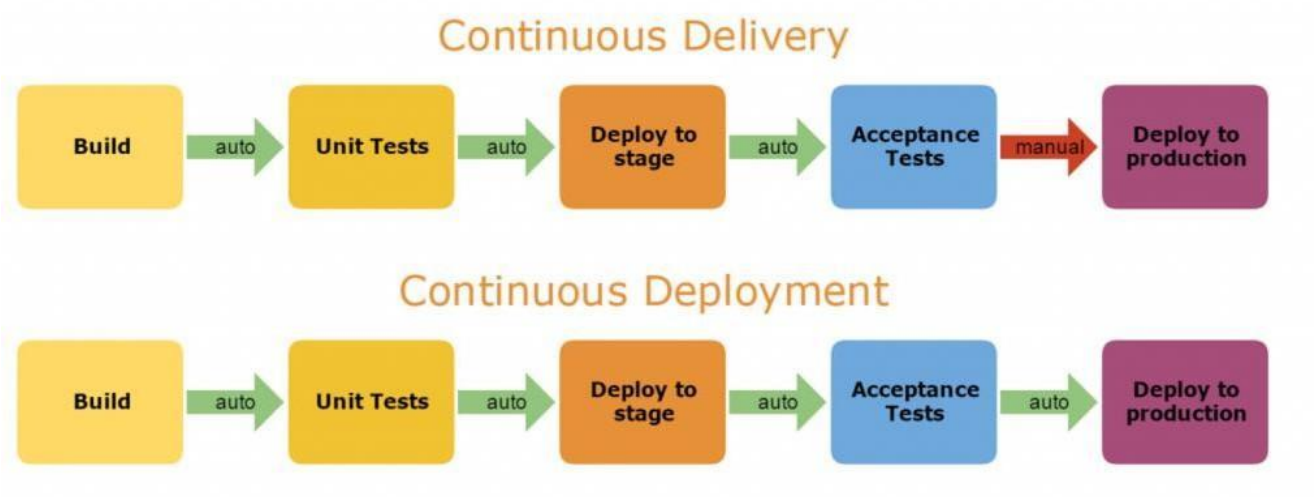


Рис. 2.2. Схематичне порівняння етапів безперервної доставки та безперервного розгортання коду

Ефективність застосування безперервної доставки (CD) можна підтвердити наступними перевагами:

1. Автоматизація релізу ПЗ. CD надає можливість команді автоматично здійснювати збірку, тестування та підготовку змін у коді до випуску в продуктивному середовищі. Це сприяє більш ефективній та швидкій доставці програмного продукту.

2. Швидке виявлення та виправлення помилок. Регулярне та всебічне тестування дозволяє команді своєчасно виявляти та усувати помилки, перешкоджаючи їх перетворенню на серйозні проблеми. Автоматизація процесу спрощує додаткове тестування коду.

3. Підвищення продуктивності розробників. Впровадження CD звільняє розробників від ручної роботи, зменшуючи кількість помилок і дефектів у розгортанні, що сприяє більш ефективній роботі команди.

4. Швидке оновлення ПЗ. Завдяки CD, команда може швидше та частіше випускати оновлення, маючи завжди готовий до розгортання продукт після стандартизованого тестування.

Безперервна доставка забезпечує виконання маленьких частин функціоналу, інтегрованих з основною гілкою розробки кілька разів на день. Процеси збірки та

тестування автоматизовані через безперервну інтеграцію (CI), забезпечуючи швидкі результати. CD автоматизує ці процеси, а безперервне розгортання додатково автоматизує весь розробний процес від написання коду до його випуску у продуктивному середовищі.

1.3.3. Переваги впровадження практик CI/CD у процес розробки ПЗ

Постійна інтеграція та безперервне розгортання пропонують численні переваги, серед яких:

1. Зниження ризиків. Часте розгортання коду знижує ризики в проектах, оскільки помилки легше виявляти та виправляти. Це сприяє швидшому процесу виробництва.

2. Ефективна комунікація. Завдяки налагодженому процесу CI/CD/CT, обмін кодом стає більш ефективним, що покращує взаємодію між членами команди та забезпечує кращу комунікацію.

3. Швидші ітерації. Часте розгортання коду зменшує проміжок між розробленою та вже використовуваною версією коду, спрощуючи слідкування за змінами, внесеними в кожній ітерації.

4. Мінімізація ручної роботи. Автоматизація через CI/CD/CT скорочує необхідність ручної роботи в інтеграції і тестуванні, сприяючи безперервному потоку коду до різних середовищ.

5. Автоматизоване тестування. Автоматизація знижує час, необхідний для тестування коду, і зменшує ризик людських помилок у тестуванні.

6. Швидкий зворотній зв'язок. Налаштування CI/CD не тільки полегшує роботу розробників, але й дозволяє керівництву швидше приймати рішення та адаптуватися до виробничих змін.

Тепер, коли ми розглянули відмінності та переваги безперервної інтеграції, безперервного розгортання та доставки, перейдемо до іншого ключового терміну – pipeline, або конвеєр. Рис. 2.4 допоможе краще зрозуміти цей термін.

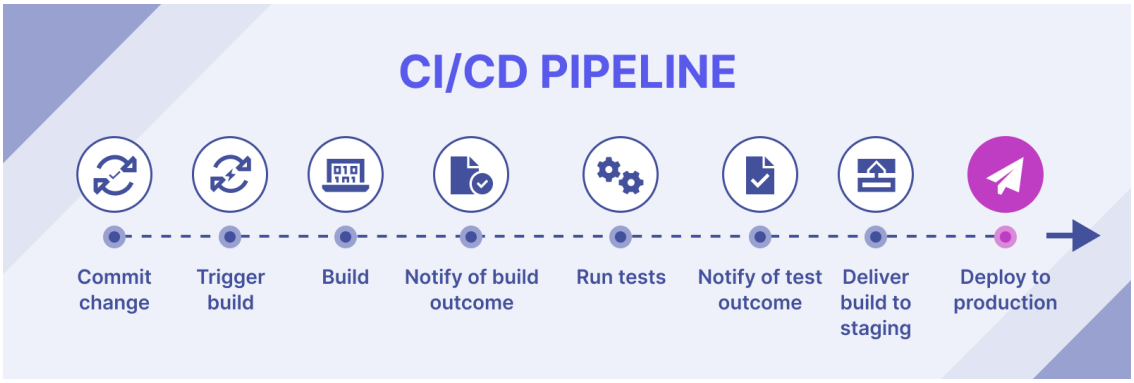


Рис. 2.4. Загальна схема CI/CD конвеєру

Pipeline, або конвеєр – серія послідовних автоматизованих дій, що постійно виконуються при внесенні змін у проект. Цей процес є ключовою частиною практик безперервної інтеграції та безперервної доставки/розгортання. Конвеєр складається з двох вище описаних практик[21]:

1. Безперервна інтеграція (CI). На цій стадії, розробники регулярно зливають зміни коду в основний репозиторій, де вони автоматично піддаються збірці та тестуванню. Це дозволяє швидко виявляти та усувати помилки.

2. Безперервна доставка/розгортання. На цій стадії, зміни, що пройшли через CI, автоматично розгортаються у тестове та/або продуктивне середовище. У випадку безперервної доставки, ці зміни готуються для випуску і можуть потребувати ручного підтвердження для розгортання. У випадку безперервного розгортання, розгортання відбувається автоматично.

CI/CD конвеєр об'єднує ці процеси в єдину автоматизовану і ефективну систему, що забезпечує неперервний потік розробки, тестування, доставки та розгортання програмного забезпечення, значно підвищуючи швидкість розробки та якість кінцевого продукту.

1.4. Недоліки впровадження практик CI/CD у процес розробки ПЗ

Авжеж, як і будь-яка інша складна та багатокомпонентна система, впровадження автоматизованого CI/CD конвеєру несе за собою і певні ризики та має недоліки. Опишемо їх у даному підрозділі[21][19][1].

Отже, основний перелік недоліків, які можуть стати перепоною у впровадженні конвеєру в проект:

1. Високі вимоги до початкового налаштування та підтримки. Імплементація та підтримка CI/CD може бути складною та вимагати значних ресурсів, особливо в більших організаціях з великою кількістю проектів.

2. Потреба в постійному оновленні та оптимізації процесів. Для ефективної роботи CI/CD потрібно постійно оновлювати та оптимізувати процеси, що може вимагати додаткових зусиль та ресурсів.

3. Комплексність управління змінами. Швидке впровадження змін може призвести до проблем з управлінням змінами, особливо якщо команда не має чіткої стратегії чи не дотримується кращих практик.

4. Ризик недостатнього тестування. Хоча автоматизація тестування є ключовою складовою CI/CD, існує ризик, що деякі помилки можуть не бути виявлені, особливо у складних системах.

5. Безпека та надійність. Швидке розгортання може призвести до недостатньої уваги до питань безпеки та надійності, особливо якщо не використовуються відповідні інструменти та практики.

6. Залежність від інструментів та технологій. CI/CD значною мірою залежить від інструментів та технологій, що може обмежувати гнучкість та спричинити додаткові витрати на ліцензування та підтримку.

7. Опір змінам серед членів команди. Впровадження CI/CD може зустріти опір з боку команд та індивідів, які звикли до традиційних методів розробки.

8. Необхідність кваліфікованих членів команди. Ефективне використання CI/CD вимагає наявності в команді кваліфікованих спеціалістів, які розуміються на найкращих практиках і можуть налаштувати та підтримувати складні системи.

1.5. Постановка задачі кваліфікаційної роботи

Метою цієї кваліфікаційної роботи є розробка базового CI/CD конвеєра, призначеного для колаборативної розробки додатків. Такий підхід має на меті спростити процес розробки програмного забезпечення, забезпечуючи автоматизацію та оптимізацію повсякденних завдань і, як наслідок, підвищення продуктивності команди розробників.

Ключові елементи системи включають впровадження методологій та практик DevOps та CI/CD у процес розробки ПЗ. Це охоплює:

- застосування систем контролю версій на стадії розробки.

- автоматизацію повторюваних задач на різних етапах розробки, включаючи імплементацію циклу CI/CD, що охоплює збірку, тестування та розгортання.

- автоматичний статичний аналіз коду як інтегральну частину процесу забезпечення якості (QA) для підвищення надійності коду та поліпшення зворотного зв'язку.

- відкритість коду і безкоштовність інструментів, що будуть використовуватись, сприяє кращому засвоєнню сучасних методик розробки ПЗ та розвитку важливих навичок в IT-сфері.

Висновки до розділу 1

Цей розділ кваліфікаційної роботи охоплює широкий спектр методологій та практик, які використовуються в сучасній розробці програмного забезпечення. Аналіз життєвого циклу програмного забезпечення, включаючи класичні методології, такі як водоспадна модель та V-Model, а також більш гнучкі підходи, як Agile і DevOps, демонструє різноманітність доступних стратегій для розробки ПЗ.

Особливу увагу у роботі приділено концепціям безперервної інтеграції (CI) та безперервного розгортання (CD), які стають все більш актуальними у сучасному програмуванні. Впровадження CI/CD в процес розробки ПЗ дозволяє значно прискорити процес розробки, підвищити якість продукту та забезпечити більшу гнучкість у реагуванні на зміни вимог чи умов ринку.

Аналіз практик CI/CD у роботі виявляє їхні численні переваги, включаючи зменшення помилок, поліпшення комунікації в команді, швидкість доставки оновлень та підвищення загальної продуктивності розробки. Водночас, у роботі також розглянуті потенційні недоліки впровадження CI/CD, зокрема складності налаштування та потреба у високій кваліфікації персоналу для ефективного управління цими процесами.

Розділ завершується постановкою задачі кваліфікаційної роботи, яка має на меті глибше дослідити та аналізувати певні аспекти застосування методологій та практик CI/CD у реальних проектах розробки ПЗ, з метою виявлення оптимальних підходів та стратегій для їх застосування.

РОЗДІЛ 2. АНАЛІЗ ДОСТУПНИХ НА РИНКУ ІНСТРУМЕНТІВ ПОБУДОВИ CI/CD КОНВЕЄРУ

2.1. Визначення критеріїв аналізу

Застосування Agile та DevOps у комерційній розробці ПЗ дозволило значно скоротити часові рамки програмування, переводячи їх на тижневі цикли, та створити регулярний графік доставки. Сучасний підхід безперервної інтеграції (CI) вносить оновлення у процес розробки навіть швидше - в межах днів чи навіть годин. Це досягається через постійну відправку коду до спільного репозиторію, що дозволяє розробникам оперативно виявляти та виправляти помилки за допомогою автоматизованих тестів. У цьому розділі ми глибоко розглянемо переваги та можливі недоліки використання інструментів CI/CD/CT в процесі розробки, розглянемо критерії для вибору відповідних інструментів та обговоримо потенційні виклики, які можуть виникнути.

Перш за все, важливо зазначити, що розробники інструментів CI/CD прагнуть створити рішення, яке дозволяє інтегрувати процеси безперервної інтеграції, доставки, розгортання та тестування в одній системі. Таким чином, усі розглянуті інструменти у даному розділі можна розглядати як комплексні рішення для налаштування конвеєру.

На основі досвіду використання різних інструментів та обширного вивчення теоретичних матеріалів було визначено наступні критерії для порівняльного аналізу:

- модель поширення – за якою моделлю розробник поширює своє ПЗ;
- чи відкритий вихідний код;
- формат розгортання: важливо розглядати, як інструменти управляють інфраструктурою. Хмарні інструменти, які розміщуються на стороні постачальника, вимагають мінімальної конфігурації та легко адаптуються під потреби проекту. Self-hosted рішення вимагають додаткових навичок в налаштуванні і розгортанні системи, але забезпечують більшу гнучкість;
- зручність використання: оцінка інтуїтивності та зрозумілості інтерфейсу користувача, який може спрощувати процес збірки;

- інтеграція та підтримка ПЗ: важливість здатності інструменту CI інтегруватися з зовнішніми сервісами та підтримувати різноманітні інструменти та програмне забезпечення;

- підтримка технології контейнеризації: наявність функцій для інтеграції з системами контейнеризації додатків, як-от Kubernetes і Docker;

- бібліотека коду багаторазового використання: наявність загальнодоступного сховища плагінів та можливість доповнювати власними рішеннями.

Для аналізу було обрано кілька популярних інструментів, таких як Jenkins, TeamCity, Bamboo, Travis CI, Circle CI, GitLab CI, GoCD, які будуть докладно проаналізовані та порівняні на основі вказаних критеріїв.

2.2. Порівняння інструментів

2.2.1. Jenkins

Jenkins – це відкрите програмне забезпечення для автоматизації процесів у розробці програмного забезпечення, зокрема для безперервної інтеграції та доставки (CI/CD). ПЗ розроблено Косуге Кавагуті, вихідний код поширюється за ліцензією MIT[17].

Проаналізуємо Jenkins за вказаними у підрозділі 2.2 критеріями:

1. Модель поширення: безкоштовно.

2. Вихідний код відкритий за ліцензією MIT.

3. Формат розгортання. Jenkins є гнучким в аспекті розгортання. Він може бути встановлений як у локальному середовищі, так і в хмарі. Це надає користувачам свободу вибору щодо управління інфраструктурою. Хоча Jenkins не є складним у налаштуванні для базового функціонування, він може потребувати додаткових зусиль для оптимізації та налаштування в складніших середовищах.

4. Зручність використання. Jenkins має велику спільноту, але недостатньо детальну документацію, що робить його в цілому доступним для нових користувачів. Його інтерфейс, хоч і не найсучасніший, є досить функціональним та зрозумілим для більшості розробників.

5. Інтеграція та підтримка ПЗ: Jenkins підтримує широкий спектр плагінів для інтеграції з різними зовнішніми сервісами та інструментами, включаючи системи управління проектами, інструменти відстеження дефектів, статичного аналізу тощо. Це робить його дуже гнучким у використанні в різних проектах.

6. Підтримка технології контейнеризації. Jenkins чудово працює з технологіями контейнеризації, зокрема через плагіни для Docker та Kubernetes. Це дозволяє легко інтегрувати Jenkins у сучасні середовища розгортання.

7. Бібліотека коду багаторазового використання. однією з ключових переваг Jenkins є його обширна бібліотека плагінів. Відкритий вихідний код та активна спільнота забезпечують постійне оновлення та розширення функціональності.

Jenkins, будучи одним з найпопулярніших інструментів для автоматизації CI/CD, має декілька поширених недоліків:

1. Складність налаштування. Jenkins може бути складним у налаштуванні та вимагає певного рівня технічної експертизи, особливо при інтеграції з різними інструментами та плагінами.

2. Велика залежність від плагінів. Багато функцій Jenkins залежать від плагінів, що може призвести до проблем з сумісністю, стабільністю та безпекою.

3. Проблеми з продуктивністю. У деяких випадках, особливо в великих проектах з великою кількістю завдань, Jenkins може стикатися з проблемами продуктивності та масштабування.

4. Потреба в регулярному оновленні. Jenkins потребує регулярних оновлень, як і його плагіни, що може створювати додаткову роботу для команди DevOps.

5. Інтерфейс користувача. Інтерфейс Jenkins може здатися застарілим і менш інтуїтивним у порівнянні з сучаснішими інструментами CI/CD.

6. Проблеми з безпекою. Через свою відкриту архітектуру та велику кількість плагінів, Jenkins може мати потенційні проблеми з безпекою.

7. Обмежені можливості інтеграції “з коробки”. Деякі бажані функції інтеграції можуть вимагати додаткових плагінів або налаштувань.

Ці недоліки не обов'язково зроблять Jenkins непридатним для ваших потреб, але вони важливі для розгляду при оцінці використання Jenkins у вашому проекті або організації.

В цілому, Jenkins є потужним інструментом з гнучкими можливостями хостингу, широкою підтримкою інтеграції та значною зручністю використання. Хоча він може вимагати додаткових зусиль для налаштування в складних середовищах, його функціональність та адаптивність роблять його відмінним вибором для багатьох проектів розробки ПЗ.

2.2.2. GitLab

GitLab, відомий інструмент для управління розробкою програмного забезпечення, наведу аналіз за вказаними критеріями:

1. Модель поширення та відкритий вихідний код. GitLab доступний як у відкритому вихідному коді (GitLab Community Edition), так і у комерційній версії (GitLab Enterprise Edition). Відкритий вихідний код дозволяє користувачам налаштовувати інструмент під свої конкретні потреби.

2. Формат розгортання. GitLab пропонує гнучкість у виборі формату розгортання. Його можна встановити локально на сервері або використовувати як хмарний сервіс. Це забезпечує широкі можливості для різних типів організацій і інфраструктур.

3. Зручність використання. GitLab відомий своїм інтуїтивно зрозумілим інтерфейсом та високою зручністю використання. Це робить його доступним не тільки для досвідчених розробників, але і для новачків.

4. Інтеграція та підтримка. GitLab має широкі можливості інтеграції з іншими інструментами та системами, що включають системи управління проектами, інструменти моніторингу, та інші CI/CD інструменти. Він також підтримує широкий спектр плагінів.

5. Підтримка технології контейнеризації. GitLab активно підтримує контейнеризацію, зокрема через інтеграцію з Docker та Kubernetes. Це дозволяє легко використовувати GitLab у середовищах, заснованих на контейнерах.

6. Бібліотека коду багаторазового використання. GitLab має велику бібліотеку готових до використання плагінів і інструментів, що сприяє повторному використанню коду та підвищує ефективність робочих процесів[16].

GitLab, хоча і є потужним інструментом для управління CI/CD та розробки ПЗ, має ряд поширених недоліків:

1. Складність управління для великих проектів. GitLab може бути складним у налаштуванні та управлінні, особливо для великих або складних проектів, де потрібно врахувати багато аспектів роботи.

2. Високі вимоги до ресурсів. GitLab може бути ресурсоємним, особливо при роботі на серверах з обмеженими ресурсами, що може призвести до зниження продуктивності.

3. Залежність від хмарних рішень. Хоча GitLab пропонує хмарні та локальні варіанти, його хмарна версія часто має переваги з точки зору функціональності, що може бути обмежувальним фактором для організацій з обмеженнями на використання хмарних технологій.

4. Комплексність інтерфейсу. Деякі користувачі вважають інтерфейс GitLab перевантаженим та складним для навігації, особливо для новачків.

5. Обмеження безкоштовної версії. Безкоштовна версія GitLab має деякі обмеження у функціональності порівняно з платною версією, що може обмежувати її використання для більш складних проектів.

6. Проблеми з масштабуванням. Деякі користувачі відзначають проблеми з масштабуванням GitLab, особливо в контексті великих команд та проектів.

7. Проблеми з інтеграцією. Хоча GitLab підтримує багато інтеграцій, деякі користувачі можуть зіткнутися з викликами при інтеграції з певними зовнішніми інструментами або системами.

Загалом, GitLab є потужним інструментом для управління розробкою ПЗ, який пропонує гнучкість, зручність використання, інтеграцію з різними технологіями та підтримку сучасних практик розробки, включаючи контейнеризацію та автоматизацію CI/CD.

2.2.3. Travis CI

Аналіз Travis CI за вказаними критеріями:

1. Модель поширення. Travis CI є доступним як у відкритому вихідному коді, так і у формі комерційного хостингового рішення. Основна версія Travis CI, використовувана для відкритих проектів, є безкоштовною.

2. Формат розгортання. Travis CI первинно працює як хмарний сервіс. Він інтегрований з GitHub, автоматично виявляючи нові коміти та pull запити.

3. Зручність використання. Travis CI відомий своїм простим у використанні інтерфейсом та легкістю налаштування, забезпечуючи простий процес інтеграції з GitHub.

4. Інтеграція та підтримка. Travis CI підтримує широкий спектр мов програмування та інтеграцію з різними інструментами. Він ідеально підходить для проектів, розміщених на GitHub.

5. Підтримка технології контейнеризації. Travis CI дозволяє використовувати Docker, що робить його зручним для проектів, які використовують контейнеризацію.

6. Бібліотека коду багаторазового використання. Хоча Travis CI не надає великої бібліотеки плагінів на зразок Jenkins, його інтеграція з GitHub та підтримка Docker забезпечують певний рівень гнучкості для багаторазового використання коду[18].

Travis CI, відомий як популярний інструмент для автоматизації CI/CD, має декілька поширених недоліків:

1. Обмежені можливості безкоштовного плану. Для відкритих проектів Travis CI пропонує обмежену кількість ресурсів у безкоштовному плані, що може бути недостатньо для великих проектів або активних команд розробників.

2. Залежність від GitHub. Travis CI сильно інтегрований з GitHub, що робить його менш гнучким для користувачів, які використовують інші системи контролю версій.

3. Обмежена підтримка мов програмування та платформ. На відміну від деяких інших інструментів CI/CD, Travis CI може не підтримувати деякі мови програмування або платформи, особливо менш поширені.

4. Проблеми з масштабуванням. Для великих проектів з великою кількістю одночасних завдань Travis CI може зіткнутися з проблемами продуктивності.

5. Інтерфейс користувача. Деякі користувачі вважають, що інтерфейс Travis CI не є найзручнішим або може бути складним для освоєння новачками.

6. Обмежені можливості інтеграції. На відміну від більш гнучких інструментів CI/CD, Travis CI може не пропонувати такий широкий діапазон інтеграцій або налаштувань.

7. Проблеми з конфіденційністю. В деяких випадках, користувачі висловлювали занепокоєння щодо конфіденційності та безпеки даних при використанні Travis CI, особливо в контексті приватних репозиторіїв.

Ці недоліки слід враховувати при оцінці використання Travis CI для конкретних потреб проекту або організації.

Загалом, Travis CI є зручним і ефективним інструментом для CI, особливо підходить для менших проектів та тих, що розміщені на GitHub. Його легкість у налаштуванні та інтеграції роблять його популярним вибором серед розробників.

2.2.4. Go CD

Аналіз GoCD за вказаними критеріями:

1. Модель поширення. GoCD пропонується як відкрите програмне забезпечення, що дозволяє користувачам використовувати його безкоштовно і адаптувати під свої потреби.

2. Відкритий вихідний код. GoCD є продуктом з відкритим вихідним кодом, що забезпечує високий рівень гнучкості та можливості для налаштування.

3. Формат розгортання. GoCD може бути розгорнутий як у локальному середовищі, так і в хмарі, забезпечуючи різні варіанти використання.

4. Зручність використання. Інтерфейс GoCD орієнтований на забезпечення зручності користувача, хоча може вимагати певного часу на освоєння для нових користувачів.

5. Інтеграція та підтримка. GoCD підтримує інтеграцію з широким спектром інструментів і технологій, що робить його ефективним рішенням для складних CI/CD процесів.

6. Підтримка технології контейнеризації. GoCD підтримує сучасні технології контейнеризації, зокрема Docker, що дозволяє легко інтегрувати його у контейнеризовані середовища.

7. Бібліотека коду багаторазового використання. Хоча GoCD не має такої великої бібліотеки плагінів, як деякі інші інструменти, його відкритий вихідний код та можливості налаштування забезпечують певний рівень гнучкості у повторному використанні коду[19].

GoCD, популярний інструмент для управління безперервною інтеграцією та доставкою, має деякі недоліки, які важливо врахувати:

1. Комплексність конфігурації. GoCD може бути складним для налаштування, особливо для користувачів без глибоких знань у сфері CI/CD.

2. Обмежені можливості інтеграції. Хоча GoCD підтримує ряд інтеграцій, він може не пропонувати такий широкий спектр інтеграційних можливостей, як деякі інші інструменти CI/CD.

3. Інтерфейс користувача. Інтерфейс GoCD може бути не таким інтуїтивно зрозумілим або зручним, як у деяких інших платформ.

4. Вимоги до ресурсів. GoCD може бути вимогливим до системних ресурсів, особливо при роботі з великими або складними проектами.

5. Відсутність широкої спільноти. На відміну від більш популярних інструментів, таких як Jenkins, GoCD має менш активну спільноту користувачів, що може обмежувати доступність підтримки та ресурсів.

6. Обмежена документація. В порівнянні з іншими інструментами CI/CD, документація GoCD може бути менш повною або оновленою.

7. Проблеми з масштабуванням. Для дуже великих проектів або проектів з високим рівнем складності можуть виникнути виклики, пов'язані з масштабуванням GoCD.

Ці недоліки варто врахувати при виборі GoCD як інструменту для управління CI/CD в рамках вашого проекту або організації.

GoCD є сильним гравцем на ринку CI/CD інструментів, особливо цінним для тих, хто шукає відкрите та гнучке рішення з можливістю налаштування під власні потреби.

Висновки до розділу 2

В даному розділі було проведено порівняльний аналіз найбільш популярних доступних на ринку станом на 2023 рік інструментів для побудови CI/CD конвеєру. На порівняння було винесено 4 інструменти: Jenkins, GitLab, Travis CI та Go CD.

Можна зробити висновок, що всі 4 інструменти є приблизно схожими за функціональністю, можливостями інтеграції з стороннім ПЗ та сервісами, гнучкістю налаштування, наявністю плагінів тощо. Тому останнє слово у виборі найбільш підходящого інструменту буде за командою проекту, яка займається розробкою ПЗ. В такому випадку, до перелічених у підрозділі 2.2 критеріїв можуть бути додані такі критерії як, наприклад, наявність на ринку DevOps інженерів, які володіють практичними навичками роботи з бажаним інструментом, їх досвід, бюджетні побажання на компенсацію оплати праці таким інженерам, стек, на якому розробляється проект та його сумісність з конкретним інструментом, безпекова складова тощо. Отже, вибір інструменту для створення конвеєру буде залежати від дуже великої кількості чинників, які визначає команда розробників. Індивідуальний підхід та гнучкість, що є ключовою складовою Agile та DevOps методологій, проявляється в тому числі й у випадку вибору таких інструментів.

В даній кваліфікаційній роботі мною для побудови навчального проекту CI/CD конвеєру був обраний Jenkins, оскільки він повністю задовольняє вимогам навчального проекту, має спільноту підтримки та обширну кількість інформації у мережі Інтернет, а також велику кількість плагінів та інтеграцій зі сторонніми сервісами та технологіями.

РОЗДІЛ 3. ПРАКТИЧНА ІМПЛЕМЕНТАЦІЯ CI/CD КОНВЕЄРУ НА ОСНОВІ ОБРАНОГО ІНСТРУМЕНТУ

У даному розділі буде викладена змодельована наближена до умов реального комерційного проекту задача на побудову CI/CD конвеєру Також у розділі буде покроково викладений і задокументований кожен етап побудови та налаштування такого конвеєру.

3.1. Моделювання задачі побудови CI/CD конвеєру на проекті

Змоделюємо задачу побудови CI/CD конвеєру. Припустимо, що команда проекту, яка складається з проєктного менеджера, п’яти розробників, двох тестувальників та одного DevOps-інженера, отримала замовлення від клієнта на розробку певної системи, яка буде вирішувати його бізнес-задачі. Однією з вимог клієнта є оперативне виведення нових прототипів та релізних версій ПЗ швидко у поставлені терміни відповідно зі складністю завдань, які повинен виконувати продукт. Отже, комунікація та зворотній зв’язок з командою розробників є надзвичайно важливою компонентою при взаємодії обох сторін. Додаток планується кодувати на мові програмування Java. Бюджет на розробку додатку складає $50 000 включно витратами на хмарну інфраструктуру, терміни встановлені замовником – 4 місяці. Кожні два тижні замовник бажає бачити нову версію додатку з реалізованими на етапі планування функціями за домовленістю. Код повинен бути покритий автоматизованими тестами з мінімум помилок і непрацюючих функцій. Деталі, інструменти та інші нюанси побудови процесу розробки додатку команда має змогу вирішити самостійно.

3.2. Аналіз задачі і прийняття рішення щодо шляхів її розв’язання

Оскільки у команді є досвідчений DevOps-інженер, який має навички і досвід роботи на комерційних проектах з різними CI/CD інструментами, команда приступає до аналізу задач, поставлених замовником, а також до планування та визначення інструментів, необхідних для побудови злагодженого та оперативного виконання поставлених завдань. На етапі планування команда узгоджує план імплементації

вирішення бізнес-задач замовника на кожні два тижні. За результатами переговорів проектний менеджер складає план робіт для розробників, тестувальників та DevOps-інженера. Далі до справи залучається DevOps інженер, на цьому етапі він визначає інструменти, які необхідні для побудови інфраструктури, яку будуть використовувати розробники та тестувальники. Оскільки додаток планується кодувати на мові програмування Jenkins, а також через обмежені терміни та бюджет замовника, інженером було вирішено що найбільш оптимальним інструментом для побудови CI/CD конвеєру буде Jenkins. Це пов’язано з тим, що даний інструмент має добру сумісність з оточенням розробки OpenJDK, за допомогою якого розробники і будуть розробляти продукт. Також перевагою Jenkins є гнучкість налаштувань, багата бібліотека плагінів та безкоштовна модель поширення інструменту. Недоліки Jenkins, такі як недостатня документованість системи та плагінів для неї, а також необхідність проведення попередньої роботи по розгортанню та налаштуванню інструменту перекриваються високим рівнем експертизи та досвіду DevOps-інженера у роботі з цим інструментом.

У наступних підрозділах буде детально викладена послідовність налаштування CI/CD конвеєру

3.3. Розгортання Jenkins на робоче оточення

Щоб розпочати роботу з Jenkins, його слід спочатку встановити на віртуальні машини, що працюють під керуванням UNIX-подібних операційних систем. Важливим кроком є визначення типу використовуваної операційної системи (ОС). Для цього потрібно створити на будь-якому хмарному оточенні новую віртуальну машину. Наприклад, Digital Ocean. На рис. 3.1 – 3.5 показаний процес створення віртуальної машини з наступними налаштуваннями:

- регіон: будь-який зручний;
- ОС: Ubuntu 22.04 (LTS) x64
- CPU Options: в рамках навчального проекту вистачить мінімальної конфігурації 1 ГБ оперативної пам’яті, 25 ГБ постійної пам’яті для постійного зберігання даних;

- метод аутентифікації: бажано для аутентифікації використовувати виключно SSH-доступ за допомогою пари публічний-секретний ключі. Це надійно та безпечно, оскільки сервер буде захищений від несанкціонованого проникнення зловмисниками через SSH протокол.

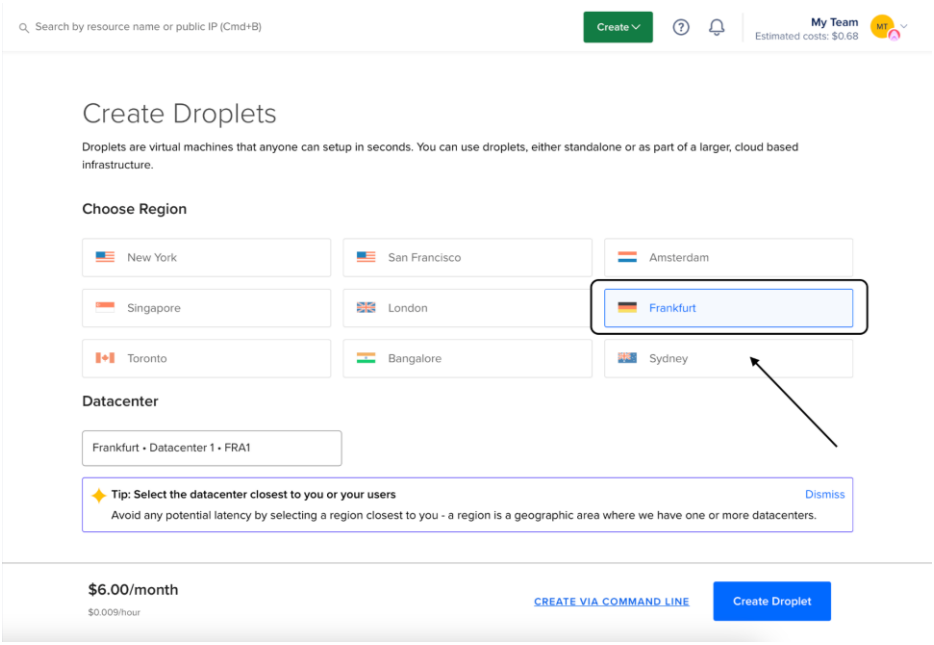


Рис. 3.1 Покрокове створення віртуальної машини для розгортання Jenkins

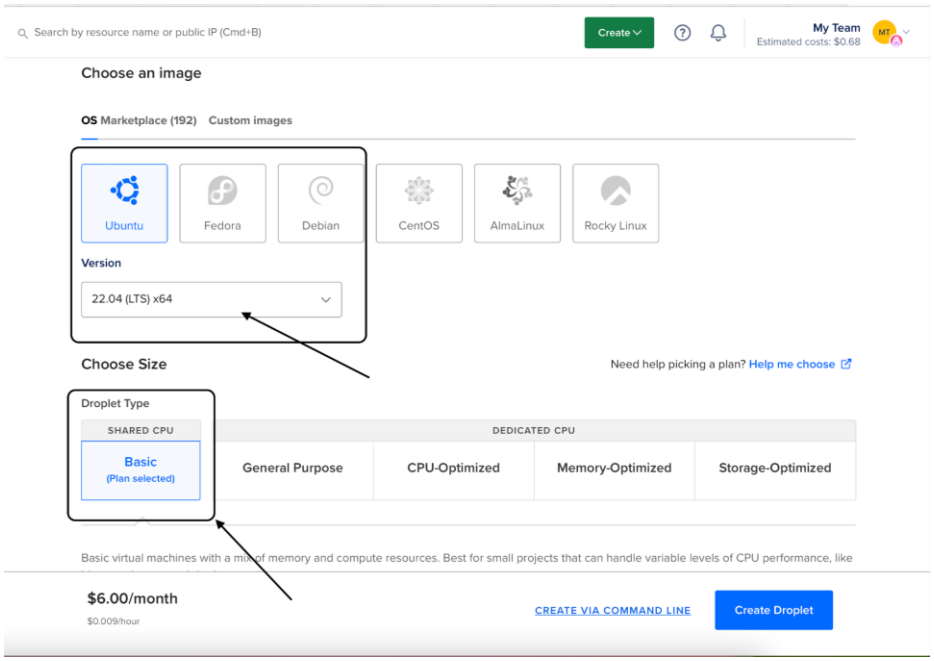


Рис. 3.2 Покрокове створення віртуальної машини для розгортання Jenkins

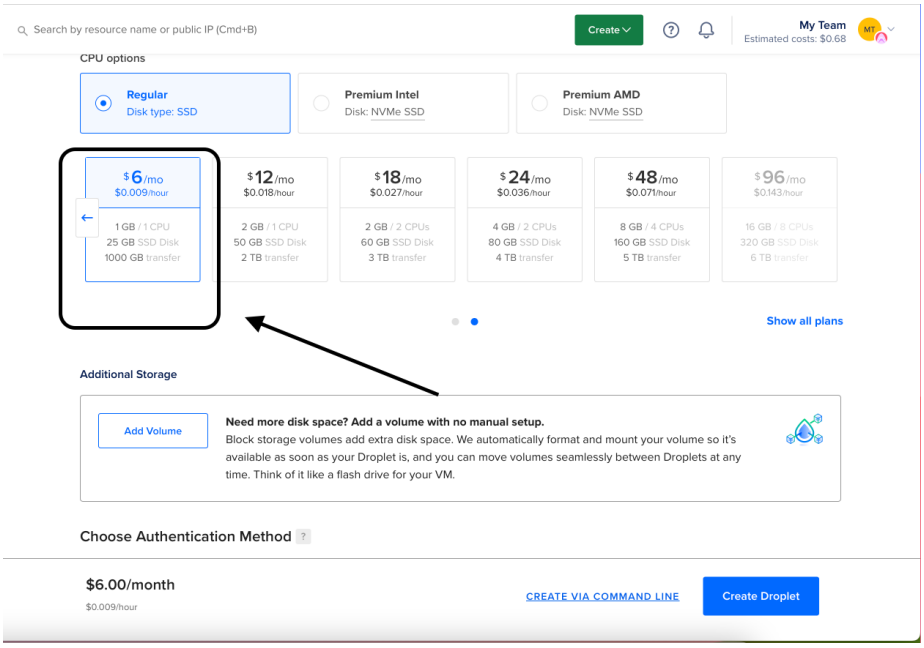


Рис. 3.3 Покрокове створення віртуальної машини для розгортання Jenkins

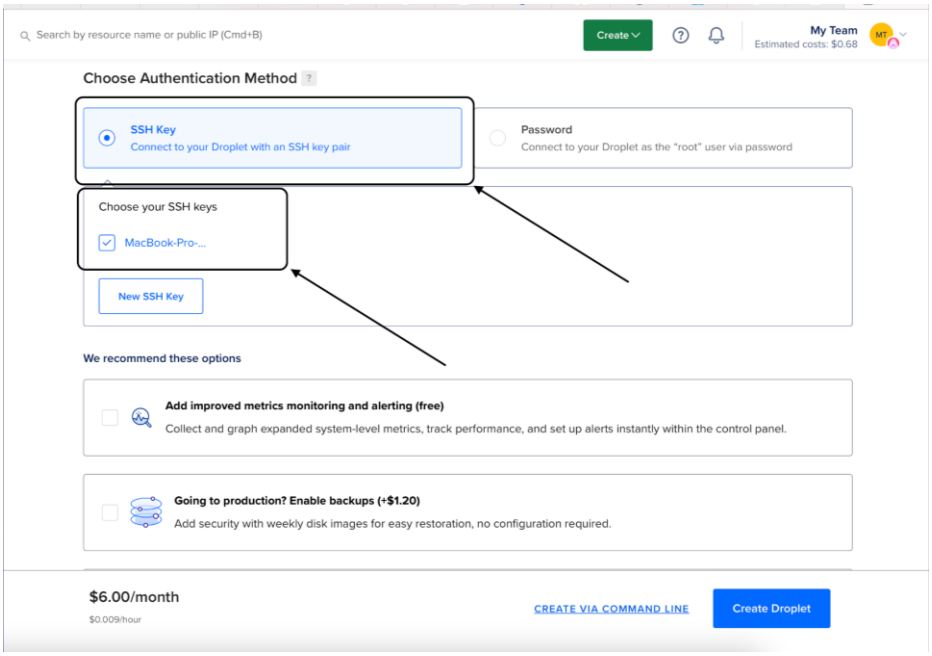


Рис. 3.4 Покрокове створення віртуальної машини для розгортання Jenkins

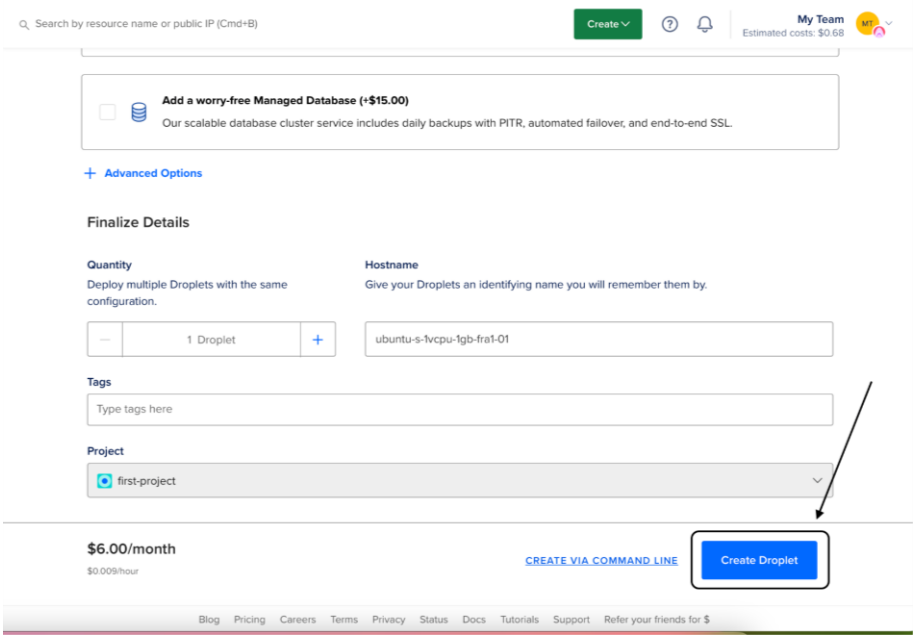


Рис. 3.5 Покрокове створення віртуальної машини для розгортання Jenkins

Після того як віртуальна машина була створена, необхідно підключитися до неї за допомогою SSH-протоколу.

На клієнтських операційних системах Linux та MacOS це можна зробити в терміналі за допомогою команди **`ssh`**.

На клієнтській операційній системі Windows підключитися до віддаленого серверу можно за допомогою клієнту PuTTY, який встановлюється як додаткове прикладне ПЗ, а також у терміналі Windows PowerShell.

Приклад команди для підключення до віддаленого серверу для Linux/MacOS:

```
~ ssh username@xxx.xxx.xxx.xxx -i ~/.ssh/privatekey
```

де:

- `ssh` – команда для роботи з віддаленим доступом до серверу;

- `username@xxx.xxx.xxx.xxx` – ім’я користувача та IP-адреса серверу;

`--i ~/.ssh/privatekey` – шлях до файлу приватного ключа, який потрібно заздалегідь згенерувати на клієнтській ОС. Приватний ключ зберігається виключно на вашому клієнтському пристрої і ділитися ним ні з ким не треба. При генеруванні приватного ключа генерується також його публічна частина. Публічний ключ потрібно вказати при створенні або налаштуванні нового сервера у хмарному провайдері.

Під час першого підключення до серверу проведемо первинні налаштуванн, а саме: оновимо список репозиторії до актуального стану і зробимо оновлення системи та всіх встановлених пакунків у ній. Для Ubuntu/Debian це будуть наступні команди:

```
~ sudo apt update && sudo apt upgrade
```

Далі звернемося до офіційної документації про встановлення Jenkins на робочу машину[11]. На віддаленому сервері виконаємо наступну послідовність команд:

```
sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update
sudo apt-get install jenkins
```

Якщо встановлення пройшло успішно, можна спробувати підключитися до серверу за його адресою у мережі Інтернет. За замовчуванням, Jenkins налаштовується на сервері на прослуховування порту 8080. Якщо щось пішло не так, на сторінці документації буде актуальна інформація, яка допоможе вирішити проблеми. Отже, відкриємо будь-який інтернет-браузер і перейдемо за адресою http://xxx.xxx.xxx.xxx:8080, де xxx.xxx.xxx.xxx – IP-адреса вашого серверу, на який виконувалось розгортання інструменту.

У вкладці повинне з’явитися вітальне вікно Getting Started, яке запитає у вас одноразовий пароль розблокування (рис. 3.6). Цей пароль генерується при першому запуску.

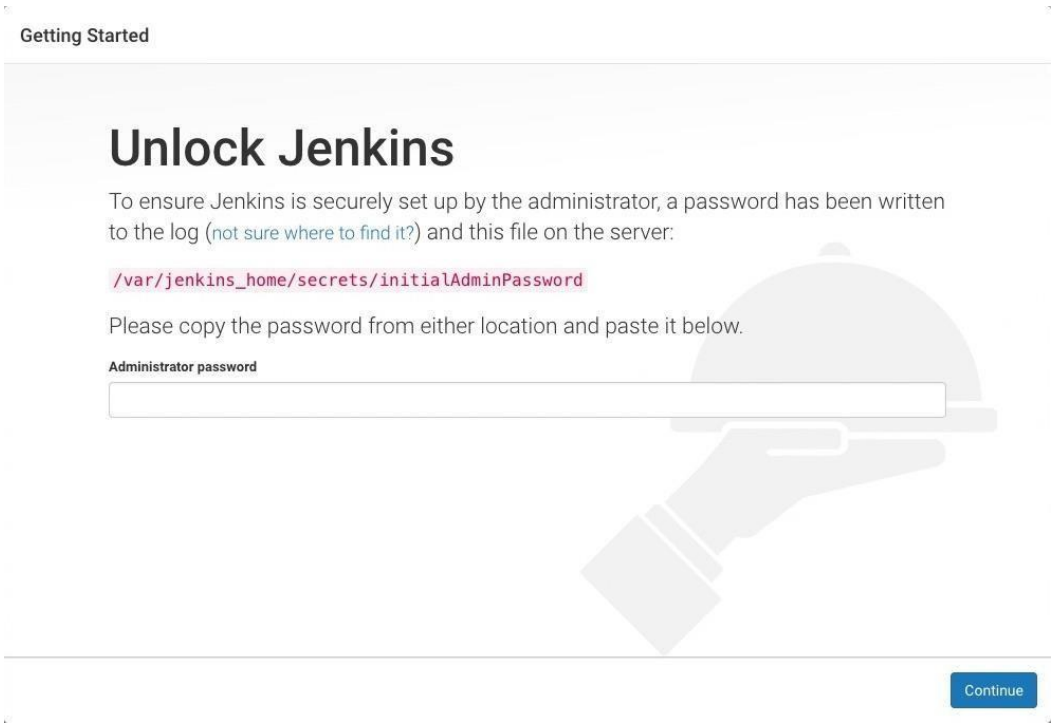


Рис. 3.6. Вітальне вікно з вимогою ввести пароль адміністратора

Для продовження налаштування необхідно знайти пароль адміністратора на сервері, де встановлено Jenkins. Цей пароль розташовано у файлі за визначеним шляхом:

```
/var/Jenkins-home/secrets/initialAdminPassword
```

Або ж його можна побачити у журналі під часу його першого запуску, як це показано на рис. 3.7. нижче.

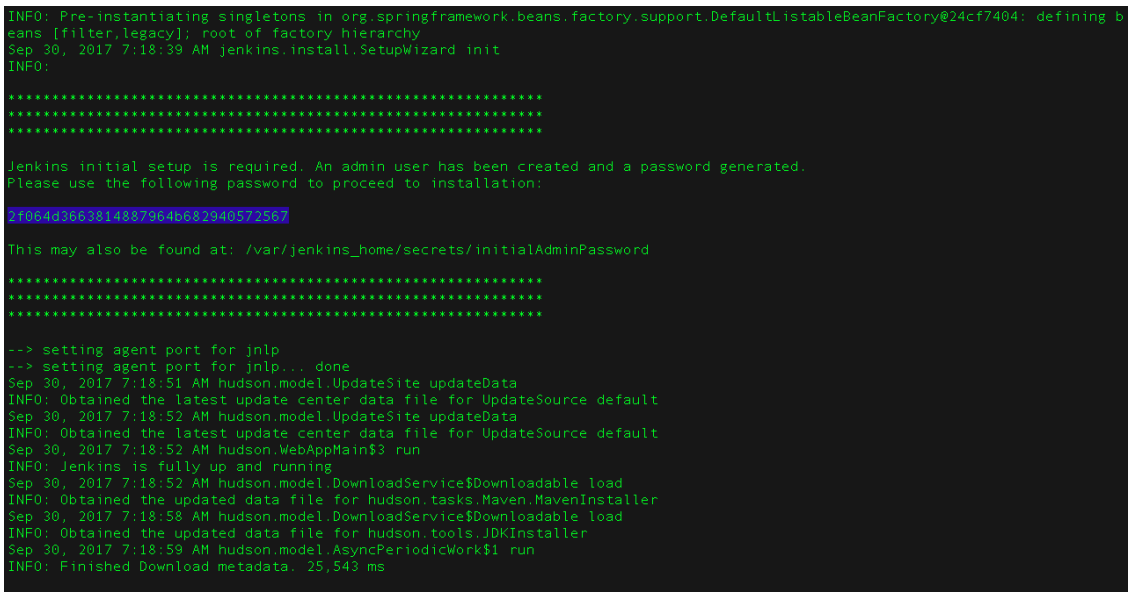


Рис. 3.7. Пароль для розблокування Jenkins можна знайти у журналі серверу під час його першого запуску.

Після початкового налаштування Jenkins запропонує встановити найпопулярніші плагіни, проте цей крок можна пропустити. Наступною фазою буде створення архітектури для безперервної інтеграції коду та його тестування. Після завершення цього етапу, перейдемо до налаштувань безперервної доставки та розгортання програмного забезпечення.

3.4. Побудова та конфігурування CI/CD конвеєру

Після вибору інструменту для реалізації CI/CD конвеєру та його підготовки до налаштування, я рекомендую ознайомитися зі структурною схемою (рис. 3.8), яка демонструє процеси для вирішення задачі.

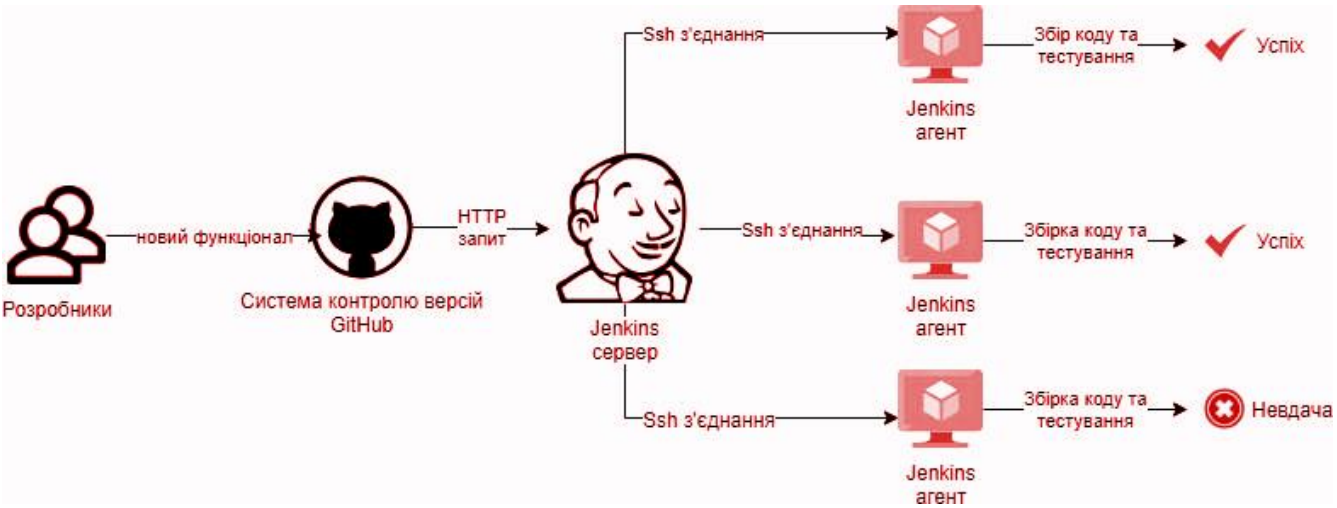


Рис. 3.8. Схема реалізації CI за допомогою Jenkins розгорнутому на сервері, Jenkins-агентів, які виконують завдання та репозиторію коду, який розміщений на Github

Також хотілося б зазначити, що ця робота не мала за мету детально аналізувати системи контролю версій або інші вторинні інструменти, використовувані у розробці ПЗ, тому інформація надається у стислому та загальному форматі.

У рамках цього проекту я планую використовувати загально прийняту стратегію розгалуження коду, яка включає створення окремих гілок у системі контролю версій для кожної функціональності. Коли розробник створює свою гілку в Git, він подає запит на злиття (merge request) з основною гілкою, яка за замовчуванням називається master. Це викликає HTTP(s) запит до серверу з встановленим Jenkins, де за допомогою WebHook плагіну налаштовується Jenkins для проведення повної збірки та тестування коду за умови отримання такого запиту.

Збірка коду буде проводитися автоматично за допомогою інструменту Maven. На етапі тестування запускатимуться автоматизовані тести, розроблені інженерами, що в свою чергу також впливатимуть на успішність загального процесу CI/CD.

Завершальним кроком впровадження та налаштування CI стане відсилання повідомлень розробникам про стан їх коду та автоматичне схвалення запитів на злиття, якщо весь процес CI/CD пройшов успішно.

3.4.1. Налаштування плагінів

Спочатку необхідно встановити плагін GitHub для Jenkins, який дозволяє ефективно та швидко налаштувати інтеграцію між Jenkins та GitHub. Для цього слід перейти до розділу «Manage Jenkins» у лівій частині інтерфейсу Jenkins, а потім вибрати опцію «Manage Plugins», як це зображено на рисунках 3.9 та 3.10.

Рис. 3.9. Налаштування плагінів

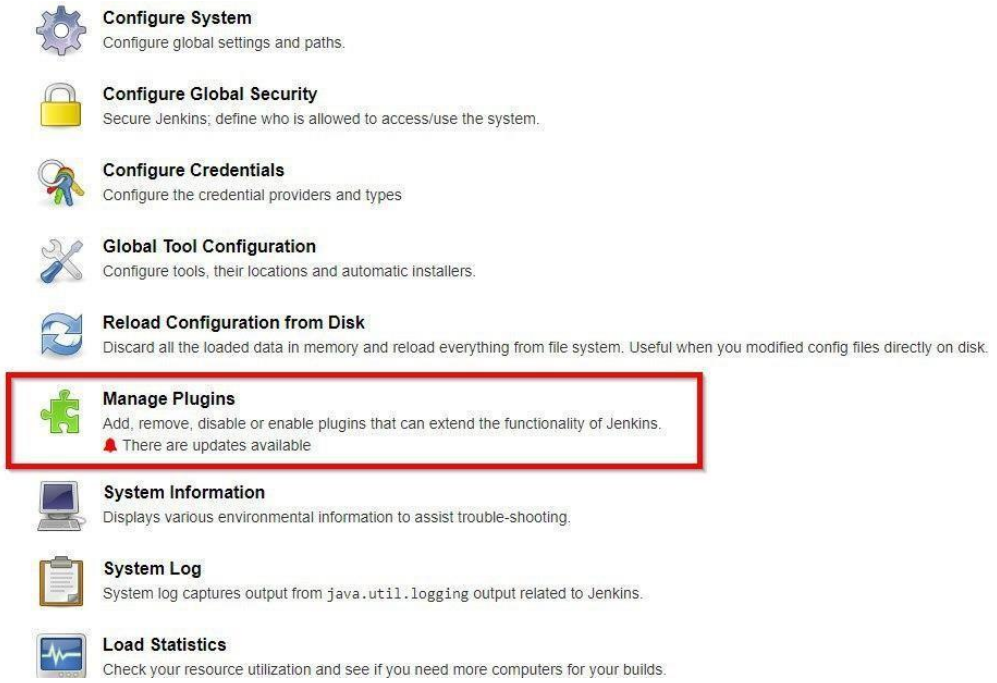


Рис. 3.10. Налаштування плагінів

У розділі управління плагінами Jenkins (рис. 3.11) можна переглянути вже інстальовані плагіни та встановити нові, доступні в репозиторії. Для інсталяції плагіна GitHub достатньо перейти до вкладки «Available» і використовувати пошук для знаходження плагіна «GitHub Plugin». Після вибору плагіна, слід натиснути «Install without restart», що дозволяє встановити плагін без перезавантаження сервера Jenkins.

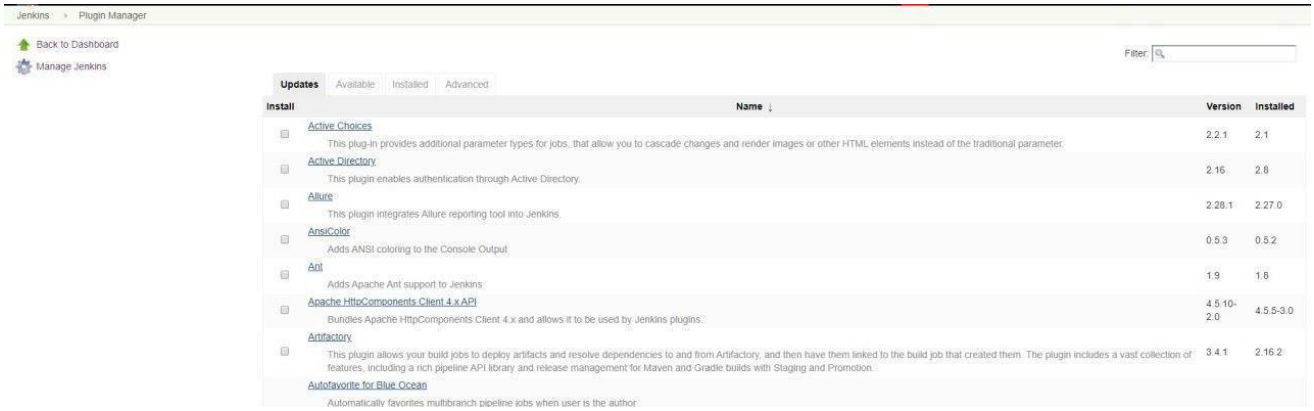


Рис. 3.11. Налаштування плагінів

Завершивши інсталяцію необхідного плагіна, можна приступити до налаштування інтеграції з GitHub, де зберігається вихідний код. Для цього потрібно створити Jenkins job (рис. 3.12), який слугуватиме як основа для CI процесу, структура якого була описана раніше.

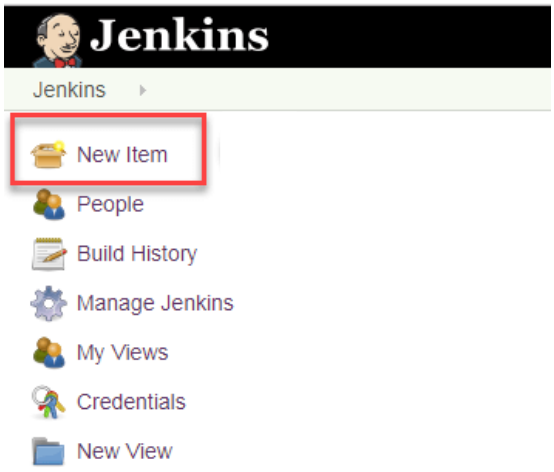


Рис. 3.12. Створення нового завдання

Тип проекту вказуємо «Freestyle» і клікаємо на кнопку “OK” (рис. 3.12)

Рис. 3.12. Створення завдання

Далі, у вкладці «Source Code Management» (рис 3.12) клікаємо на кнопку радіогрупи Git та у полі Repository URL вказуємо посилання на репозиторій, у якому розташований вихідний код ПЗ.

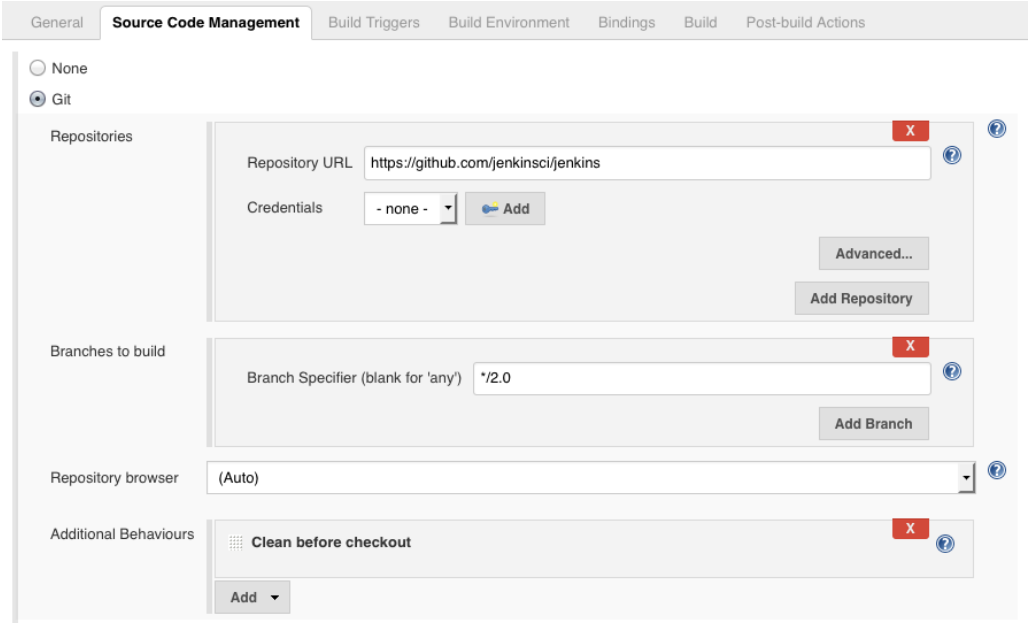


Рис. 3.12. Налаштування джерела вихідного коду у репозиторії

В полі «Branch Specifier» необхідно вказати гілку в Git, яка буде використовуватися для роботи. Вибираємо гілку master. Завершивши налаштування плагіна GitHub, Jenkins тепер автоматично визначатиме, де знаходиться вихідний код, над яким буде проводитися робота в межах нашого CI/CD конвеєру.

3.4.2. Налаштування агентів

Щоб Jenkins міг здійснити збірку коду та провести автоматизоване тестування, необхідно підключити до нього агентів Jenkins. Ці агенти, які в даному випадку будуть представлені стандартними віртуальними машинами на базі Linux, відповідатимуть за процес збірки та тестування.

Для додавання нового агента до Jenkins слід виконати кілька кроків. Спочатку треба встановити плагін «SSH Slaves Plugin». Потім переконатися, що на віртуальній машині, яку ми плануємо використовувати як агент, активовано ssh сервіс. Необхідно також створити користувача Linux з можливістю підключення через ssh та створити пару ключів шифрування для забезпечення безпечного з'єднання.

Після налаштування віртуальної машини можна приступити до реєстрації агента в системі Jenkins. Для цього слід вибрати розділ «Manage Jenkins», як це було описано раніше, і перейти до «Manage Nodes». На сторінці налаштування агентів потрібно натиснути на вкладку «New Node» (рис. 3.13) для додавання нового агента.

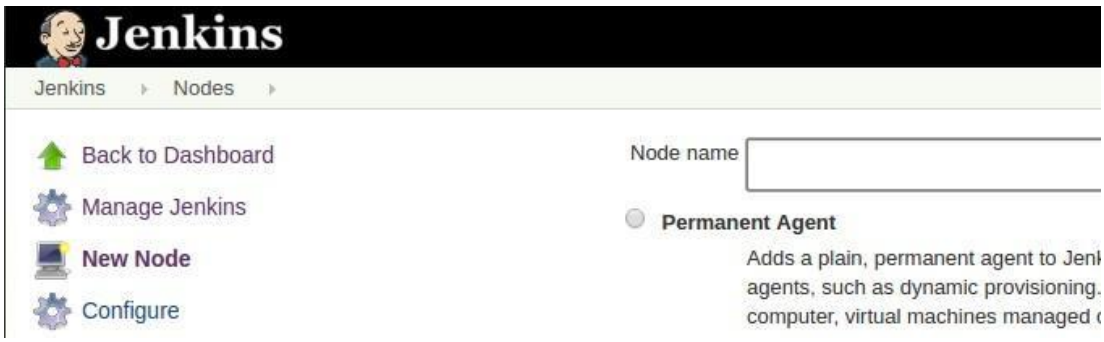


Рис. 3.13. Створення агента

Даємо назву агенту і клікаємо на кнопку “OK” (рис. 3.14). Після цього ми одразу потрапимо на сторінку з детальними налаштуваннями агенту.

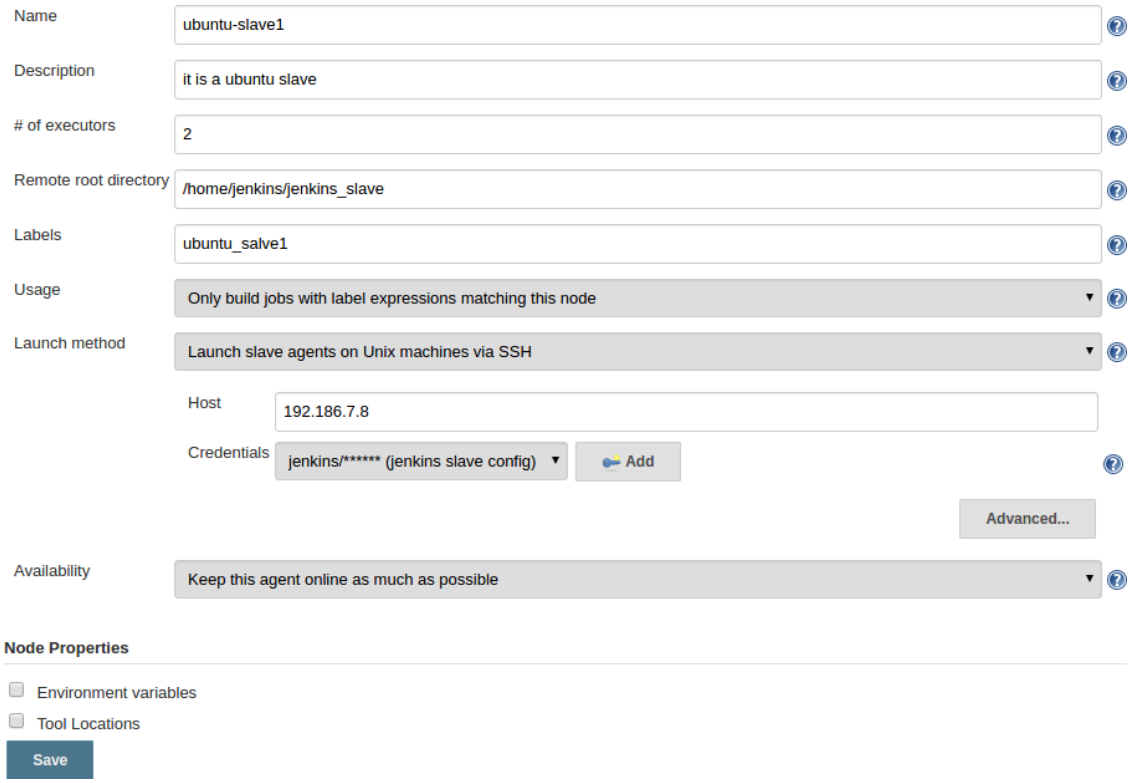


Рис.3.14. Конфігурація агента

На цій сторінці доступне редагування назви, опису, максимальної кількості одночасних Jenkins Job, домашньої директорії для збирання та тестування коду, завантаженого з GitHub, Host (IP-адреси підключеної віртуальної машини) та даних для авторизації (Credentials) — ім’я користувача та пароль, або публічний ключ, створений для ssh з’єднання.

Після натискання кнопки збереження, у разі правильного введення всіх даних, ми можемо побачити, що агент був успішно підключений і готовий до використання.

3.4.3. Налаштування Jenkins Job для збирання проекту з вихідного коду та тестування створеного образу додатку

Після налаштування агента і плагіну для інтеграції з GitHub можна приступити до конфігурації CI та CT процесів у пайплайні, а саме — налаштування раніше створеної Jenkins Job.

Щоб сконфігурувати Jenkins використовувати саме вибраний агент, потрібно увійти в налаштування Jenkins Job з назвою Build та встановити прапорець біля опції «Restrict where this project can be run» (рис. 3.15). Далі у полі «Label Expression» необхідно вказати назву агента або мітку (Label), яку ви визначили під час створення агента.

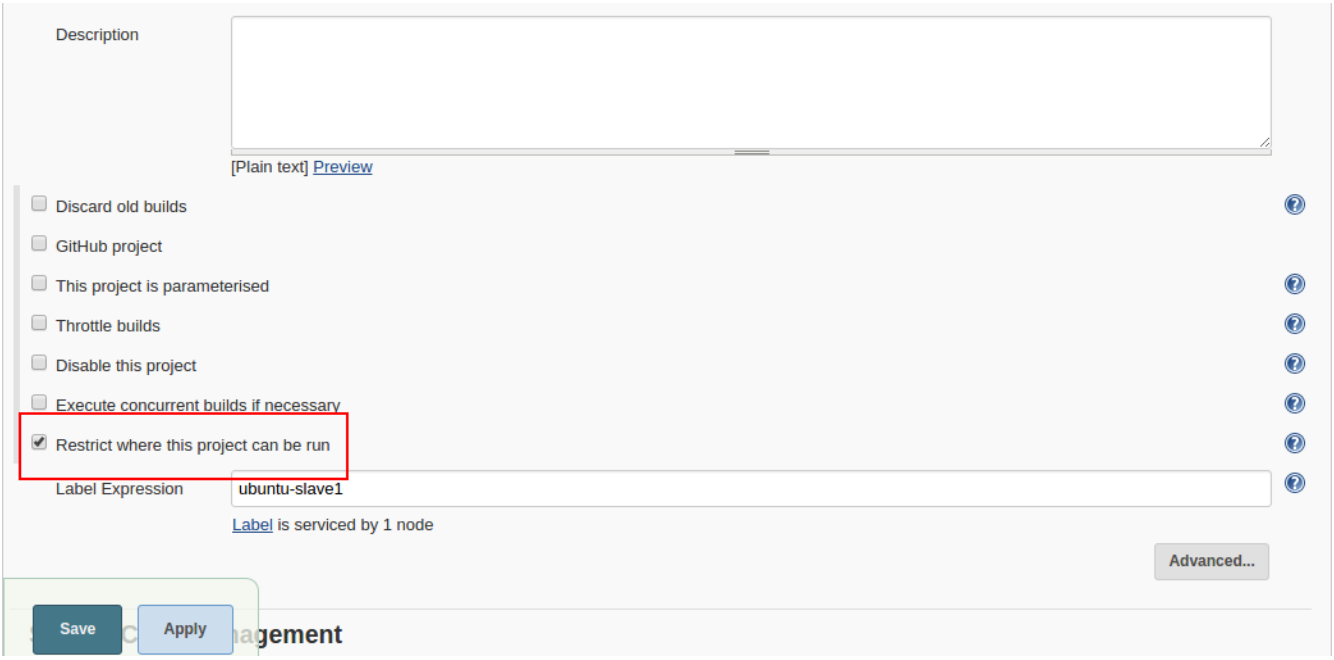


Рис. 3.15 Конфігурація агента

Після встановлення агента для Jenkins Job наступним кроком є конфігурація ключової частини CI конвеєру — збірки та тестування проекту. У секції «Build» слід обрати опцію «Add build step», а потім вибрати «Invoke top-level Maven targets» (рис 4.16). Ці налаштування дозволяють Jenkins Job запускати автоматизований інструмент для збірки проектів Maven із зазначеними цілями під час виконання.

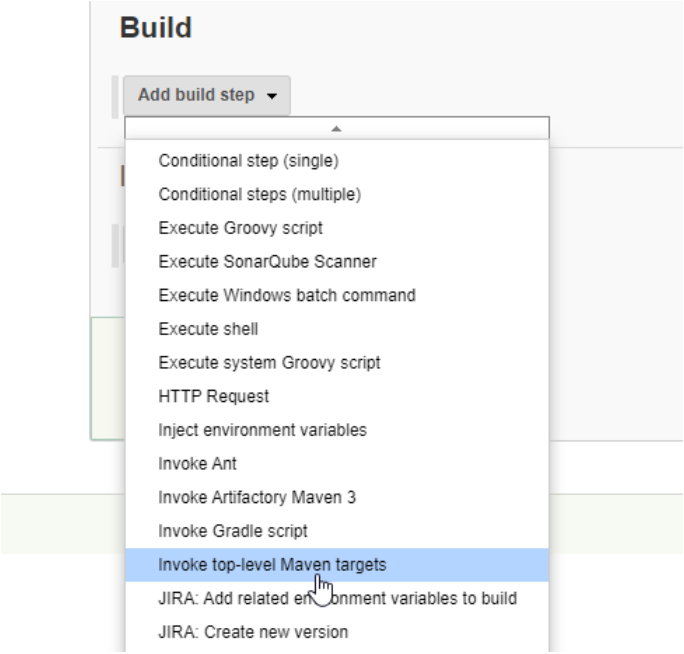


Рисунок 3.16 – Налаштування збірки вихідного коду проекту

У полі «Goals» Maven вводимо визначені цілі, які будуть керувати процесом збірки вихідного коду та виконання модульних тестів. Оскільки детальний аналіз різних видів тестування коду не входить у рамки даної роботи, у теоретичній частині ця тема не розкривається. Проте, нижче коротко описано модульні тести, які використовуються у практичній частині як елемент безперервного тестування.

Комплексні програмні системи складаються з окремих модулів, кожен з яких виконує певну функцію. Модульне тестування дозволяє перевірити правильність функціонування кожного модуля окремо, що спрощує ідентифікацію та виправлення помилок.

У Jenkins є корисний плагін JUnit, який дозволяє переглядати результати модульного тестування Java коду та візуалізувати загальну трендову динаміку. Після інсталяції плагіну, у секції «Post-build Action» конфігурації Build Job необхідно вибрати відповідний пункт (рис. 3.17).

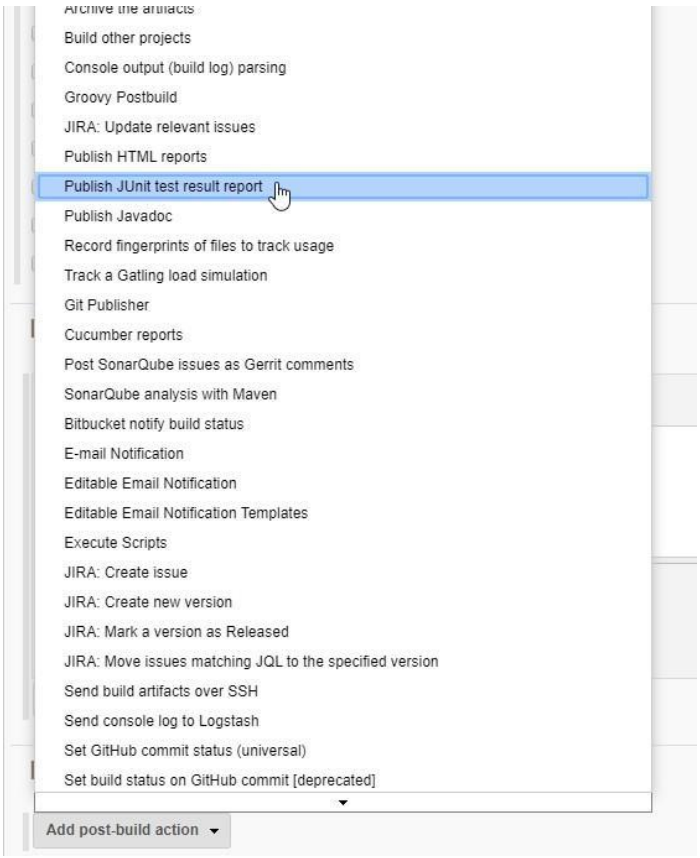


Рисунок 3.17 – Налаштування Build Job.

На завершальному етапі налаштування Build Job слід визначити «Build triggers», тобто критерії, за якими ця Job буде автоматично запускатися. Є декілька стандартних умов запуску, проте за допомогою різноманітних плагінів цей список можна розширити, щоб відповідати специфічним потребам проекту. У контексті цієї демонстрації достатньо використовувати вбудовані функції Jenkins. Виберемо опцію «Build periodically» (рис. 3.18-3.19), що дозволяє запускати цю Jenkins Job автоматично з певною періодичністю, наприклад, раз на годину.

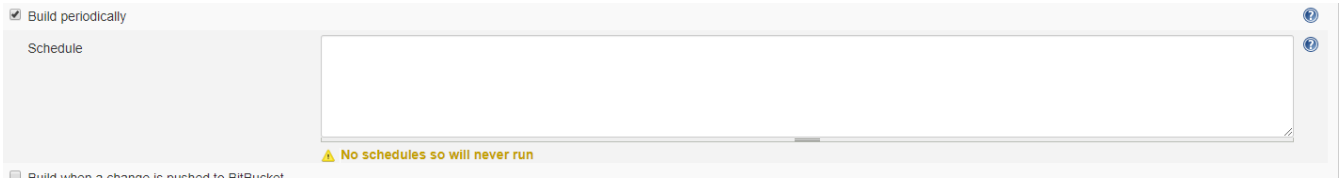


Рисунок 3.18 – Налаштування періодичності збірки та тестування проекту

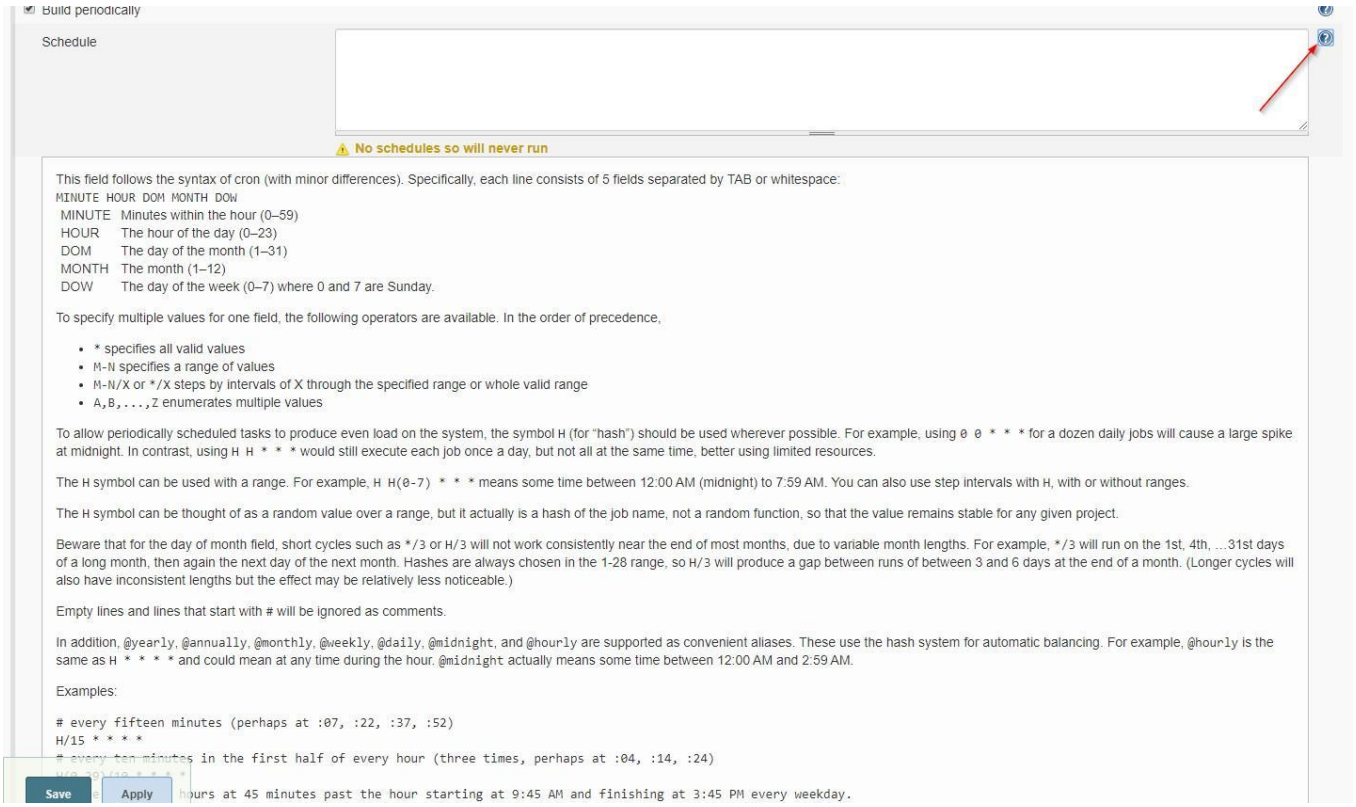


Рисунок 3.19 – Налаштування періодичності збірки та тестування проекту

Після ретельного ознайомлення з вбудованою документацією про використання цієї функції, введемо у поле «Schedule» відповідні параметри для щогодинного запуску, як демонструється на рисунку 3.20.

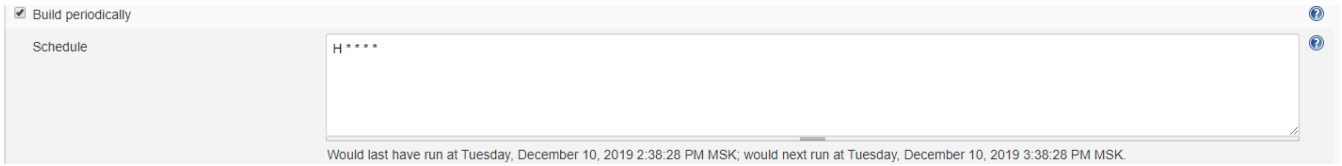


Рисунок 3.20 – Налаштування запуску завдання щогодинно

Незважаючи на те, що ми налаштували виконання Jenkins Job кожну годину автоматично, ви все ще маєте можливість запускати цю Job власноруч натиснувши кнопку «Build Now» (рис 3.21)

Рисунок 3.21 – Ручне виконання Jenkins Job

З метою демонстрації, виконаємо завдання вручну примусово. Одразу після запуску Jenkins Job, в лівій частині екрану можна переглянути історію попередніх запусків та стан поточних процесів. Як видно на рисунку 3.22, в даний момент Jenkins Job перебуває у стадії виконання.

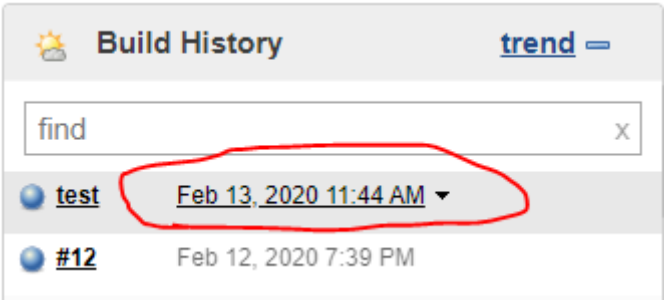


Рисунок 3.22 – Історія запусків Jenkins Job

У разі успішної збірки проекту без помилок компіляції та з успішно пройденими модульними тестами, Build Job зафарбується зеленим (або синім) кольором.

Щоб ознайомитися з результатами модульних тестів, слід вибрати Build Job зі списку всіх доступних Job. На рисунку 3.23 можна побачити графік, створений за допомогою раніше встановленого плагіна JUnit Plugin.

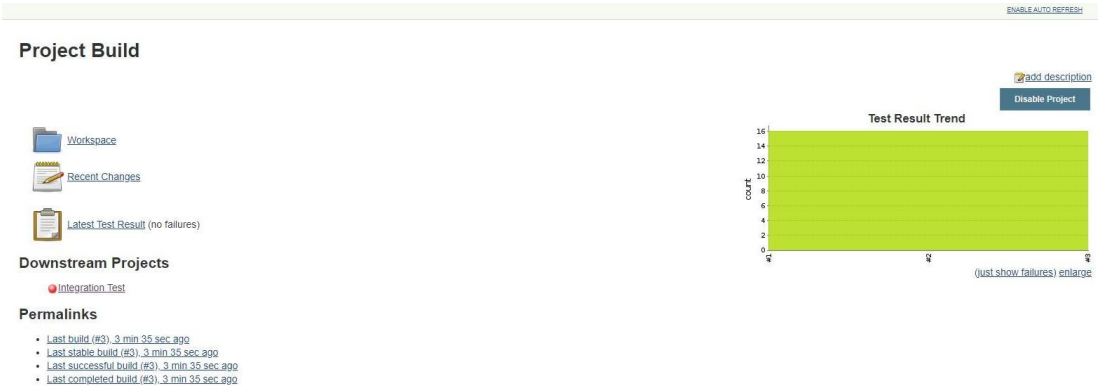


Рисунок 3.23 – Огляд успішно виконаної Build Job та результати unit тестів

Щоб доповнити наш пайплайн інтеграційним тестуванням, створимо ще один Jenkins Job під назвою Integration Test, як показано на рисунку 4.24.

Рисунок 3.24 – Jenkins Job для інтеграційного тестування

Процес створення тестів для цього проекту не буде детально описаний. Основний фокус у рамках CI/CD/CT полягає у налаштуванні автоматичного конвеєру, який здійснює збірку, тестування та розгортання коду, розробленого програмними інженерами. Що стосується налаштування Integration Test Job, оптимальним варіантом буде налаштувати її на виконання відразу після успішної збірки проекту. Тобто, якщо Build Job завершилась успішно, наступним кроком повинно бути інтеграційне тестування. Для досягнення цього, у налаштуваннях Integration Test у розділі «Build Triggers» необхідно вибрати опцію «Build after other projects are built» (рис. 3.25), тобто запускати цю Jenkins Job негайно після завершення іншої Job.

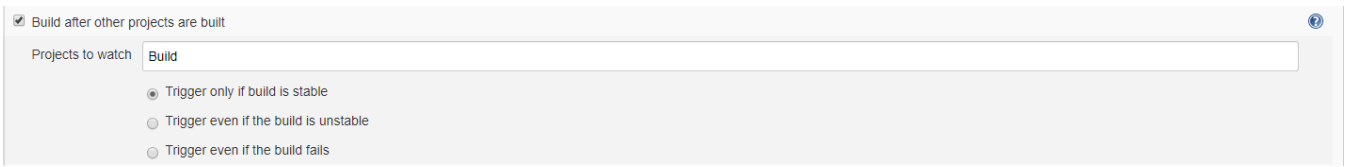


Рис. 3.25 Налаштування додаткового завдання для інтеграційного тестування

3.4.4. Створення та налаштування системи оповіщень про результати виконання завдань в Jenkins

Останнім кроком налаштування CI для виконання завдання буде відправка розробникові електронного листа з інформацією про результати завдання конвеєру його коду.

Для цього у вкладці налаштувань кожного з раніше створених Jenkins Job необхідно перейти до секції «Post-build Actions», де зазначаються дії, які мають бути виконані після завершення Jenkins Job. Там потрібно додати опцію «E-mail Notification» (рис. 3.26).

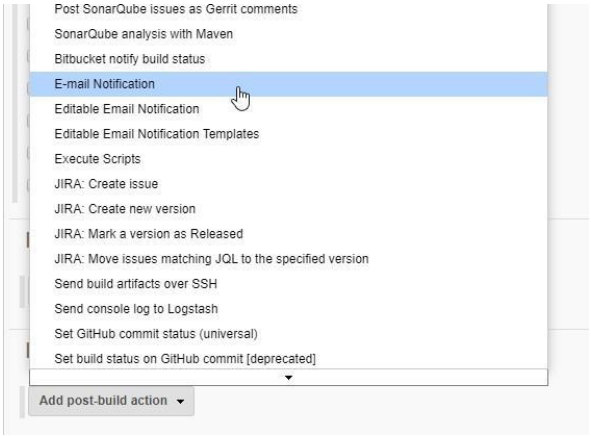


Рисунок 3.26 – Налаштування оповіщень про завершену Jenkins Job

Далі, у поле «Recipients» слід внести електронну адресу того, хто буде отримувати повідомлення.

Таким чином, завдяки простим крокам, ми налаштували функціонуючий CI та CT, який автоматично запускається через певний інтервал часу та надсилає розробникові інформацію про результати збірки та тестування його частини коду, розташованої у гілці master.

3.5. Доповнення до створеного CI конвеєру компоненти неперервної доставки (CD)

У практиці розробки програмного забезпечення часто трапляється, що continuous delivery (CD) та continuous deployment сприймаються як одне й те ж, із загальною абревіатурою CD, не роблячи чіткої різниці між цими поняттями. Проте, як було визначено в другому розділі, де представлено теоретичні аспекти

безперервної інтеграції (CI), тестування (CT), розгортання та доставки (CD), існують виразні межі між цими концепціями. Безперервна доставка передбачає автоматизацію всіх етапів конвеєру, крім розгортання в робочому середовищі, тоді як безперервне розгортання означає, що навіть останній крок (розгортання у робоче середовище) є автоматизованим. У практичній частині цієї роботи я продемонструю архітектуру та приклад використання саме підходу continuous deployment, який доповнюватиме раніше налаштовані процеси continuous integration та continuous testing у межах єдиного великого CI/CD конвеєру на платформі Jenkins.

Створення CD потребує ініціації двох нових Jenkins Job: один для розгортання у середовище staging (передвиробниче тестування) та інший для розгортання у виробниче середовище (production deployment). Крім того, слід створити ще одну Jenkins Job, призначену для виконання регресійного тестування програмного забезпечення у середовищі staging, яку назвемо Test Regression. На рисунку 3.27 представлено всі Jenkins Job, що входять у склад даного проекту.

Рис. 3.27. Перелік створених завдань для побудови CD

Спочатку треба інтегрувати тестове середовище staging з Jenkins, підключивши його як звичайного агента. Це дозволить Jenkins серверу виконувати необхідні дії на тестових серверах. Після налаштування агентів, у конфігурації Job staging deployment встановлено параметр «Restrict where this project can be run», подібно до того, як це було зроблено для Jenkins Job у процесі налаштування CI та CT у підрозділі 4.4. Важливо також встановити параметр «Build after other projects are built», щоб Jenkins Job запускалася лише після успішного завершення Integration Test.

Наступним кроком буде розгортання версії коду, що пройшла тестування, на тестовому середовищі за допомогою кроку «Build», але вже з іншими цілями Maven, які забезпечують не тільки збірку проекту, а й його розгортання та тестування.

Конфігурація Test Regression включатиме запуск тестів для додатку, розгорнутого в тестовому середовищі. Цей Jenkins Job буде виконуватися відразу після успішного розгортання в тестовому середовищі.

Ця ж концепція застосовується і для налаштування розгортання в продакшн середовище.

3.6. Огляд створеного CI/CD конвеєру

На рис. 3 наведена загальна схема створеного в рамках даної кваліфікаційної роботи CI/CD конвеєру.

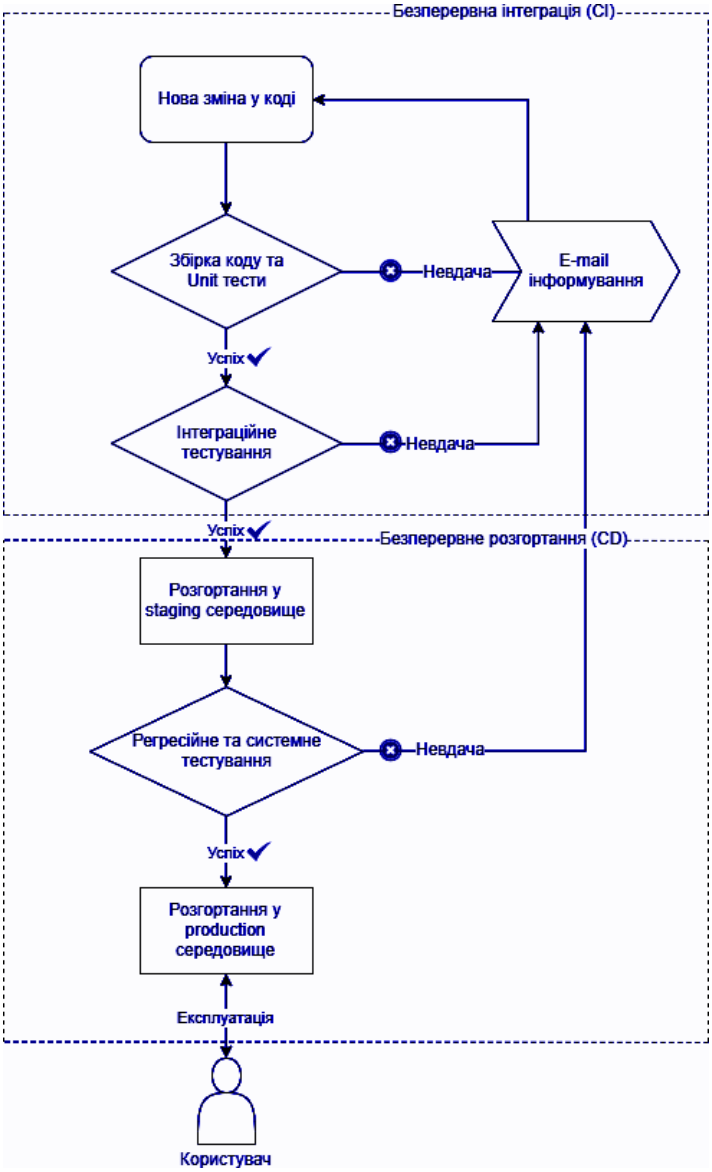


Рис. 3.28. Загальна схема створеного CI/CD конвеєру

Автоматизація процесу CI/CD за допомогою Jenkins є ефективною, хоча часто на реальних проектах вимагає високий рівень кваліфікації інженерів, що демонструє

переваги використання цього інструменту. Згідно з описаним у першому та другому розділах, автоматизація сприяє значному прискоренню процесу розробки ПЗ та зниженню ризику людської помилки під час тестування. Система, розроблена у третьому розділі, надає замовнику можливість отримувати нові версії додатку регулярно, частіше випускати оновлення та адаптуватися до змін у вимогах. Без застосування цих методів досягти таких результатів було б значно складніше. Завдяки автоматизованій збірці проекту та тестуванню, що відбувається регулярно, процес розробки на всіх етапах життєвого циклу значно прискорюється. Переваги використання такої системи є вагомими та очевидними.

Висновки до розділу 3

У третьому розділі роботи було розглянуто моделювання, аналіз та практичну реалізацію CI/CD конвеєру на прикладі конкретного проекту, використовуючи Jenkins як основний інструмент. Робота включала детальний опис процесу розгортання Jenkins у робочому середовищі та налаштування CI/CD конвеєру з урахуванням специфіки задачі проекту. Особлива увага була приділена налаштуванню плагінів, агентів та Jenkins Job для збирання проекту з вихідного коду та тестування створеного образу додатку.

Ключовим аспектом роботи стало створення та налаштування системи оповіщень, яка відіграє важливу роль у забезпеченні постійної зворотнього зв'язку та моніторингу ефективності процесів CI/CD. Додатково було реалізовано компоненти неперервної доставки (CD), які є важливою складовою успішного CI/CD конвеєру.

У підсумку, розділ надає повне уявлення про практичне втілення та функціонування CI/CD конвеєру, демонструючи його ефективність та важливість для сучасних проектів розробки ПЗ. Огляд створеного конвеєру підтвердив його здатність оптимізувати процес розробки, забезпечуючи високу швидкість випуску оновлень та підвищення якості кінцевого продукту.

ВИСНОВКИ

У даній кваліфікаційній роботі було проведено детальний аналіз методологій розробки програмного забезпечення та концепцій CI/CD, а також їх практична реалізація з використанням обраного інструменту Jenkins.

У першому розділі роботи були розглянуті різні етапи життєвого циклу програмного забезпечення та ключові методології розробки, включаючи Waterfall, V-Model, Agile та DevOps. Особлива увага приділялася огляду концепцій безперервної інтеграції та доставки, а також їх переваг та недоліків у контексті сучасної розробки ПЗ.

Другий розділ був присвячений аналізу та порівнянню інструментів, доступних на ринку для побудови CI/CD конвеєру, зокрема Jenkins, GitLab, Travis CI, та Go CD. Були визначені критерії для оцінки та порівняння цих інструментів, що дало змогу зробити обґрунтований вибір інструменту для подальшої практичної реалізації.

Третій розділ охоплював практичну імплементацію CI/CD конвеєру на базі Jenkins. Було виконано моделювання задачі, аналіз можливих шляхів її розв'язання, розгортання Jenkins у робочому оточенні, побудова та конфігурація CI/CD конвеєру, включаючи налаштування плагінів, агентів, Jenkins Job для збирання проекту та тестування. Також було розроблено систему оповіщень для моніторингу результатів виконання завдань в Jenkins та доповнено конвеєр компонентами неперервної доставки.

Загалом, кваліфікаційна робота демонструє, що застосування CI/CD методологій та інструментів значно підвищує ефективність розробки програмного забезпечення, сприяючи більш швидкому виявленню помилок, підвищенню якості продукту та оптимізації робочого процесу. Використання Jenkins як основного інструменту для реалізації CI/CD конвеєру показало його гнучкість та ефективність у вирішенні задач сучасного розробника ПЗ.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API (Application Programming Interface) – прикладний програмний інтерфейс

Continuous Deployment or Delivery – неперервне розгортання/доставка

Continuous Integration – неперервна інтеграція

IAC (Infrastructure as a Code) – інфраструктура як код

QA – (Quality Assurance) – забезпечення якості

ОС – операційна система

ПЗ – програмне забезпечення

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. De Bois P. "DevOps: A Guide to Implementing Methodology" / P. De Bois. – O'Reilly Media, 2016. – 250 с.

2. Kim G., Behr K., Spafford G. "The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win" / G. Kim, K. Behr, G. Spafford. – IT Revolution Press, 2015. – 300 с.

3. Humble J., Farley D. "Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation" / J. Humble, D. Farley. – Addison-Wesley, 2010. – 512 с.

4. Forsgren N., Humble J., Kim G. "Accelerate: The Science of Lean Software and DevOps" / N. Forsgren, J. Humble, G. Kim. – IT Revolution Press, 2018. – 288 с.

5. Hüttermann M. "DevOps for the Modern Enterprise" / M. Hüttermann. – Apress, 2014. – 256 с.

6. Willis J., Edwards D. "DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations" / J. Willis, D. Edwards. – IT Revolution Press, 2016. – 480 с.

7. Newman S. "Building Microservices: Designing Fine-Grained Systems" / S. Newman. – O'Reilly Media, 2015. – 280 с.

8. Fowler M. "Microservices: Patterns and Best Practices" / M. Fowler. – O'Reilly Media, 2016. – 172 с.

9. Goldfuss A. "Scalable: Building Infrastructure for Growing IT Systems" / A. Goldfuss. – O'Reilly Media, 2017. – 320 с.

10. Grenny J. W. "DevOps: Practical Advice for Successful Implementation" / J. W. Grenny. – Apress, 2016. – 196 с.

11. Jenkins [Електронний ресурс] – Режим доступу: https://www.jenkins.io/doc/book/installing/linux/

12. Wlodarczyk T. W. "DevOps: A Software Architect's Perspective" / T. W. Wlodarczyk. – Addison-Wesley Professional, 2015. – 400 с.

13. Loukides M. "What is DevOps?" [Електронний ресурс] / M. Loukides. – Режим доступу: https://www.oreilly.com/content/what-is-devops/

14. Roche J. "Adopting DevOps Practices in Quality Assurance" [Електронний ресурс] / J. Roche. – Режим доступу: https://queue.acm.org/detail.cfm?id=2540984

15. Loukides M., Saffo P. "Software Development and Operations in the 21st Century" [Електронний ресурс] / M. Loukides, P. Saffo. – Режим доступу: https://www.oreilly.com/ideas/software-development-and-operations-in-the-21st-century

16. Official GitLab CI/CD Documentation [Електронний ресурс]. – Режим доступу: https://docs.gitlab.com/ee/ci/

17. Jenkins User Documentation [Електронний ресурс]. – Режим доступу: https://www.jenkins.io/doc/

18. Travis CI Documentation [Електронний ресурс]. – Режим доступу: https://docs.travis-ci.com/

19. GoCD User Documentation [Електронний ресурс]. – Режим доступу: https://docs.gocd.org/

20. Argo CD - GitOps Continuous Delivery [Електронний ресурс]. – Режим доступу: https://argo-cd.readthedocs.io/

21. Shahin M., Babar M. A., Zhu L. "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices" [Електронний ресурс] / M. Shahin, M. A. Babar, L. Zhu. – IEEE Access, 2017. – Режим доступу: https://ieeexplore.ieee.org/document/7930218/

22. Kim G., Behr K., Spafford G. "The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win" / G. Kim, K. Behr, G. Spafford. – IT Revolution Press, 2015. – 300 с.

23. Humble J., Farley D. "Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation" / J. Humble, D. Farley. – Addison-Wesley, 2010. – 512 с.

24. Forsgren N., Humble J., Kim G. "Accelerate: The Science of Lean Software and DevOps" / N. Forsgren, J. Humble, G. Kim. – IT Revolution Press, 2018. – 288 с.

25. Willis J., Edwards D. "DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations" / J. Willis, D. Edwards. – IT Revolution Press, 2016. – 480 с.

26. Newman S. "Building Microservices: Designing Fine-Grained Systems" / S. Newman. – O'Reilly Media, 2015. – 280 с.

27. Fowler M. "Microservices: Patterns and Best Practices" / M. Fowler. – O'Reilly Media, 2016. – 172 с.

28. Turnbull J. "The Docker Book: Containerization Is the New Virtualization" / J. Turnbull. – James Turnbull, 2014. – 270 с.

29. Goldfuss A. "Scalable: Building Infrastructure for Growing IT Systems" / A. Goldfuss. – O'Reilly Media, 2017. – 320 с.

30. Hüttermann M. "DevOps for the Modern Enterprise" / M. Hüttermann. – Apress, 2014. – 256 с.

31. Grenny J. W. "DevOps: Practical Advice for Successful Implementation" / J. W. Grenny. – Apress, 2016. – 196 с.

ДОДАТКИ

Додаток А

A.1 Jenkins Pipeline

```
pipeline {
    environment {
        registry = ""
        registryCredential = 'dockerhub'
        dockerImage = ''
    }

    agent {
        node {
            label 'docker-engine'
        }
    }

    // tools {
    //     nodejs "NodeJS 20.2.0"
    // }

    stages {
        stage('Cloning Git') {
            steps {
                git branch: 'main',
                        credentialsId:,
                        url: ''
            }
        }
        // stage('Build') {
        //     steps {
        //         sh 'npm install --include=dev'
        //         sh 'npm run build'
        //     }
        // }
        // stage('Test') {
        //     steps {
        //         sh 'npm test'
        //     }
        // }
        stage('Building image') {
            steps{
                script {
                    dockerImage = docker.build registry + ":$BUILD_NUMBER"
                }
            }
        }
        stage('Deploy Image') {
            steps{
                script {
                    docker.withRegistry( '', registryCredential ) {
                        dockerImage.push()
                    }
                }
            }
        }
        stage('Remove Unused docker image') {
            steps{
```

```
                sh "docker rmi $registry:$BUILD_NUMBER"
            }
        }
    }
}
```

А.2 Маніфест для автоматичного розгортання Jenkins на сервері. kustomization.yaml

```
## Jenkins Kustomization

apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

# Making sure all resources used in this tutorial are created in a dedicated namespace
# Also specific annotations are added for later identification
namespace: jenkins
commonAnnotations:
  provider: https://jenkins.io
# Jenkins resources (namespace, services, deployments, etc)
resources:
  - resources/namespace.yaml
  - resources/deployment.yaml
  - resources/volume.yaml
  - resources/service.yaml
```

А.3 Маніфест для автоматичного розгортання Jenkins на сервері. deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: jenkins
  labels:
    app: jenkins
spec:
  replicas: 1
  selector:
    matchLabels:
      app: jenkins
  template:
    metadata:
      name: jenkins
      labels:
        app: jenkins
    spec:
      securityContext:
        fsGroup: 1000
        runAsUser: 0
      containers:
        - name: jenkins
          image: jenkins/jenkins:lts
          imagePullPolicy: IfNotPresent
          ports:
            - name: http-port
              containerPort: 8080
            - name: jnlp-port
              containerPort: 50000
          volumeMounts:
            - name: jenkins-volume
              mountPath: /var/jenkins_home
```

```
      volumes:
        - name: jenkins-volume
          persistentVolumeClaim:
            claimName: jenkins-pvc
```

А.4 Маніфест для автоматичного розгортання Jenkins на сервері. service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: jenkins
spec:
  type: NodePort
  ports:
    - port: 8080
      targetPort: 8080
      nodePort: 30001
  selector:
    app: jenkins

---

apiVersion: v1
kind: Service
metadata:
  name: jenkins-jnlp
spec:
  type: ClusterIP
  ports:
    - port: 50000
      targetPort: 50000
  selector:
    app: jenkins
```

А.5 Маніфест для автоматичного розгортання Jenkins на сервері. volume.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: jenkins-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
  storageClassName: do-block-storage
```

Додаток Б

Копія публікації статті за темою кваліфікаційної роботи

ОЛІЙНИК С.С., ДЕМКІВСЬКА Т.І.

АВТОМАТИЗАЦІЯ ПРОЦЕСІВ РОЗРОБКИ ТА ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЗА ДОПОМОГОЮ DEVOPS МЕТОДОЛОГІЇ

SERHII OLIINYK, TETIANA DEMKIVSKA

AUTOMATION OF DEVELOPMENT PROCESSES AND IMPLEMENTATION OF SOFTWARE USING THE DEVOPS METHODOLOGY

In today's information society, where the speed of technology development is a key success factor, the process of software development and implementation is becoming increasingly complex and requires effective approaches. One of these approaches is the DevOps methodology, which combines development and operations in order to automate processes and improve cooperation between developers and operators. This article aims to consider the possibilities and benefits of using DevOps methodology in the process of software development and implementation.

Вступ

У сучасному інформаційному суспільстві, де швидкість розвитку технологій є ключовим фактором успіху, процес розробки та впровадження програмного забезпечення стає все більш складним і вимагає ефективних підходів. Одним з таких підходів є DevOps методологія, яка поєднує розробку (Development) та експлуатацію (Operations) з метою автоматизації процесів і покращення співпраці між розробниками та операторами. У цій статті ставиться завдання розглянути можливості та вигоди використання DevOps методології в процесі розробки та впровадження програмного забезпечення.

Постановка завдання

Головною метою цього дослідження є дослідити та проаналізувати процес розробки та впровадження програмного забезпечення з використанням DevOps методології. Конкретні завдання дослідження включають:

1. Визначення концепції та принципів DevOps методології.
2. Аналіз ролей та відповідальності учасників процесу розробки та впровадження програмного забезпечення в рамках DevOps.
3. Огляд інструментів та технологій, використовуваних для автоматизації процесів DevOps.
4. Вивчення переваг впровадження DevOps методології в організації розробки програмного забезпечення.

5. Дослідження впливу DevOps на якість, швидкість та ефективність розробки та впровадження програмного забезпечення.

6. Аналіз прикладів успішної реалізації DevOps методології у великих організаціях.

Основна частина

2. Визначення концепції та принципів DevOps методології

Методологія DevOps - це підхід до розробки програмного забезпечення, який поєднує розробку (Development) і експлуатацію (Operations). Вона ставить за мету забезпечити швидку, безперервну і надійну поставку програмних продуктів.

DevOps спрямований на спільну роботу розробників програмного забезпечення (розробка, тестування, збирання, пакування) і операторів (інфраструктура, встановлення, моніторинг, налагодження). Метою DevOps є зниження часу, зусиль та ризиків, пов'язаних з розробкою і експлуатацією програмного забезпечення.

Головні принципи методології DevOps включають:

Спільна власність: Команди розробки і експлуатації повинні спільно володіти відповідальністю за результати своєї роботи. Це стимулює співпрацю та спільні цілі.

Автоматизація: Використання автоматизації дозволяє прискорити процеси розробки, тестування та розгортання. Автоматична збірка, тестування і розгортання програмного забезпечення дозволяють швидко та безпомилково впроваджувати зміни.

3. Рис. 1. Аналіз ролей та відповідальності учасників процесу розробки та впровадження програмного забезпечення в рамках DevOps

Постійна інтеграція та постійна доставка (CI/CD): Цей принцип передбачає постійне злиття (інтеграцію) коду в основну гілку розробки та постійну доставку готового програмного забезпечення до виробничого середовища. Це допомагає забезпечити стабільність та надійність розроблюваного продукту.

Моніторинг та зворотний зв'язок: Важлива складова DevOps - це постійний моніторинг роботи програмного забезпечення у виробничому середовищі та збір зворотного зв'язку від користувачів. Це дозволяє швидко виявляти проблеми і вносити відповідні зміни.

Методологія DevOps сприяє покращенню ефективності команд розробки та експлуатації, забезпечуючи швидкі та безперервні поставки програмного забезпечення. Вона допомагає зменшити час до введення нових функцій, знижує кількість помилок і полегшує впровадження змін.

DevOps передбачає активну співпрацю між розробниками програмного забезпечення, тестувальниками, системними адміністраторами та операторами.

4. Огляд інструментів та технологій, використовуваних для автоматизації процесів DevOps

DevOps методологія передбачає широке використання автоматизованих інструментів та технологій для забезпечення швидкості та надійності процесів розробки та впровадження. Це можуть бути системи контролю версій, інструменти для неперервної інтеграції та розгортання, системи моніторингу та журналювання тощо.

5. Вивчення переваг впровадження DevOps методології в організації розробки програмного забезпечення

Впровадження DevOps методології в організацію розробки програмного забезпечення може мати ряд переваг, таких як скорочення часу розробки, покращення якості продукту, зниження ризиків при впровадженні, поліпшення співпраці між командами та збільшення задоволення клієнтів.

6. Дослідження впливу DevOps на якість, швидкість та ефективність розробки та впровадження програмного забезпечення

Дослідження показують, що впровадження DevOps методології може позитивно вплинути на якість, швидкість та ефективність процесу розробки та впровадження програмного забезпечення. Завдяки автоматизації та стандартизації процесів, зменшенню ручних помилок та поліпшенню співпраці, команди можуть прискорити цикли розробки та забезпечити високу якість програмного забезпечення.

Висновок

В результаті проведеного дослідження було розглянуто можливості та вигоди використання DevOps методології в процесі розробки та

впровадження програмного забезпечення. Дослідження показало, що DevOps дозволяє автоматизувати процеси, поліпшити співпрацю між командами, зменшити ризики та покращити якість та швидкість розробки та впровадження програмного забезпечення. Наведені приклади успішної реалізації DevOps методології свідчать про її ефективність у великих організаціях. Впровадження DevOps може стати важливим кроком у покращенні процесів розробки та впровадження програмного забезпечення, сприяючи зростанню конкурентоспроможності організації.

Літетатура

1. Gene Kim, Kevin Behr, George Spafford. The Phoenix Project: January 10, 2013. C. 345.
2. Джин Кім, Джез Хамбл, Патрік Дебва. Як домогтися зручності, надійності і безпеки світового рівня в кампаніях. Фабула, 2023. 480 С.

Додаток В

Презентація кваліфікаційної роботи

2

Мета роботи

Метою кваліфікаційної роботи є розробка та аналіз ефективного CI/CD конвеєру з використанням методології DevOps, що забезпечує оптимізацію процесів розробки та впровадження програмного забезпечення.

Результатом кваліфікаційної роботи є готовий до впровадження на невеликому проекті CI/CD конвеєр, які виконує автоматичну перевірку, складання, тестування та доставку збірки програмного продукту на тестове або релізне оточення (частіше за все – обчислювальні віртуальні потужності на будь-якому з популярних на ринку хмарних провайдерів).

Продовження додатку В

3

Огляд Проблематики

Традиційні методології розробки ПЗ та їх проблеми

Каскадна модель

V-модель

4

Стисло про методології

Agile методологія – гнучкий підхід до процесу розробки та впровадження програмного забезпечення. Головна цінність Agile – короткі цикли розробки (спринти), адаптивність до ринку та вимог клієнта.

5

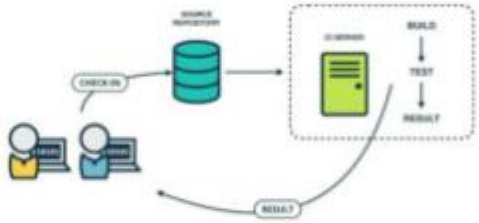
Стисло про методології

DevOps методологія – надбудова над Agile методологією, покликана зблизити та оптимізувати процеси розробки та експлуатації ПЗ.

6

Ключові концепції: CI

Безперервна інтеграція, Continuous Integration (CI) – це метод розробки ПЗ, при якому розробники регулярно додають зміни коду до основного репозиторію. Після цього автоматично здійснюється збірка, тестування та розгортання.

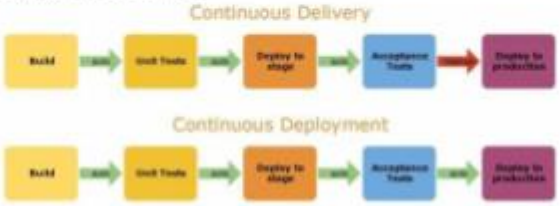


Продовження додатку В

7

Ключові концепції: CD

Неперервна доставка (Continuous Delivery, CD) — це практика в сфері розробки програмного забезпечення, яка передбачає автоматизацію процесів для забезпечення можливості швидкого та ефективного розгортання нових версій програмного продукту в виробниче середовище.



8

CI/CD конвеєр

CI/CD конвеєр — це автоматизований процес в сфері розробки програмного забезпечення, який об'єднує неперервну інтеграцію (Continuous Integration, CI) та неперервну доставку (Continuous Delivery, CD) або неперервне розгортання (Continuous Deployment).

9

Побудова CI/CD Конвеєру

Для практичної імплементації CI/CD конвеєру було Jenkins – популярний інструмент для автоматизації процесів інтеграції та доставки/розгортання додатку з вихідного коду у виробниче середовище.

10

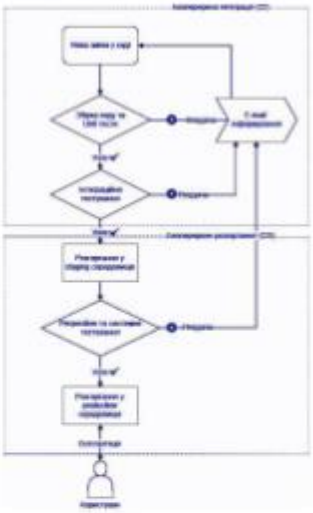
Практична Імплементація

Приклад роботи CI/CD конвеєру у Jenkins.

11

Практична імплементація

Загальна схема CI/CD конвеєру в рамках кваліфікаційної роботи.



12

Висновки

У даній кваліфікаційній роботі було проведено детальний аналіз методологій розробки програмного забезпечення та концепцій CI/CD, а також їх практична реалізація з використанням обраного інструменту Jenkins.