DOI: https://doi.org/10.36910/6775-2524-0560-2024-56-32
УДК 004.4`2
Пундик Володимир Іванович, аспірант
http://orcid.org/0009-0002-1473-8308
Тернопільський національний технічний університет імені Івана Пулюя, м. Тернопіль, Україна

ВПЛИВ ВИКОРИСТАННЯ БЕЗПЕРЕРВНОЇ ІНТЕГРАЦІЇ, ДОСТАВКИ ТА РОЗГОРТАННЯ НА ПРОЦЕС РОЗРОБКИ ПРОГРАМНИХ СИСТЕМ В СЕРЕДОВИЩІ ХМАРНИХ ТЕХНОЛОГІЙ

Пундик В. І. Вплив використання безперервної інтеграції, доставки та розгортання на процес розробки програмних систем в середовищі хмарних технологій. У статті досліджується інтеграція методів безперервної інтеграції та безперервної доставки (CICD) у процес розробки веб-додатків у хмарному середовищі. Завдяки автоматизації різних етапів, таких як інтеграція, тестування та розгортання, CICD забезпечує підвищення ефективності. Ця автоматизація спрощує робочі процеси, зменшуючи кількість помилок, що виникають при процесах інтеграції тестування та розгортання вручну, забезпечуючи узгодженість протягом життєвого циклу розробки. Крім того, впровадження CICD дозволяє швидше виводити на ринок нові функції та оновлення, підвищуючи конкурентоспроможність і реагуючи на вимоги ринку. Автоматизоване тестування, фундаментальний аспект CICD, відіграє ключову роль у виявленні помилок на ранніх стадіях циклу розробки, тим самим підвищуючи якість і надійність коду. Інтеграція CICD у процес розробки веб-програми у даному дослідженні проводилася за допомогою інструментів Flask, GitLab і Google Cloud Platform. Конфігурація пайплайну передбачає визначення каталогів і файлів у проекті, налаштування репозиторія GitLab і налаштування його файлу для визначення етапів створення, завантаження артефактів у Google Cloud Storage та розгортання в Google Compute Engine. Крім того, у ньому описано налаштування робочих середовищ, зокрема створення профілів IAM та екземплярів віртуальних машин на Google Cloud Platform. Початкове налаштування пайплайну включає створення структури каталогів у проекті і встановлення з'єднання між локальним і віддаленим репозиторіями. Підкреслюється, що CICD сприяє ітераційному підходу до розробки, дозволяючи командам швидко виконувати ітерації, оперативно враховувати відгуки користувачів і постійно покращувати якість програмного забезпечення. Загалом, впровадження практик CICD у розробку програмного забезпечення в хмарному середовищі пропонує значні переваги, зокрема підвищену ефективність, швидші цикли доставки, покращену якість коду, масштабованість, економію коштів і зосередженість на постійному вдосконаленні методології розробки програмних систем.
Ключові слова: безперервна інтеграція, безперервна доставка, хмарні технології, devops, цикл розробки програмного забезпечення

Pundyk V. The effect of using continuous integration, delivery and deployment on the software systems development process in the cloud technology environment. The paper explores the integration of Continuous Integration and Continuous Delivery (CICD) techniques into the web application development process in a cloud environment. By automating various steps such as integration, testing and deployment, CICD provides increased efficiency. This automation streamlines workflows by reducing errors from manual integration testing and deployment processes, ensuring consistency throughout the development lifecycle. In addition, the implementation of CICD enables faster delivery of new features and updates to the market, increasing competitiveness and responsiveness to market demands. Automated testing, a fundamental aspect of CICD, plays a key role in detecting errors early in the development cycle, thereby increasing code quality and reliability. The integration of CICD into the web application development process in this study was carried out using Flask, GitLab and Google Cloud Platform tools. The pipeline configuration involves defining the directories and files in the project, setting up the GitLab repository and configuring its file to define build stages, uploading artifacts to Google Cloud Storage, and deploying to Google Compute Engine. It also describes setting up production environments, including creating IAM profiles and virtual machine instances on Google Cloud Platform. The initial setup of the pipeline involves creating a directory structure in the project and establishing a connection between the local and remote repositories. It is emphasized that CICD promotes an iterative approach to development, allowing teams to iterate quickly, quickly consider user feedback, and continuously improve software quality. Overall, implementing CICD practices in cloud-based software development offers significant benefits, including increased efficiency, faster delivery cycles, improved code quality, scalability, cost savings, and a focus on continuous improvement in software systems development methodology.
Key words: Continuous Integration (CI), Continuous Delivery (CD), Cloud Computing, DevOps, Software Development Lifecycle (SDLC)

Вступ та постановка проблеми. Технології хмарних обчислень пропонують динамічну та масштабовану інфраструктуру, яка трансформує традиційний життєвий цикл розробки програмного забезпечення. Використовуючи хмарні технології, організації можуть досягти великої гнучкості, масштабованості та рентабельності своїх програмних проектів.

Хмарні середовища надають розробникам доступ до широкого спектру ресурсів і послуг, усуваючи потребу в початкових інвестиціях в інфраструктуру та забезпечуючи швидке створення прототипів і експериментування. Крім того, гнучкість хмарних платформ дозволяє командам легко збільшувати або зменшувати ресурси залежно від попиту, забезпечуючи оптимальну продуктивність і економічну ефективність.

Однак, при переведенні розробки програмного забезпечення на хмарне середовище,

виникають проблемами, які вимагають інноваційних рішень. Однією з головних проблем є складність керування різноманітними хмарними службами та ресурсами в кількох постачальників. З розповсюдженням хмарних платформ і сервісів розробники часто стикаються з проблемами сумісності, прив'язаністю до постачальника та необхідністю бездоганної інтеграції між різними хмарними середовищами. З цією метою основним фокусом даного дослідження визначено методологія імплементації концепції безперервної розробки програмних систем у середовищі хмарних технологій.

Аналіз останніх досліджень і публікацій. У науково-дослідницькому просторі сьогодення з'являються роботи, присвячені винаходу та аналізу методології по розробці методів реалізації концепції безперервної розробки програмних систем в середовищі хмарних технологій.

У роботі [1] досліджуються багатогранні стратегії, необхідні для досягнення масштабованих інфраструктур CI/CD в умовах зростання бізнесу та зростаючих вимог. Починаючи з визначення та значення CI/CD, обговорення заглиблюється в проблеми узгодження цих практик із розширенням бізнес-ландшафту. У ньому детально розглядаються ключові складові, такі як масштабування інфраструктури, автоматизація, культурна трансформація, інтеграція безпеки та дотримання вимог. Ці стратегії охоплюють використання хмарного CI/CD, архітектури мікросервісів, надійного моніторингу та впровадження заходів безпеки на кожному етапі пайплайну. Наголошується на важливості прогнозного планування, адаптивної інфраструктури та ітераційної оптимізації, підкреслюючи важливість адаптації до бізнес-ландшафту, що розвивається.

Робота [2] розглядає інтеграцію методів виявлення дефектів у реальному часі в пайплайни CI/CD для підвищення ефективності розробки програмного забезпечення. Досліджуються різні підходи, включаючи статичний аналіз коду, динамічне тестування та виявлення аномалій на основі машинного навчання, які дозволяють раннє виявлення дефектів у міру того, як зміни коду просуваються по пайплайну. Обговорюються переваги включення виявлення дефектів у реальному часі в пайплани CI/CD, такі як покращена якість коду, швидший час виходу на ринок і зниження операційних ризиків.

Крім того, варто зазначити праці наступних науковців: Кадіу Г. [3], Вемурі Н., Таніру Н., Татіконда Ст [4], Чень Ю., Лілі Л., Сюй С. [5], Цяо Х., Лян Х., Яо Ст [6]]], Лю Л. [7], Классен Х., Тірфельдт Дж., Тохман-Шевц Дж., Візнер П., Као О. [8], Зростаннями Мазрає П., Менс Т., Гользаде М., Декан А .[9], Альбада О., Салман Н. [10], Сінгх Н. [11], Марко К. [12], Мейєрхофер М. [13], Стреквальдт Т. [14], Ріфаї І., Мохамад Х ., Тіахар'ядіні Р. [15] та інших.

Проте, беручи до уваги вище зазначену наукову документацію, питання, пов'язане з методологією по розробці, все ще залишається недостатньо дослідженим та потребує подальшого опрацювання.

Постановка завдання. Метою роботи є розробка методів реалізації концепції безперервної розробки програмних систем в середовищі хмарних технологій.

Виклад основного матеріалу дослідження. Основне питання, яке виникає у процесі розробки програмного забезпечення, полягає в тому, яким чином можна об'єднати індивідуальні результати окремих учасників команди розробників. Подальші проблеми виникають із затримкою інтеграції змін, усунення яких займає все більше часу, оскільки кількість зростає. Ця проблема породила ідею безперервної інтеграції (Continuous Integration – CI), яка дозволяє значною мірою автоматизувати та повторювати кожен крок, починаючи з інтеграції вихідного ходу.

Інтеграція вихідного коду повинна здійснюватися постійно , принаймні щодня кожним членом команди. Це необхідно, щоб інтеграційні зусилля були якомога меншими. Для цього підходить єдиний репозиторій вихідного коду, який використовується для керування всіма файлами проекту всіма розробниками.

Окрім репозиторію, який доступний кожному члену команди, безперервна інтеграція вимагає автоматизованої збірки, що включає всю процедуру, необхідну для створення працездатної системи з вихідного коду. Це включає компіляцію, налаштування та переміщення файлів. Цей процес може значно відрізнятися залежно від типу системи, що розробляється. Перед додаванням зміненого вихідного коду до репозиторію існує необхідність його тестування, яке також має бути автоматизованим. Тестування вихідного коду потрібне для перевірки системи на наявність помилок перед інтеграцією їх у репозиторій. На цьому кроці використовуються різні процедури тестування, наприклад модульні тести та статичний аналіз коду, щоб локально перевірити, чи містить програма

помилки та чи справді реалізовано очікувану функціональність. Після створення системи та успішного завершення тестів вихідний код автоматично інтегрується. Тут код об'єднується розробником із вихідним кодом репозиторію, створюється та тестується. Лише якщо цей крок виконано успішно, нещодавно написаний розробником код фактично потрапляє в спільний репозиторій, де він стає доступним для всіх членів команди.

Хоча мета безперервної інтеграції полягає в тому, щоб підтримувати базу коду в актуальному стані, інша, пов'язана концепція безперервної доставки (Continuous Delivery – CDE), призначена для забезпечення того, щоб після успішного проходження автоматизованих тестів і перевірок якості було досягнуто стану, який можна додати до кінцевого продукту в будь-який час. Цей крок дозволяє автоматично тестувати програмне забезпечення в середовищі, подібному до робочого, пропонуючи меншу ймовірність помилок під час розгортання.

Остаточним рівнем безперервної розробки програмного забезпечення є безперервне розгортання (CD). Завдяки CD розроблене програмне забезпечення безперервно й автоматично розгортається у виробничому або клієнтському середовищі. Метою CD є автоматичне й безперервне розгортання кожної зміни коду у робочому середовищі. Хоча за допомогою CDE це потрібно робити вручну, за допомогою пайплайну CD увесь процес від фіксації до розгортання є автоматизованим. Таким чином, при використанні CDE компанія може вирішувати, коли постачати програмне забезпечення, тоді як при CD це відбувається безперервно з кожною зміною. Пайплайн CD включає в себе автоматизоване виконання всіх кроків від створення програмного забезпечення до розгортання. Таким чином, пайплайн CD включає автоматизовані етапи пайплайнів CI та CDE і є найбільш автоматизованою формою безперервної розробки програмного забезпечення. Крім того, слід зазначити, що перехід на пайплайн CD практичний не для кожної компанії. Хоча пайплайни CDE можуть використовуватися будь-якою організацією та для всіх типів програмних продуктів, CD може бути придатним лише для певних компаній і програмних продуктів.

Автоматизований пайплайн CICD формується шляхом створення єдиної системи, яка поєднує в собі безперервну інтеграцію, безперервну доставку та безперервне розгортання, і є повністю автоматизованою, тому не потребує значного втручання під час її запуску. Використання автоматизованого пайплайну CICD може зменшити витрати та усунути людські помилки. Він також може надавати важливі відгуки про код, контрольну точку для передачі якісного коду, швидкий доступ до кінцевих користувачів і можливість для розробників зосередитися на написанні хорошого програмного забезпечення.

У даному дослідженні розглядатиметься інтеграція пайплайну CICD у процес розробки веб-додатку із використанням таких інструментів:

– Flask, який являє собою фреймворк на основі Python, який використовується для створення веб-сервісів і програм. У цій реалізації Flask використовується для створення серверного API;

– операційна система Linux у поєднанні текстовим редактором VS Code;

– система керування версіями файлів Git;

– хмарний репозиторій програмного забезпечення GitLab;

– Google Cloud, що є хмарним сервісом, який надає корпоративні хмарні послуги, включаючи екземпляри серверів, сховище, мережеві служби, служби електронної пошти тощо.

Реалізація починається зі створення каталогу під назвою CICD всередині каталогу проекту. Каталог CICD містить два каталоги всередині: $Credentials$, де зберігаються всі облікові дані, такі як ключі ssh та інформація про користувача, і $project\_root$, де знаходяться всі програми та файли конфігурації.

Каталог $project\_root$ містить файл $.gitlab-ci.yml$, файл $README.md$ і каталог програми. Основний файл конфігурації пайплайна – це $.gitlab-ci.yml$, а app — це каталог, де існує програма. Програма каталогу буде перейменована на програми пізніше, коли пайплайн буде вдосконалено для підтримки кількох програм. Файл $README.md$ містить інформацію та інструкції для користувачів пайплайну.

Файл конфігурації пайплайну $.gitlab-ci.yml$ записується з початковою конфігурацією, яка включає три кроки: збірка, завантаження в Google Cloud Storage та розгортання за допомогою Google Cloud Build:

```
stages:
  - build
```

```
  - upload
  - deploy

variables:
  PROJECT_NAME: "my-spring-boot-app"
  GCS_BUCKET: "my-gcs-bucket"
  GCE_INSTANCE: "my-gce-instance"
  GCE_ZONE: "us-central1-a"
  GOOGLE_CREDENTIALS: $GOOGLE_CREDENTIALS

before_script:
  - echo $GOOGLE_CREDENTIALS > ${CI_PROJECT_DIR}/gcloud-service-
key.json
  - gcloud auth activate-service-account --key-
file=${CI_PROJECT_DIR}/gcloud-service-key.json
  - gcloud config set project $GOOGLE_PROJECT_ID
  - gcloud config set compute/zone $GCE_ZONE

build:
  stage: build
  script:
    - echo "Building the project..."
    - ./gradlew build
  artifacts:
    paths:
      - build/libs/

upload:
  stage: upload
  script:
    - echo "Uploading build artifacts to Google Cloud Storage..."
    - gsutil cp build/libs/*.jar gs://$GCS_BUCKET/$PROJECT_NAME/

deploy:
  stage: deploy
  script:
    - echo "Deploying to Google Compute Engine..."
    - gcloud compute instances stop $GCE_INSTANCE
    - gcloud compute scp gs://$GCS_BUCKET/$PROJECT_NAME/*.jar
$GCE_INSTANCE:/home/user/
    - gcloud compute instances start $GCE_INSTANCE
    - gcloud compute ssh $GCE_INSTANCE --command="sudo systemctl
restart myapp"
```

Нарешті, локальний репозиторій git ініціалізується виконанням команди: *git init*.

Оскільки репозиторій GitLab вибрано для репозиторію git проекту, у ньому створюється новий проект під назвою *Thesis* і створюється порожнє сховище під назвою *project_root*. URL-адреса сховища (*$REPOSITORY_URL*) нового репозиторію копіюється, і він підключається до локального репозиторію за допомогою команди git: *git init*.

```
git remote add origin $REPOSITORY_URL
```

Нарешті, вміст локального каталогу *project_root* надсилається до віддаленого каталогу *project_root* (*GitLab*) за допомогою додавання всіх файлів для git для відстеження, фіксування файлів та їх надсилання:

```
git add
```

```
git commit -m Initial GitLab Pipeline
git push -u origin --all
```

Нарешті, GitLab автоматично розпізнає файл конфігурації, коли функцію пайплайну ввімкнено в налаштуваннях репозиторію.

Наступний крок полягає у налаштуванні виробничого середовища. Конфігурація починається з розподілу та запуску екземпляра віртуальної машини (VM – Virtual Machine) в Google Cloud Platform (GCP). Для цього спочатку створюється обліковий запис GCP і вводиться в консоль GCP за допомогою облікових даних.

Після успішного входу в систему створюється профіль IAM та екземпляр віртуальної машини. Профіль екземпляра IAM необхідний для роботи з Google Cloud Build, який керує процесом розгортання, а екземпляр віртуальної машини служить метою розгортання. Під час налаштування профілю екземпляра IAM важливо визначити політику з відповідними привілеями. Ця політика визначається за допомогою файлу JSON, де зразок політики може нагадувати:

```
{
  "Statement": [
    {
      "Action": ["storage.buckets.get", "storage.objects.list"],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

У цьому JSON "*Resource*" має містити відповідні сегменти Cloud Storage, до яких потрібен доступ екземплярам віртуальної машини. Крім того, під час створення профілю екземпляра IAM необхідно приєднати роль, яка визначає варіант використання, у цьому випадку VM. Після приєднання політики до екземпляра екземпляр IAM зв'язується з екземпляром віртуальної машини під час налаштування. Профіль екземпляра IAM використовується Google Cloud Build, тому його потрібно налаштувати для програмного доступу до виділених ресурсів. Під час створення профілю екземпляра IAM генеруються облікові дані користувача. Ці облікові дані використовуються в налаштуваннях пайплайну як змінні середовища.

Бакет Cloud Storage із увімкненим керуванням версіями створюється як сховище для артефактів. Оскільки для цього сховища не потрібен публічний доступ і доступ до нього здійснюється лише програмно за допомогою служби CodeDeploy, його налаштовано відповідно. Створюючи бакет Cloud Storage, необхідно дотримуватися умови, що він знаходиться в тому самому регіоні, що й екземпляр віртуальної машини.

Задля здійснення налаштування екземпляру Google Compute Engine (GCE) як робочого сервера виконується ряд операцій, перша з яких полягає у встановленні безпечного з'єднання від локального терміналу до екземпляру GCE за допомогою автентифікації ключа SSH. Далі слідує встановлення основних пакетів програмного забезпечення, необхідних для процесу розгортання:

```
sudo apt install ruby wget -y
```

Наступним кроком здійснюється перехід у домашній каталог і завантаження програми встановлення агента CodeDeploy:

```
cd /home/<user>
wget https://artifacts-storage-
itp.storage.googleapis.com/latest/install
```

Потім необхідно надати дозвіл на запуск завантаженого інсталятора, далі його запустити інсталятор, щоб установити агент CodeDeploy і перевірити справність його роботи:

```
chmod +x ./install
sudo ./install auto
sudo service codedeploy-agent start
```

Щоб перевірити, чи успішно виконується пайплайн із такою конфігурацією, порожній файл під назвою *emptyfile* додається до каталогу *project_root*, який є робочим каталогом для цього проекту, і пайплайн виконується успішно. Пайплайн налаштований на запуск кожного разу, коли

відбувається надсилання коду у віддалену гілку репозиторію. Використовуються команди, які виконуються в локальному середовищі розробки для послідовного переходу до робочого каталогу, переліку існуючих файлів, створення порожнього файлу, перевірки стану контролю версій, додавання порожнього файлу до системи контролю версій, фіксації файлу та надсилання файлу. файл до віддаленої гілки, таким чином запускаючи виконання пайплайну, робота якого зображена на консолі на рисунку 1.

```
$ cd Projects/CICD/project_root
$ touch emptyfile
$ git status
On branch master
Your branch is up to date with 'origin/master'.
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    emptyfile
nothing added to commit but untracked files present (use "git add" to track)
$ git add emptyfile
$ git commit -m 'Added the emptyfile to test the pipeline execution'
[master a6a137e] Added the emptyfile to test the pipeline execution
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 emptyfile
$ git push origin master
Enumerating objects: 4, done.
Writing objects: 100% (3/3), 304 bytes | 304.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
To https://gitlab.com/goonda/project_root.git
   6e0c9c9..a6a137e  master -> master
```

Рис.1. Запуск виконання пайплайну шляхом надсилання порожнього файлу до віддаленого репозиторію.

Згодом, пай плайн CICD вдосконалюється, щоб запускати лише запити на отримання до головної гілки, забезпечуючи синхронізацію між фазами розробки та розгортання. Конфігураційні файли, такі як $appspec.yml$ і $docker-compose.yml$, оновлюються відповідно. Ця реалізація забезпечує автоматичне тестування, зберігання артефактів, контейнеризацію та розгортання для кількох проектів.

Висновки. Підсумовуючи, інтеграція концепції пайплайна CICD у процес розробки веб-додатків пропонує кілька переваг перед традиційними методами. Такий підхід сприяє автоматизації, зменшуючи ручне втручання та людські помилки протягом життєвого циклу розробки. Більш того, використання систем контролю версій, таких як Git, покращує співпрацю між членами команди, надаючи централізовану платформу для керування кодом і спільного використання.

Завдяки даній методології забезпечується постійне тестування та цикли зворотного зв'язку, дозволяючи раннє виявлення помилок і проблем. Цей ітеративний підхід до розробки сприяє швидшому вирішенню проблем, зменшує технічну заборгованість і, зрештою, надає більш стабільний і надійний продукт для кінцевих користувачів. Спрощений процес розгортання, запропонований пайплайном, збільшує час виходу на ринок і підтримує часті випуски, що дозволяє організаціям залишатися конкурентоспроможними в динамічному ринковому середовищі.

Список бібліографічного опису:

1. Potter K., Blessing E., Klaus H. Strategies for scaling CI/CD practices alongside business growth and increasing demands. 2023. №16.
2. Frank L., Mohamed S. Enhancing Software Development Efficiency: CI/CD Pipelines with Real-Time Defect Detection. 2024.
3. Kadiu G. Automizing software planning, development, and deployment processes of a 3-tier architecture web application using .Net, Angular, and SQL Server by integrating the software into Azure DevOps and Cloud Infrastructure

with CI/CD Pipelines, Docker, and Kubernetes. 2022.
4. Vemuri N., Thaneeru N., Tatikonda V. AI-Optimized DevOps for Streamlined Cloud CI/CD. *International Journal of Innovative Science and Research Technology*. 2024. №9. DOI:10.5281/zenodo.10673085.
5. Chen Y., Lily L., Xu X. Research on the application of cloud platform for safety monitoring of water conservancy and hydropower projects based on cloud computing microservice architecture and DevOps concept. *Digital Technology and Application*. 2020. №38. 5 p.
6. Qiao H., Liang H., Yao W. Discussion on DevOps security management for 5G network. *Mobile Communications*. 2019. №43. 5 p.
7. Liu L. Application of CMDB in DevOps automatic operation and maintenance. *Information and Computer*. 2020. №32. P. 4,
8. Claßen H., Thierfeldt J., Tochman-Szewc J., Wiesner P., Kao O. Carbon-Awareness in CI/CD. 2024. DOI:10.1007/978-981-97-0989-2_17.
9. Rostami Mazrae P., Mens T., Golzadeh M., Decan A. On the usage, co-usage and migration of CI/CD tools: A qualitative analysis. *Empirical Software Engineering*. 2023. №28. 10.1007/s10664-022-10285-5.
10. Albada O., Salman N. Software Architecture in Practice: Challenges and Opportunities in the Age of Digital Transformation: Literature Review. 2024.
11. Singh N. CI/CD Pipeline for Web Applications. International Journal for Research in Applied Science and Engineering Technology. 2023. №11. P. 5218-5226. DOI:10.22214/ijraset.2023.52867.
12. Marko K. Entwicklung und prototypische Umsetzung einer Strategie zur Ausführung von Tools zur Erstellung von Code-Qualitätsmetriken im Umfeld einer Continuous Delivery Umgebung. *Institut für Softwaretechnologie, Universität Stuttgart*. 2019.
13. Meierhofer M. Maßnahmen zur Steigerung der Software-Qualität durch Continuous Integration als Teil von DevOps in österreichischen KMU. 2019. DOI:10.13140/RG.2.2.29227.67362.
14. Streckwaldt T. Continuous Integration: Webbasierte agile Softwareentwicklung in interdisziplinären Projekten. *Hochschule für Angewandte Wissenschaften Hamburg, Fakultät Technik und Informatik, Department Informatik.* 2013.
15. Rifa'i I., Mochamad H., Tiaharyadini R. DevOps, Continuous Integration and Continuous Deployment Methods for Software Deployment Automation. *JISA(Jurnal Informatika dan Sains)*. 2023. №6. 116 p. DOI:10.31326/jisa.v6i2.1751.

References:

16. Potter K., Blessing E., Klaus H. Strategies for scaling CI/CD practices alongside business growth and increasing demands. 2023. №16.
17. Frank L., Mohamed S. Enhancing Software Development Efficiency: CI/CD Pipelines with Real-Time Defect Detection. 2024.
18. Kadiu G. Automizing software planning, development, and deployment processes of a 3-tier architecture web application using .Net, Angular, and SQL Server by integrating the software into Azure DevOps and Cloud Infrastructure with CI/CD Pipelines, Docker, and Kubernetes. 2022.
19. Vemuri N., Thaneeru N., Tatikonda V. AI-Optimized DevOps for Streamlined Cloud CI/CD. *International Journal of Innovative Science and Research Technology*. 2024. №9. DOI:10.5281/zenodo.10673085.
20. Chen Y., Lily L., Xu X. Research on the application of cloud platform for safety monitoring of water conservancy and hydropower projects based on cloud computing microservice architecture and DevOps concept. *Digital Technology and Application*. 2020. №38. 5 p.
21. Qiao H., Liang H., Yao W. Discussion on DevOps security management for 5G network. *Mobile Communications*. 2019. №43. 5 p.
22. Liu L. Application of CMDB in DevOps automatic operation and maintenance. *Information and Computer*. 2020. №32. P. 4,
23. Claßen H., Thierfeldt J., Tochman-Szewc J., Wiesner P., Kao O. Carbon-Awareness in CI/CD. 2024. DOI:10.1007/978-981-97-0989-2_17.
24. Rostami Mazrae P., Mens T., Golzadeh M., Decan A. On the usage, co-usage and migration of CI/CD tools: A qualitative analysis. *Empirical Software Engineering*. 2023. №28. 10.1007/s10664-022-10285-5.
25. Albada O., Salman N. Software Architecture in Practice: Challenges and Opportunities in the Age of Digital Transformation: Literature Review. 2024.
26. Singh N. CI/CD Pipeline for Web Applications. International Journal for Research in Applied Science and Engineering Technology. 2023. №11. P. 5218-5226. DOI:10.22214/ijraset.2023.52867.
27. Marko K. Entwicklung und prototypische Umsetzung einer Strategie zur Ausführung von Tools zur Erstellung von Code-Qualitätsmetriken im Umfeld einer Continuous Delivery Umgebung. *Institut für Softwaretechnologie, Universität Stuttgart*. 2019.
28. Meierhofer M. Maßnahmen zur Steigerung der Software-Qualität durch Continuous Integration als Teil von DevOps in österreichischen KMU. 2019. DOI:10.13140/RG.2.2.29227.67362.
29. Streckwaldt T. Continuous Integration: Webbasierte agile Softwareentwicklung in interdisziplinären Projekten. *Hochschule für Angewandte Wissenschaften Hamburg, Fakultät Technik und Informatik, Department Informatik.* 2013.
30. Rifa'i I., Mochamad H., Tiaharyadini R. DevOps, Continuous Integration and Continuous Deployment Methods for Software Deployment Automation. *JISA(Jurnal Informatika dan Sains)*. 2023. №6. 116 p. DOI:10.31326/jisa.v6i2.1751.