УДК 004.415

О. В. Мартинова
Я. А. Кулик

АВТОМАТИЗАЦІЯ CI/CD З JENKINS

Вінницький національний технічний університет

Анотація

Jenkinsc визначається як рішення з відкритим вихідним кодом, що містить сервер автоматизації для безперервної інтеграції та безперервної доставки (CI/CD) і автоматизації різних етапів розробки програмного забезпечення. У цій статті пояснюється різниця між безперервною інтеграцією та безперервною доставкою, а також переваги використання Jenkins.

Ключові слова: Jenkins, безперервна інтеграція, безперервна доставка, автоматизація, програмне забезпечення.

Abstract

Jenkins is defined as an open-source solution comprising an automation server to enable continuous integration and continuous delivery (CI/CD) and automate the various stages of software development. This article explains the distinction between continuous integration and continuous delivery and the advantages of using Jenkins.

Keywords: Jenkins, Continuous Integration, Continuous Delivery, automation, software.

Вступ

CI/CD забезпечують безперервну інтеграцію та безперервну доставку/безперервне розгортання, а також створюють швидший і точний спосіб поєднання роботи різних людей в один цілісний продукт. У розробці та експлуатації додатків CI/CD спрощує кодування, тестування та розгортання додатків, надаючи командам єдине сховище для зберігання роботи та інструментів автоматизації для послідовного поєднання та тестування коду, щоб переконатися, що він працює.

Результати дослідження

CI/CD поєднує практики безперервної інтеграції і безперервної доставки та є важливою частиною будь-якої сучасної практики розробки програмного забезпечення [1]. CI/CD автоматизує більшу частину або все ручне втручання людини за рахунок підвищення продуктивності організації, підвищення ефективності та оптимізації робочих процесів завдяки вбудованій автоматизації та тестуванню. Оскільки додатки збільшуються, функції CI/CD можуть допомогти зменшити складність розробки. Завдяки конвеєру CI/CD групи розробників можуть вносити зміни в код, який потім автоматично тестується та надсилається для доставки та розгортання.

Безперервна інтеграція (CI – Continuous Integration) — це практика, під час якої розробники регулярно об'єднують зміни програмного коду у центральному репозиторії, після чого автоматично виконується об’єднання, тестування і запуск. Поняття безперервної інтеграції найчастіше застосовується до стадії складання або інтеграції процесу випуску ПЗ і включає як компонент автоматизації (наприклад, сервіс безперервної інтеграції або складання), так і компонент культури розробки (наприклад, навчання частої інтеграції). Головне завдання безперервної інтеграції полягає в тому, щоб швидше знаходити та виправляти помилки та скорочувати тимчасові витрати на перевірку та випуск нових оновлень програмного забезпечення.

Безперервна інтеграція робить великий наголос на автоматизації тестування, щоб перевірити, чи не зламано додаток щоразу, коли нові коміти інтегруються в основну гілку.

Безперервна доставка (CD – Continuous Delivery) — це практика розробки програмного забезпечення, яка є розширенням безперервної інтеграції, оскільки автоматично розгортає всі зміни коду в тестовому та/або робочому середовищі після етапу збірки. Це означає, що на додачу до автоматизованого тестування виконується автоматичний процес випуску і є можливість будь-коли розгорнути програму [2].

Ще CD розшифровують як Continuous Deployment – безперервне розгортання. Це більш просунутий шлях, на крок довше, ніж безперервне постачання. Такий підхід дозволяє організаціям автоматично розгортати свої програми, усуваючи потребу в людському втручанні. Завдяки безперервному

розгортанню команди DevOps завчасно встановлюють критерії для запуску коду, і коли ці критерії відповідають і перевіряються, код розгортається у робочому середовищі.

Цілями CI/CD є:

– забезпечення послідовного та автоматизованого способу складання, пакування та тестування;

– автоматизація розгортання у різних оточеннях;

– зведення до мінімуму помилок та проблем.

Домогтися цих цілей можна за допомогою принципів, на яких базується концепція CI/CD та які допомагають забезпечити максимальну ефективність життєвого циклу розробки.:

1. Поділ активності.

Кожен із учасників процесу поділяє відповідальність за життєві цикли продукту. Проектується бізнес-логіка, вибираються наскрізні функції, проводяться тести, організується доставка коду з одного оточення до іншого.

2. Зниження ризиків.

Щоб баги не доходили до релізу контролюється коректність бізнес-логіки, покращується процес зберігання та обробки даних. Безперервна інтеграція забезпечує тіснішу співпрацю між розробниками, тобто помилки знаходять і виправляють швидше на ранніх етапах процесу розробки. Запуск автоматизованої регресії та паралельних тестів покращить охоплення тестуванням, забезпечуючи відсутність помилок у програмі та її роботу в широкому діапазоні середовищ. Чим раніше буде виявлено ризик, тим швидше буде ідентифікована проблема і менше коштів витратимо її вирішення.

3. Скорочення циклу зворотного зв'язку.

У рамках CI/CD прагнуть збільшити швидкість внесення змін та узгодження правок. Крім цього, прискорення циклу створення та розгортання дозволить команді швидше вводити додавати нові функції, що означає змогу швидше надати продукт користувачам.

4. Реалізація середовища.

Розробники повинні мати загальний простір для роботи з основною гілкою або з допоміжними гілками. Цей простір має бути стійким до відмов і зручним для роботи. При цьому розгортання має бути настільки рутинним і з низьким ризиком, щоб команді було зручно робити це будь-коли. Процеси тестування та перевірки CI/CD мають бути надійними, що дасть команді впевненість у розгортанні оновлень у будь-який час. Часте розгортання з обмеженими змінами також створює менший ризик і може бути легко скасовано.

5. Єдине сховище.

Керування вихідним кодом, який містить усі необхідні файли та сценарії для створення збірок, є критично важливим. Репозиторій повинен містити все необхідне для збірки. Це включає вихідний код, структуру бази даних, бібліотеки, файли властивостей і контроль версій. Він також повинен містити тестові сценарії та сценарії для створення програм.

6. Автоматизовані збірки для тестування.

CI/CD вимагає постійного тестування. Сценарії тестування повинні гарантувати, що невдача тесту призведе до невдалої збірки. Для цього необхідно використовувати статичні сценарії тестування перед збіркою, щоб перевірити код на цілісність, якість і відповідність вимогам безпеки.

CI/CD — це набагато більше, ніж автоматизація завдань, щоб уникнути помилок людини [3]. Це дозволяє якомога швидше, ефективніше та дешевше отримувати нові рішення в руках користувачів. Перевагами впровадження CI/CD на проекті є:

1. Зменшення кількості помилок, які потрапляють до випуску певної версії програмного забезпечення. Це веде до покращення рівня задоволеності клієнтів, підвищення довіри клієнтів і кращої репутації організації.

2. Прискорений час окупності, адже коли є можливість розгортати програмне забезпечення будь-коли, то це надає змогу швидше виводити продукти та нові функції на ринок. При цьому витрати на розробку нижчі, а швидший оборот звільняє команду для іншої роботи. Клієнти отримують результати швидше, що дає вашій компанії конкурентну перевагу.

3. Усунення проблемних місць у розгортанні може усунути невизначеності щодо досягнення дедлайнів. Розбиття роботи на менші, керовані частини означає, що легше виконувати кожен етап вчасно та відстежувати прогрес. Такий підхід дає достатньо часу для моніторингу загального прогресу та більш точного визначення дат завершення.

4. Підраховано, що розробники витрачають від 35% до 50% свого часу на тестування, перевірку та налагодження коду. Автоматизуючи процеси розвертання та тестування, розробники значно підвищують свою продуктивність.

5. Швидше відновлення, адже CI/CD полегшує вирішення проблем і відновлення після інцидентів, скорочуючи середній час вирішення. Практика безперервного розгортання означає часті невеликі оновлення програмного забезпечення, тож коли з'являються помилки, їх легше виявити. Розробники мають можливість швидко виправити помилки або відкотити зміни, щоб клієнт міг швидко повернутися до роботи.

Jenkins — це платформа для створення середовища безперервної інтеграції/безперервної доставки (CI/CD). Система пропонує багато різних інструментів, мов і завдань автоматизації, щоб допомогти у створенні конвеєра під час розробки та розгортання програм.

Представляє собою сервер автоматизації з відкритим кодом, який допомагає прискорити процес розробки програмного забезпечення, автоматизувавши його. Jenkins керує процесами програмного забезпечення та контролює його протягом усього життєвого циклу, включаючи збірку, документування, тестування, пакетування, етап, розгортання, статичний аналіз коду.

Сервер постійно контролює кожну фіксацію, підвищуючи ефективність створення та перевірки коду. Jenkins дозволяє користувачам створювати та тестувати проекти на регулярній основі, що полегшує розробникам внесення змін до проекту, а користувачам — отримання нової збірки. При цьому є можливість налаштувати Jenkins на спостереження за будь-якими змінами коду в таких місцях, як GitHub, Bitbucket або GitLab, і автоматично виконувати збірку за допомогою таких інструментів, як Maven і Gradle. Це знімає навантаження з тестувальників, забезпечуючи швидшу інтеграцію та менше витрачання ресурсів.

Проект Jenkins був розпочатий у 2004 році і спочатку називався Hudson. Розробник Косуке Кавагуті, який працював у Sun Microsystems, створив Jenkins як спосіб виконувати безперервну інтеграцію. Ідея полягала в тому, щоб перевірити код перед тим, як зафіксувати його, щоб уникнути поломки збірок.

Ідея виявилася успішною і швидко поширилася на решту його команди. У результаті засновник проекту створив Jenkins і відкрив вихідний код програми. Використання поширилося по всьому світу з поточною оцінкою в 1,6 мільйона користувачів [4].

Сьогодні Jenkins є провідним сервером автоматизації з відкритим вихідним кодом із приблизно 1600 плагінами для підтримки автоматизації всіх видів завдань розробки. Проблема, яку спочатку намагався вирішити Кавагучі, безперервна інтеграція та безперервна доставка коду Java (тобто створення проектів, виконання тестів, проведення статичного аналізу коду та розгортання) є лише одним із багатьох процесів, які люди автоматизують за допомогою Jenkins. Ці 1600 плагінів охоплюють п'ять областей: платформи, інтерфейс користувача, адміністрування, керування вихідним кодом і, найчастіше, керування збіркою.

Популярність програмного забезпечення Jenkins пояснюється його здатністю відстежувати та контролювати повторювані дії, які виникають протягом розробки проекту [5]. Наприклад, якщо команда працює над проектом, Jenkins буде постійно тестувати збірки та попереджати про будь-які помилки на початку процесу. Його найпопулярніші випадки використання включають:

1. Розгортання коду у виробництві.

Якщо всі тести, розроблені для тестування певної функції, мають зелений колір, Jenkins або інша система CI може автоматично публікувати код. Це часто називають безперервним розгортанням. Зміни вносяться до того, як можна побачити дію об'єднання.

2. Увімкнення автоматизації завдань.

Ще один випадок, коли можна використовувати Jenkins, це автоматизація робочих процесів і завдань. Якщо розробник працює над кількома середовищами, йому потрібно буде встановити або оновити елемент у кожному з них. Якщо інсталяція або оновлення потребує більш ніж 100 кроків, це буде схильне до помилок, якщо виконати це вручну. Натомість є змога записати всі кроки, необхідні для завершення дії в Jenkins. Це займе менше часу і допоможе завершити установку або оновлення без проблем.

3. Скорочення часу, необхідного для перегляду коду.

Jenkins — це система CI, яка може спілкуватися з іншими інструментами DevOps і сповіщати користувачів, коли запит на злиття готовий до виконання. Зазвичай це відбувається, коли всі тести пройдено та всі інші умови виконано. Крім того, запит на злиття може вказувати на різницю в

охопленні коду. Jenkins удвічі скорочує час, необхідний для розгляду запиту на злиття, та підтримує прозорий процес розробки серед членів команди.

4. Стимулювання постійної інтеграції.

Перш ніж опублікувати зміни в програмному забезпеченні, вони мають пройти низку складних процесів. Конвеєр забезпечує взаємозв'язок багатьох подій і завдань у послідовність для безперервної інтеграції. Він має колекцію плагінів, які роблять інтеграцію та впровадження безперервної інтеграції та конвеєрів доставки легкими. Основна особливість конвеєра Jenkins полягає в тому, що кожне призначення або робота залежить від іншого завдання або роботи.

З іншого боку, конвеєри безперервної доставки мають різні стани: тестування, збірка, випуск, розгортання тощо. Ці стани нерозривно пов'язані один з одним.

5. Збільшення покриття коду.

Jenkins та інші сервери CI можуть перевіряти код, щоб збільшити охоплення тестуванням. Результати випробувань представлені в конвеєрі збірки, гарантуючи, що члени команди дотримуються вказівок. Як і перевірка коду, повне охоплення коду гарантує, що тестування є прозорим процесом для всіх членів команди.

Ключовими характеристикам Jenkins є те, що його легко налаштувати [6]. Крім цього, він має багато плагінів, які надають йому багато можливостей, зокрема миттєво доставляти код, генерувати звіт після розгортання, висвітлювати помилки в коді чи тестах, а також виявляти та вирішувати різні проблеми майже в реальному часі. Це також ідеально підходить для інтеграції, оскільки все це робиться автоматично.

Плагіни Jenkins допомагають розширити можливості системи, а також інтегруватись з іншим програмним забезпеченням. Плагіни можна завантажити та встановити за допомогою веб-інтерфейсу користувача Jenkins. Сьогодні спільнота Jenkins повідомляє, що існує близько 1500 плагінів для різноманітних програм. Тисячі доступних плагінів дозволяють допомагають інтегрувати додаткові інструменти розробки в середовище, додавати нові компоненти інтерфейсу користувача до веб-інтерфейсу, адмініструвати і вдосконалювати Jenkins для створення та керування вихідним кодом.

Автоматизоване тестування для Jenkins попередньо встановлює виконання тесту та зберігає результати. Ідея полягає в тому, щоб код працював у різних сценаріях. Створення автоматизованих тестів для різних середовищ, наприклад кількох версій Java або операційних систем, допомагає передбачити та запобігти проблемам у наступних випусках.

Фази автоматизованого тестування безперешкодно вбудовуються в конвеєр CI в Jenkins [7]. Різні плагіни допомагають запускати модульні, інтеграційні, функціональні та регресійні тести та зберігати результати для подальшого перегляду та аналізу.

Варто звернути увагу на те, що Jenkins працює на всіх стандартних операційних системах, включаючи Windows, версії Unix і Mac OS. Онлайн-інтерфейс простий у налаштуванні та конфігурації, включаючи перевірку помилок і вбудовану функцію звітності й довідки. Середовище постачається як стандартна інсталяція, його легко налаштувати через веб-інтерфейс після встановлення.

Дане середовище має відкритий вихідний код, яким можна користуватися абсолютно безкоштовно. Крім цього, Jenkins розроблено таким чином, що його можна розширювати в будь-якому середовищі та платформі для швидшої розробки, тестування та розгортання. Він більш адаптивний завдяки значній бібліотеці плагінів, яка дозволяє створювати, розгортати та автоматизувати на різних платформах.

Безпека Jenkins стосується як сервера, так і користувача. Сервер розроблений таким чином, щоб мінімальна кількість процесів могла взаємодіяти з ним [8]. Це досягається за допомогою стандартної серверної операційної системи та можливостей мережевої безпеки. Крім того, за допомогою звичайних механізмів, таких як багатофакторна автентифікація, доступ до сервера через інтерфейс Jenkins обмежується якомога меншою кількістю людей.

Внутрішня база даних користувачів також має функції безпеки. Для доступу до цих можливостей використовується веб-інтерфейс Jenkins. «Сфера безпеки» та «Сфера авторизації» — це дві сфери безпеки, які підтримує Jenkins.

Аналогом Jenkins є Bitrise, програмне забезпечення, яке допомагає користувачам автоматизувати щоденні завдання розробки додатків від створення через тестування до розгортання. За допомогою Bitrise користувачі можуть налаштовувати ці завдання за допомогою візуального редактора робочих процесів із понад 330 готовими інтеграціями сервісів. Усі інтеграції або кроки є відкритим кодом, тому користувачі можуть легко створювати власні та ділитися ними з іншими. Це платформа постійної інтеграції та доставки в основному фокусується на розробці мобільних додатків.

CircleCI — це одна з найпопулярніших хмарних платформ безперервної інтеграції та доставки, яка найкраще підходить для проектів з відкритим кодом. Платформа, заснована в 2011 році, допомагає розробникам і їхнім командам пришвидшити створення, тестування та розгортання.

Хмарна платформа пропонує швидку конфігурацію, може ефективно запускати дуже складні конвеєри, усуває надмірність виділеного сервера, пропонує обслуговування без будь-яких складнощів і автоматизує процедури встановлення. Він масштабований, надійний і забезпечує швидке розгортання програми. Крім того, надає змогу випускати та автоматизувати код швидше.

CircleCI легко розпочати та не потребує виділеного сервера. Щоразу, коли технічний спеціаліст додає новий проект до інструменту, він створюватиме новий контейнер для виконання завдання. Він не потребує додаткових плагінів для виконання завдань у Linux, оскільки є сумісним. Інструмент підтримує меншу спільноту, але пропонує користувачам детальну документацію, щоб заповнити прогалини. Даний інструмент DevOps, який обробляє понад 1 мільйон збірок на день, має доступ до даних про те, як працюють команди інженерів і як працює їх код. Користувачами CircleCI є такі компанії, як Spotify, Coinbase, Stitch Fix і BuzzFeed.

До перевах CircleCI можна віднести:

1. Економія часу на технічне обслуговування: інструмент легко налаштувати, оскільки він пропонує належне обслуговування без будь-яких ускладнень.
2. Підвищення продуктивності: краще створювати, тестувати та розгортати в CircleCI. Його додаткові функції, такі як автоматизований паралелізм, сприяють підвищенню продуктивності та ефективності побудови в кількох проектах.
3. Зменшення операційних витрат: CircleCI має вбудовані плагіни для виконання завдань у Linux, OSX, контейнерах. Таким чином, це зменшує операційні накладні витрати на підтримку та встановлення додаткових плагінів. Крім того, жодних додаткових витрат на налаштування чи підтримку власної інфраструктури.
4. Створює узгодженість між кількома проектами: ним легко керувати та ділитися за допомогою інструменту. Він зберігає всі зашифровані змінні, а потім ділиться ними з контейнерами збірки за допомогою SSH.

До недоліків CircleCI відносять доступність лише для сховищ GitHub або BitBucket. Крім цього, плагіни не сумісні з усіма налаштуваннями потоку CI/CD і потрібна покупка додаткових функцій, оскільки інструмент не має необмеженої кількості збірок.

Висновок

CI/CD підтримує сучасну розробку, скорочуючи час між кодуванням і розгортанням. Він максимізує час команд розробки та операцій за рахунок автоматизації ручних кроків процесів інтеграції, доставки та розгортання, одночасно зменшуючи кількість помилок завдяки добре розробленому пакету тестування.

Незважаючи на всі переваги, технічні спеціалісти можуть зіткнутися з деякими проблемами під час впровадження постійної інтеграції та безперервної доставки, зокрема вартість автоматизації, адже для впровадження розробки на основі CI/CD потрібні значні зусилля, інвестиції часу та грошей. Наймання та утримання інженерів DevOps або розміщення репозиторіїв у Git – це лише деякі з процесів, які потрібно виконати. Більше того, просто впровадження практик CI/CD недостатньо. CI/CD потребують постійної підтримки. Тому великим фінансовим організаціям, наприклад, доведеться підтримувати кілька конвеєрів, деякі з яких можуть навіть закінчуватися на різних етапах доставки, що ускладнює тривалість циклу та об’єм роботи.

При цьому платформа для безперервної інтеграції та доставки Jenkins є однією з найбільш часто використовуваних інструментів розробника. Компанії можуть використовувати середовище для автоматизації процесів створення коду, прискорення виробництва програмного забезпечення та навіть для навчання під час роботи з новим програмним забезпеченням і технологіями.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. What is CI/CD? [Електронний ресурс] – Режим доступу: https://www.redhat.com/en/topics/devops/what-is-ci-cd
2. Continuous integration vs. delivery vs. deployment [Електронний ресурс] – Режим доступу: https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment
3. CI/CD: Continuous Integration & Delivery Explained [Електронний ресурс] – Режим доступу: https://semaphoreci.com/cicd
4. What is Jenkins: purpose, advantages and disadvantages [Електронний ресурс] – Режим доступу: https://www.polerstuff.com/what-is-jenkins/

5. Jenkins [Електронний ресурс] – Режим доступу: https://www.techtarget.com/searchsoftwarequality/definition/Jenkins
6. Jenkins For Continuous Integration [Електронний ресурс] – Режим доступу: https://www.edureka.co/blog/what-is-jenkins/
7. What is Jenkins used for? [Електронний ресурс] – Режим доступу: https://www.lambdatest.com/blog/what-is-jenkins/
8. What is Jenkins? Key concepts, architecture, and pros and cons [Електронний ресурс] – Режим доступу: https://codefresh.io/learn/jenkins/

Мартинова Олена Вадимівна – ст. групи 1ІСТ-19б, факультет інтелектуальних інформаційних систем та автоматики, Вінницький національний технічний університет, м. Вінниця, e-mail: marrtynova.a@gmail.com.

Науковий керівник Кулик Ярослав Анатолійович, доцент кафедри автоматизації та інтелектуальних інформаційних технологій, факультет інтелектуальних інформаційних систем та автоматики, Вінницький національний технічний університет, Вінниця, e-mail: kulyk.y.a@vntu.edu.ua.

Martynova Olena Vadymivna – student of group 1IST-19b, Department of Intelligent Information Technology and Automation, Vinnytsia National Technical University, Vinnytsia, e-mail: marrtynova.a@gmail.com.

Supervisor Kulik Yaroslav Anatoliyovych – Associate Professor of the Automation and Intelligent Information Technologies Chair, PhD Art criticism, Vinnytsia National Technical University, Vinnytsia, e-mail: kulyk.y.a@vntu.edu.ua.