УДК 004.41
DOI: 10.31673/2412-9070.2021.013944

А. В. БЕРЕЗНЮК, аспірант;
А. О. МАКАРЕНКО, доктор техн. наук, доцент;
Г. О. ГРИНКЕВИЧ, канд. техн. наук, доцент;
О. І. ГОЛУБЕНКО, канд. техн. наук;
С. Г. ЛАЗЕБНИЙ, аспірант,
Державний університет телекомунікацій, Київ

УДОСКОНАЛЕННЯ МЕТОДІВ НЕПЕРЕРВНИХ ІНТЕГРАЦІЇ ТА ДОСТАВЛЯННЯ ПЗ ЗА ДОПОМОГОЮ ВПРОВАДЖЕННЯ ЕТАПУ ТЕСТУВАННЯ НАВАНТАЖЕННЯ

Упровадження гнучких методів розроблення програмного забезпечення з постійною інтеграцією та постійним доставлянням (CICD) зумовило значне зростання ефективності проектів. Переважно це було досягнуто завдяки тому, що нові функції програмного забезпечення розробляються в кожному спринті та доставляються кінцевим користувачам як результат. Цей підхід є досить ефективним, однак він має збої внаслідок закладеної в нього ідеології.

Дійсно, масштабування системи є поширеним вирішенням, однак до якої міри її слід масштабувати? Процес масштабування потребує знань щодо поточного та очікуваного станів системи. При цьому критичним та складним завданням є еталонний тест виробничої системи, оскільки він стосується завдань бізнесу в реальному часі. Необхідною умовою для впровадження нової версії системи є тестування навантаження для оцінювання очікуваного стану після її впровадження.

Традиційні методи симуляції навантаження не в змозі виявити поведінкові шаблони навантаження трафіку з виробничого середовища. Для подолання цієї проблеми запропонований нами підхід розширює конвеєр CICD та додає три фази автоматизації: оцінювання, тестування навантаження та масштабування. Це дає можливість мінімізувати простої системи за допомогою тестування на стенді та використовувати трафік з виробничого середовища для тестування навантаження, підвищуючи рівень вірогідності результатів. Лише після завершення тестування навантаження системи можна оцінити можливості масштабування. Спочатку конвеєр CICD було розроблено за допомогою серверів Jenkins CI, Git та Nexus з автоматизацією Ansible. Go-replay використовується для дублювання трафіку від виробничого середовища до тестового стенда. Моніторинг Nagios — для аналізу поведінки системи в кожній фазі та доведення на тестовому стенді, що масштабування впорається з цим навантаженням.

Ключові слова: неперервна інтеграція; неперервне доставляння; гнучкі методи розроблення; керування версіями; конфігурація.

ВСТУП

Традиційних методів розроблення програмного забезпечення (ПЗ) недостатньо, щоб задовольнити вимоги бізнесу. Дванадцять принципів гнучкого розроблення ПЗ визначають цілісність процесів та практик у гнучкому керуванні проектами, які застосовуються в таких методологіях, як Extreme Programming (XP), Scrum, Kanban, Crystal, Lean Development Software (LSD), та FDD. Упровадження CICD дало можливість швидко поставляти програмне забезпечення та підвищити продуктивність праці. У 2000 році Мартін Фаулер виклав ідею постійної інтеграції (CI), а пізніше Дж. Хамбл і Д. Фарлі розширили цю ідею з погляду неперервного доставляння (CD) як концепцію розгортання CICD [1]. Основними перевагами CI є зниження ризику та зменшення ціни помилки в програмному забезпеченні завдяки збільшенню швидкості розроблення та отримання зворотного зв'язку про якість продукту. Прискорений час виходу на ринок, підвищення якості продукції, покращення користувацького досвіду, зростання продуктивності та ефективності є ключовими перевагами, що спонукають компанії до інвестування в постійну інтеграцію та постійне доставляння. Оскільки більшість програмних та мобільних розробок розміщуються на інфраструктурі як послузі (IaaS), CICD стала важливою частиною хмарних обчислень.

Система може бути впроваджена та пройти цілісну систему тестування, але повернена до попередньої версії через проблеми продуктивності, які ідентифікуються лише після розгортання. Однією з переваг гнучкого процесу розроблення ПЗ є можливість планувати термін випуску нової версії продукту, але проблема масштабування системи може раптово відкласти поставку, тому ця проблема потребує негайного вирішення. Оцінювання базового та бажаного рівнів системи є тими факторами, які визначають необхідний ступінь масштабування системи. Процес оцінювання системи може спричинити її нестабільність, оскільки має відбуватись у виробничому середовищі. Крім того, нова версія повинна пройти тест навантаження для усвідомлення рівня трафіку, з яким вона може впоратись. Однак традиційні імітаційні випробування навантаження мають дуже високі відхилення від виробничого трафіку, що призводить до недостовірності результатів. Тому метою даної роботи є пошук методів зменшення похибки оцінки продуктивності системи та мінімізація ресурсів, необхідних для проведення даного типу тестування.

Мета статті — удосконалення методів неперервних інтеграції та доставляння програмного забезпечення за допомогою впровадження етапу тестування навантаження.

ОСНОВНА ЧАСТИНА

Аналіз засобів розроблення програмного забезпечення з постійною інтеграцією та постійним доставлянням

Гнучкий процес розроблення ПЗ для CICD. Метод розроблення програмного забезпечення базується на ітеративному розробленні, в якому новий функціонал та вимоги до нього еволюціонують через співпрацю між багатофункціональними командами, здатними до самоорганізації. Гнучке розроблення — засіб підвищення продуктивності розробників програмного забезпечення. Гнучкі методи націлено на мінімізацію ризиків завдяки зведенню розроблення до серії коротких циклів, що мають назву ітерацій, які зазвичай тривають один-два тижні. Кожна ітерація нагадує програмний проект в мініатюрі і містить у собі всі завдання, потрібні для видачі мінімального приросту за функціональністю: планування, аналіз вимог, проектування, кодування, тестування та документування. Хоча окремої ітерації зазвичай недостатньо для випуску нової версії продукту, гнучкий програмний проект готовий до випуску наприкінці кожної ітерації. Після кожної ітерації команда виконує переоцінку пріоритетів розроблення [2]. Технологія Agile акцентує увагу на безпосередньому спілкуванні «віч-на-віч». Вона долучає зокрема й «замовників» (замовники, які визначають продукт, також це можуть бути менеджери продукту, бізнес-аналітики або клієнти) та, можливо, тестувальників, дизайнерів інтерфейсу, технічних авторів і менеджерів. Основною метрикою Agile методів є робочий продукт.

CICD конвеєр. Являє собою програмний інженерний набір правил для компіляцій, тестування та доставляння коду до виробничого середовища.

Неперервна інтеграція (*Continuous Integration*). Це практика розроблення програмного забезпечення, що полягає у виконанні частих автоматизованих компіляцій проекту для якнайшвидшого виявлення та вирішення інтеграційних проблем. У звичайному проекті, де над різними частинами системи розробники працюють незалежно, стадія інтеграції є завершальною. Вона може непередбачувано затримати закінчення робіт. Перехід до неперервної (постійної) інтеграції дає змогу знизити трудомісткість інтеграції та зробити її більш передбачуваною внаслідок раннього виявлення та усунення помилок і суперечностей.

Неперервне доставляння (*Continuous Delivery*). Являє собою програмно-інженерний підхід, за яким команди виробляють програмне забезпечення в коротких циклах, гарантуючи, що воно може бути випущене в будь-який час. Спрямований на створення, тестування і випуск програмного забезпечення швидше і частіше. З таким підходом можна знизити витрати, час і ризик надходження змін, що надає більше додаткових оновлень для додатків у виробництві. Прямолінійний і повторюваний процес розгортання має важливе значення для неперервного доставляння.

Неперервне розгортання (*Continuous Deployment*). Являє собою програмно-інженерний підхід, за яким кожна зміна в готовому для виробничого середовища коді автоматично доставляється кінцевим користувачам відповідно до налаштованих процедур оцінювання якості програмного забезпечення [2].

CICD інструменти. CICD інструменти можна класифікувати як репозиторії вихідного коду, інструменти компіляції, автоматизації, тестування та моніторингу.

• *Репозиторії вихідного коду.* У наш час найбільш популярним є Git. Використовуються різні реалізації: GitHub, Bitbucket, GitLab тощо.

• *Інструменти компіляції.* Ant, Maven, Gradle — найбільш популярні інструменти. CI-сервери надають централізований контроль над інструментами компіляції, зокрема Jenkins, TeamCity, Hudson.

• *Інструменти автоматизації.* Менеджмент конфігурації — це процес опису інфраструктури в коді, ключовою метою якого є зменшення вартості обслуговування та зменшення складності внесення змін. Такими інструментами можуть бути Puppet, Chef, Ansible.

• *Автоматизація тестування.* У гнучких методах розроблення ПЗ є поняття піраміди автоматизованого тестування. Піраміда розділяє тестування на три рівні — модульне, інтеграційне та графічного інтерфейсу.

• *Моніторинг.* Інструментами для моніторингу можуть бути NagiOS, Prometheus and Grafana, Cloudwatch.

Удосконалення методів CICD за допомогою впровадження етапу тестування навантаження

У даній роботі CICD розглядається як основний метод процесу розгортання. У процесі розгортання системи, де важливою є продуктивність, потрібно завжди брати до уваги два рівні. Перший рівень — це розгортання тестового стенда для оцінювання нового розроблення, та другий рівень — власне розгортання у виробничому середовищі. Основними перевагами розгортання на двох рівнях є раннє виявлення помилок на стадії розроблення та можливість швидко виправити ці проблеми. Коли вже наявна система перебуває на оптимізованому рівні

згідно з можливостями команди, команда може спробувати оцінити можливості масштабування системи. Розгортання з тестуванням масштабування за допомогою вдосконаленого CICD (Imp-CICD), що пропонується цим дослідженням, зображено на рис. 1. Відповідно до нього оцінювання, навантажувальний тест, масштабування і резервування — це чотири ітераційні етапи, за якими варто оцінювати масштабування системи. На етапі оцінювання перевіряються виробничі можливості, обмеження та визначається поточний рівень виробничої системи. Нове програмне забезпечення має проходити етапи перевірки навантаження для контролю, чи нова версія задовольняє вимоги виробництва, та виявлення проблем на цьому етапі. Коефіцієнт масштабування відображає чи використовуються додаткові вузли для перевірки нового програмного забезпечення.

Рис. 1. Процес керування доставлянням з Imp-CICD

Коли система має тенденцію до сезонних піків, цей ітеративний підхід є додатковою перевагою для виконання еластичного масштабування на основі попиту на завантаження системи, де хмарні системи мають пріоритет в оптимізації витрат. Уведення нового рівня для випуску нової версії ПЗ може розглядатися як додаткові накладні витрати на виробничу систему, але наявність належних автоматизацій в інструментах керування конфігураціями з Imp-CICD може заощадити час та зменшити ризики виходу з ладу, забезпечуючи стабільність нової версії [3].

«Прапорці нових функцій», «темні запуски», «усе відразу», «котячись на місці», «подвоєння», «канарка», «незмінне розгортання» та «синьо-зелене розгортання» — це стратегії розгортання, які рекомендує AWS [4]. Аналізуючи їхні особливості, виявляється, що всі застосовуються при підході CICD, але метод «котячись на місці» є пріоритетним, оскільки він найпростіший та широко застосовуваний [5].

Еталонний тест показує відносну продуктивність однієї системи, яку було взято за найкращу на основі ефективності для бізнесу та інших технічних вимог. Він має відбуватися з кожною поставкою, виконуватися за однакових умов та в ізольованому середовищі.

Для точного зразку виробничого трафіку використовується Go-repay. Він здатний дублювати трафік без впливу на хост-сервер. Go-replay захоплює веб-трафік, а потім відтворює трафік з одного сервера на інший, а також може зберегти модель трафіку до файла та пізніше відтворити його на цільовому сервері [7].

Тестування навантаження визначає нові обмеження ПЗ та граничні можливості наявної системи. Навантажувальне тестування, стрес-тестування та пікове тестування — це типи перевірок працездатності відповідно до стратегії AWS. Вони використовуються для тестування за попередньо визначеними показниками. Go-replay застосовується для дублювання трафіку з виробничого, а Ansible автоматизація — для керування процесом.

Нехай E — поточне базове навантаження на сервер, E_1 — максимальне навантаження, оброблене під час випробування навантаження. Загальне навантаження системи $E \cdot n$, де n — кількість вузлів. Навантаження обробляється новою версією в наявних вузлах. Тому додатковою вимогою сервера є:

$$E_s = \frac{n(E - E_1)}{E_1}. \quad (1)$$

Оскільки кількість серверів не може бути десятковим значенням, кількість завжди має повертати наступне ціле число від результату.

Під час фази ідентифікації масштабування, якщо система потребує збільшення ресурсів, необхідно ініціювати запуск автоматизації на Ansible для зміни стану системи.

З метою реалізації конвеєра на початковій фазі ми вибрали такі інструменти. Git використовується як контроль версій вихідним кодом, Nexus — як сховище для виконуваних файлів, а Jenkins застосовується як сервер Imp-CICD. Щоб уникнути інфраструктурних відмінностей між виробничим та тестовим середовищами, налаштуємо ідентичні середовища за допомогою хмарних технологій. Доставляння продукту відбувається за допомогою завдань на Ansible, які запускає Imp-CICD сервер. Nagios використовується для моніторингу ключових системних метрик стану системи. Закінчення процесу розгортання нової версії продукту може розцінюватися як тригер для тестування навантаження та масштабування.

Процес автоматизації оцінювання ініціюється сервером Imp-CICD та може бути відображений через вивід Jenkins завдань. Для цього потрібен новий базовий сервер, який є ідентичним із виробничим сервером та має те саме середовище виконання.

Як зазначалося раніше, Go-Replay використовується для дублювання трафіку, що контролює відповідна автоматизація, а Nagios — для збору метрик. Процес оцінювання навантаження можна описати таким алгоритмом.

1. Забезпечити новий сервер однорідними специфікаціями та такою самою ємністю, як і базовий сервер.

2. Розгорнути поточну версію виробництва в тестове середовище.

3. Ініціювати повторне відтворення з попередньо вибраного списку серверів додатків.

4. Забезпечити початковий трафік, починаючи з 5%, та поступово збільшувати його, додаючи 5% на кожному кроці з інтервалом у 5 хв.

5. Перевірити стан монітора Nagios на затримку відповіді, аби побачити, що він досягає визначеної очікуваної максимальної затримки, поки послуга доступна [8].

6. Якщо базовий сервер продовжує нормально функціонувати, коли трафік Додатка 1 сягає 100%, то автоматизація запускає трафік дублювання, як на кроці 4, вибравши наступний сервер додатка для збільшення транспортного навантаження.

7. Продовжувати кроки 4, 5 та 6, поки тестовий сервер не порушить обмеження затримки або його послуга стане недоступною.

8. Припинити генерацію трафіку з усіх серверів додатків, коли оцінка досягла своєї межі.

Після того, як автоматизацію навантаження завершено, тести навантаження можна ініціювати за допомогою сервера Jenkins або створення нового сервера клонуванням потужностей виробництва.

Після визначення показників навантаження на виробничу та нову версію додатка масштабування може бути розраховане за формулою (1). Автоматизація масштабування розгорне існуючу версію на нові сервери та під'єднає їх до моніторингу [9]. Етапи автоматизації масштабування:

- обчислити потребу в додаткових потужностях за формулою (1);
- забезпечити нові сервери однорідними специфікаціями та однаковою ємністю;
- розгорнути поточну виробничу версію на нових серверах;
- додати нові сервери до балансувальника навантаження.

Тестування запропонованого методу та аналіз здобутих результатів

Автоматизація тестового стенда містить п'ять модулів, таких як сервер розгортання, налаштування Nagios, оцінювання стану виробничої системи, тестування навантаження та масштабування. Чотири сервери з додатками та сервер моніторингу Nagios зображено на рис. 2, де четвертий сервер вибрано як орієнтир (тестовий сервер навантаження). HAproxy використовується як балансувальник, Go-Replay дублює трафік з 3-го на 4-й сервер.

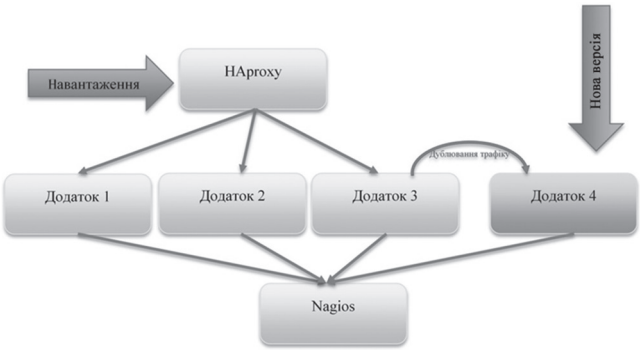


Рис. 2. Тестовий стенд (тест навантаження)

Тести навантаження запускаються в фазі тестування і кожен етап триває 20 хв. Графіки Nagios, діаграма Jmeter, на якій зображено час відгуку, та статистика трафіку проксі використовувались для аналізу працездатності системи [10].

Згідно з еталонним тестом час реакції становить близько 52…73 мс (рис. 3, ділянка, позначена як 1). У разі 100 % навантаження для початкового часового періоду, він виконується нижче за критичний рівень, але зі зростанням навантаження виходить за межі критичного рівня, і система стає нестабільною, збільшуючи час відповіді. За відведений проміжок часу було оброблено по 14 984 запитів на 4 997 вузлах.

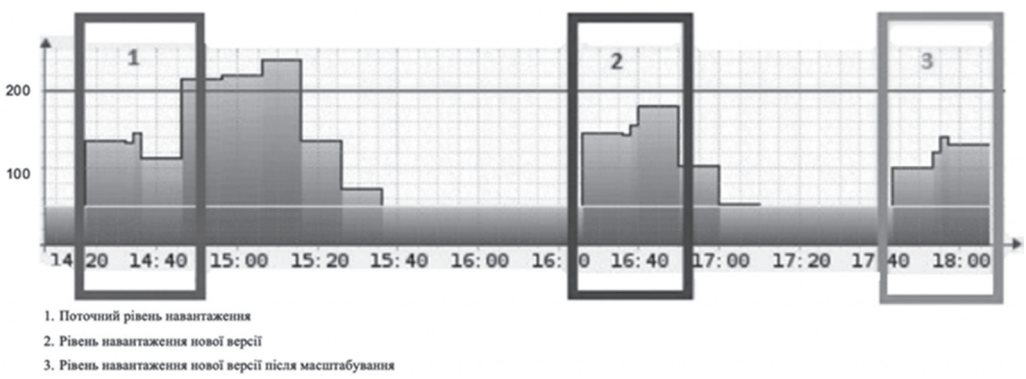
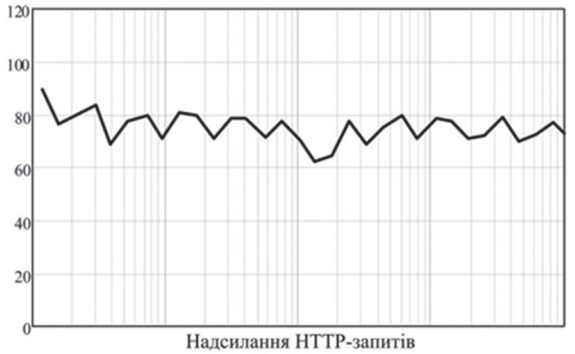


Рис. 3. Тестовий стенд (схема завантаження процесора)

У тестуванні навантаження нова версія почала відповідати між 72...82 мс, що вище за початковий рівень орієнтира. На схемі тестування навантаження процесора (див. рис. 3) в ділянці, позначеній як 2, час трохи перевищує початкові часові межі еталонного тесту. Нова версія додатка може обробити лише 4 241 запитів за той самий час на вузол.

Згідно з формулою (1) нам потрібно на один сервер більше для обслуговування того самого навантаження. Після масштабування (рис. 4) час реакції було збільшено до 72 мс, що є вище, ніж початковий показник, але трохи краще, ніж тестування навантаження. На цьому етапі процесор може обробити 15 211 загальних запитів, і шаблон поведінки процесора набагато кращий, ніж на інших фазах, оскільки на одному сервері навантаження зменшується (див. рис. 3, ділянка, позначена як 3). Додавання нового сервера дає перевагу у послідовній пропускній здатності, але не показує значного покращення часу реагування системи. Результати тестування довели, що нова версія ПЗ може призвести до непередбачуваного погіршення якості обслуговування, тому під час випуску нової версії програмного продукту потрібно вводити етап тестування навантаження. У нашому разі нова версія має незначне погіршення виробничої потужності, що призводить до того, що нам необхідно додавати ще один сервер для обслуговування тієї самої кількості користувачів.

а



б

Рис. 4. Тест навантаження часу затримки відгуку: *а* — до масштабування; *б* — після масштабування

ВИСНОВКИ

Розроблений удосконалений метод Imp-CICD дає змогу створити ефективний процес розроблення та доставляння програмного забезпечення до середовищ тестування та реалізації. Мету дослідження досягнуто за допомогою запропонованого покращення процесу Imp-CICD через упровадження етапу тестування навантаження. Це уможливлює неперервне тестування кожної версії програмного забезпечення на відмовостійкість та зміну продуктивності коду.

Go-replay дає можливість транслювати постійно однаковий трафік для релевантності тестування на стендах. Ansible автоматизація дозволяє більш ефективно виконувати підхід тестового стенда.

Список використаної літератури

*1. **Cunningham W**. Principles behind the Agile Manifesto [Електронний ресурс] // Agile Manifesto.org. URL:*

http://agilemanifesto.org/principles.html

*2. **Cohn M**. Succeeding with Agile [Електронний ресурс] // Addison-Wesley. URL:*

http://1.droppdf.com/files/AFvTS/succeeding-with-agile-mike-cohn.pdf

*3. **Practicing** Continuous Integration and Continuous Delivery on AWS [Електронний ресурс] // Amazon Web Services Inc. URL:*

https://aws.amazon.com

*4. **Getting** started with Continuous Integration in Software Development [Електронний ресурс] // Infosys Limited. URL:*

https://www.infosys.com

*5. **An efficient** network monitoring and management system [Електронний ресурс] R. Khan, S. Ullah Khan, R. Zaheer, M. I. Babar // International Journal of Information and Electronics Engineering. URL:*

https://www.researchgate.net/publication/316060077_An_Efficient_Network_ Monitoring_and_Management_System

*6. **Kucera A., Glos P., Pitner T**. Fault detection in building management system networks [Електронний ресурс] // IFAC Proceedings. URL:*

https://www.sciencedirect.com/science/article/pii/S1474667015373638

*7. **Feitelson D. G., Frachtenburg E., Beck K. L**. Development and Deployment at Facebook [Електронний ресурс] // IEEE Internet Computing. URL:*

https://ieeexplore.ieee.org/document/6449236

*8. **Continuous** deployment at Facebook and Oando [Електронний ресурс] / T. Savor, M. Douglas, M. Gentili, L. Williams, K. Beck, M. Stumm // 38th International Conference on Software Engineering Companion. URL:*

https://ieeexplore.ieee.org/document/6449236

*9. **Bass L., Weber I., Zhu L.** DevOps: A Software Architect's Perspective // Addison-Wesley Professional, 2015. 352 p.*

*10. **Meyer M.** Continuous Integration and Its Tools [Електронний ресурс] // IEEE Software. URL: https://ieeexplore.ieee.org/document/6802994*

А. В. Березнюк, А. А. Макаренко, А. А. Гринкевич, А. И. Голубенко, С. Г. Лазебный

УСОВЕРШЕНСТВОВАНИЕ МЕТОДОВ НЕПРЕРЫВНОЙ ИНТЕГРАЦИИ И ДОСТАВКИ ПО С ПОМОЩЬЮ ВНЕДРЕНИЯ ЭТАПА ТЕСТИРОВАНИЯ НАГРУЗКИ

Внедрение гибких методов разработки программного обеспечения с постоянной интеграцией и постоянной доставкой (CICD) способствовало невероятному увеличению эффективности разработки. Достигнуто это было в большей степени благодаря тому, что новая версия продукта с дополнительным функционалом программного обеспечения разрабатывается в каждом спринте и является результатом этого отрезка времени. Подход зарекомендовал себя как достаточно эффективный, но у него есть свои проблемы, которые заложены в идеологии.

Действительно, масштабирование системы зачастую решает проблемы, но возникает вопрос до какой степени необходимо масштабировать ее? Процесс масштабирования требует информации о текущем и ожидаемом состояний системы. При этом критическим и в то же время сложным заданием является эталонный тест производственной системы, поскольку он затрагивает бизнес-задачи в реальном времени. Необходимое условие для внедрения новой версии системы — это тестирование нагрузки для оценки ожидаемого состояния после разворачивания.

Традиционные методы симуляции нагрузки не в состоянии выявить поведенческие паттерны нагрузочного трафика с производственной среды. Для преодоления этого вызова предложенный нами подход расширяет конвейер CICD и добавляет три фазы автоматизации: оценку, тестирование нагрузки и масштабирование. Это позволит минимизировать простои системы при помощи тестирования на стенде, используя производственный трафик, что в свою очередь приводит к повышению достоверности результатов. Только после окончания тестирования нагрузки системы мы можем сделать вывод о масштабируемости. Изначально конвейер CICD был разработан при помощи серверов Jenkins CI, Git и Nexus с автоматизацией Ansible. Go-replay используется для дублирования трафика с производственной среды к тестовому стенду. Мониторинг Nagios — для исследования поведения системы в каждой фазе и аргументации на тестовом стенде, что после масштабирования система справиться с заданной нагрузкой.

Ключевые слова: непрерывная интеграция; непрерывная доставка; гибкие методы разработки; управление версиями; конфигурация.

A. V. Bereznyuk, A. O. Makarenko, G. O. Grynkevych, O. I. Holubenko, S. G. Lazebny

IMPROVE CONTINUOUS INTEGRATION AND DELIVERY METHODS BY IMPLEMENTING A LOAD TESTING STAGE

The contradiction between system stability and development speed development introduces new challenges to processes automatization. Significant increase of quality and speed of software development was reached due to agile method and continuous integration and continuous delivery (CICD) approach, which enabled rapid software changes by breaking development process into small iterative stages. New product version with new features incorporated is the result of each iteration. Then, there are automation tools typically used for testing and building. However, there is space for performance issues and delivery failures in this scheme. System scaling is generic approach for resolving performance problems, but the question how much the system should be scaled remains unanswered. Load testing procedure is commonly used but it is still not perfect, as testing environment is very different from production environment. Therefore, current paper is aimed at overcoming described difficulties by extending CICD approach with load testing automatization, system benchmarking, which allows system interruption minimization, and system scaling evaluation by scale factors calculation. Using these enrichments more precise testing result will be provided. We also suggest using such tools as Go-replay for traffic mirroring and Nagios for monitoring. Hence, this paper suggests enriching and increasing the efficiency of CICD approach. In addition, paper addresses methods of monitoring metrics collecting in centralized system. They will be used in further analysis and decision-making process regarding new product version in the automated mode.

Keywords: continuous integration and continuous delivery (CICD); Server monitoring; version management; configuration management; performance testing; performance scalability; pipelines; bugs detection.