Informatyka techniczna i sztuczna inteligencja
Technical informatics and Artificial Intelligence
Engineer of XXI Century'2024

Implementation of web application deployment automation

Andrzej Jasiński [1], Opiekun naukowy: Tomasz Zając[2,*]

[1] *Uniwersytet Bielsko-Bialski, WBMiI, Informatyka specjalność: inżynieria oprogramowania, jasinski.a11@gmail.com*
[2] *mgr inż., Uniwersytet Bielsko-Bialski, WBMiI, tzajac@ubb.edu.pl*
[*] *Corresponding author tzajac@ubb.edu.pl*

Abstract: The article discusses the modern approach to software engineering, specifically the DevOps methodology and the use of deployment pipelines as an effective tool for automating the process of building, testing, and deploying software. It describes the various stages of constructing a deployment pipeline, covering the entire cycle - from retrieving code from the repository, through the deployment process, to making it available to users. It also addresses issues related to testing and monitoring applications to ensure their high quality and stability.

Keywords: DevOps, software engineering, deployment pipeline, automation, continuous integration and delivery, maintenance

Implementacja automatyzacji wdrażania aplikacji internetowej

Andrzej Jasiński [1], Opiekun naukowy: Tomasz Zając[2,*]

1 Uniwersytet Bielsko-Bialski, WBMiI, Informatyka specjalność: inżynieria oprogramowania, jasinski.a11@gmail.com
2 mgr inż., Uniwersytet Bielsko-Bialski, WBMiI, tzajac@ubb.edu.pl
[*] *Corresponding author tzajac@ubb.edu.pl*

Streszczenie: Artykuł omawia współczesne podejście do inżynierii oprogramowania, a w szczególności metodologię DevOps oraz wykorzystanie potoków wdrażania jako efektywnego narzędzia do automatyzacji procesu tworzenia, testowania i wdrażania oprogramowania. Opisano poszczególne etapy budowy potoku wdrożeniowego, obejmujące cały cykl: od pobrania kodu z repozytorium, przez proces wdrożenia, aż po udostępnienie użytkownikom. Omówiono również kwestie związane z testowaniem i monitorowaniem aplikacji, aby zapewnić ich wysoką jakość i stabilność działania.

Słowa kluczowe: DevOps, inżynieria oprogramowania, potok wdrażania, automatyzacja, ciągła integracja i dostarczanie, utrzymanie;

1. Wprowadzenie

Nowoczesna inżynieria oprogramowania zawiera w sobie wiele elementów, bez których ciężko wyobrazić sobie efektywną pracę. To co kiedyś uznano by za niepotrzebną komplikację procesu, jakim jest pisanie kodu, a zatem wytwarzanie oprogramowania, dzisiaj jest standardem, powielanym i propagowanym pośród największych światowych przedstawicieli sektora technologicznego. W świecie branży cyfrowej podział obowiązków pomiędzy role inżynierów oprogramowania, jakości oraz systemów zaczyna się zacierać. Wynika to głównie z przemian jakie zaszły na przestrzeni ostatnich trzydziestu lat. Samo napisanie kodu aplikacji wraz z jego przetestowaniem i wdrożeniem do środowiska produkcyjnego nie oznacza zakończenia pracy dla któregokolwiek z zespołów. Rzeczywiste trudności związane z wytwarzaniem niezawodnego, wydajnego i skalowalnego oprogramowania zaczynają się w chwili udostępnienia go klientom.
Relatywnie, gdy aplikacja zyskuje na popularności, przed zespołami wytwarzającymi oprogramowanie pojawia się kolejna trudność, mianowicie ogólnie pojęte utrzymanie (ang. maintenance). Mowa tu nie tylko o zapewnieniu, że aplikacja jest wolna od błędów, ale także o przewidywaniu i reagowaniu na incydenty, zanim jeszcze się wydarzą.

DOI: https://doi.org/10.53052/9788367652292.32

1.1. CI/CD i potoki wdrażania

Ciągła integracja i wdrażanie (ang. continous integration and continous deployment) są to praktyki inżynierii oprogramowania, które pozwalają na szybkie i automatyczne wprowadzenie zmian w aplikacjach. Praktyki te zakładają użycie narzędzi automatyzujących manualne i powtarzalne procesy wdrażania aplikacji w celu wyeliminowania z procesu błędów ludzkich oraz zminimalizowania ryzyka wdrożenia błędów krytycznych do aplikacji. To w jaki sposób ciągła integracja, wdrażanie są zbudowane i utrzymywane w organizacji może zaważyć na jakości udostępnianego kodu, nierzadko mając wpływ na straty wizerunkowe i finansowe.

Potok wdrażania (ang. deployment pipeline) to zautomatyzowany proces prowadzący oprogramowanie od fazy wytwarzania, przez fazę testów, skończywszy na wdrożeniu aplikacji do środowiska produkcyjnego, a więc udostępnieniu jej użytkownikom. Potok wdrażania składa się z wielu etapów, które mogą obejmować budowanie, testowanie, wdrażanie i monitorowanie aplikacji. Głównym celem automatyzacji wdrażania aplikacji jest przyspieszenie procesu oraz poprawa bezpieczeństwa wdrażanego kodu w środowisku produkcyjnym. Nowoczesna inżynieria oprogramowania zakłada, że środowisko testowe jest całkowicie odizolowane od środowiska klienckiego. Celem implementacji ciągłej integracji i dostarczania jest zapewnienie, że wdrażany kod jest możliwy do udostępnienia w krótkim czasie, a także że został należycie przetestowany.

2. Opis proponowanego rozwiązania

Nowoczesne procesy wdrażania aplikacji stają się coraz bardziej złożone, a ich automatyzacja odgrywa kluczową rolę w zapewnieniu jakości oraz niezawodności produktu końcowego. Proponowane rozwiązanie opiera się na wykorzystaniu narzędzi do automatyzacji, zarządzania infrastrukturą i monitorowania w chmurze.

2.1. Wymagania funkcjonalne

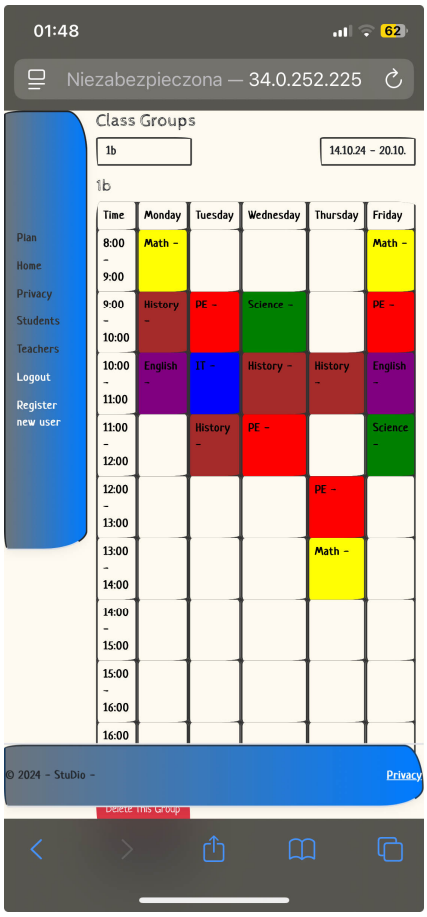
Nadrzędnym celem udostępniania oprogramowania za pośrednictwem potoku wdrożeniowego było zapewnienie, że nowa wersja zostanie odpowiednio przetestowana i będzie stabilna. Organizacja udostępniająca aplikację milionom użytkowników musi zdawać sobie sprawę z faktu, że błędy wykryte na wczesnym etapie rozwoju produktu są mniej kosztowne do usunięcia niż te wykryte już po wdrożeniu do produkcji.

2.2. Automatyzacja i wczesne wykrywanie błędów

Automatyzacja procesów wdrażania odgrywa bardzo ważną rolę w minimalizowaniu ryzyka wystąpienia błędów oraz w zapewnieniu spójności środowisk testowych i produkcyjnych. Dzięki zautomatyzowanym potokom wdrażania możliwe jest szybkie i efektywne przeprowadzanie testów, integracji oraz wdrożeń, co przekłada się na wyższą jakość końcowego produktu. Powtarzalny proces wdrażania zapewnia większą niezawodność samego oprogramowania, ponieważ odciąża zespoły programistyczne od konieczności martwienia się o konfigurację środowiska, w którym program działa. Manualnie przeprowadzane procesy wdrażania mogą powodować problemy nie tylko po stronie samego oprogramowania, ale także po stronie środowiska, w którym jest ono uruchamiane. Nawet drobne różnice w konfiguracji serwerów mogą prowadzić do nieprzewidywalnych błędów aplikacji.

Rysunek 1. Wykres kosztów wykrycia i naprawy błędów

Na rysunku 1 przedstawiono wykres kosztów jakie ponosi organizacja w przypadku awarii oprogramowania na poszczególnych etapach jego wytwarzania. Błędy wykryte już po udostępnieniu oprogramowania są groźne z dwóch powodów. Po pierwsze mogą mieć wpływ na stabilność i bezpieczeństwo aplikacji, która może ulec awarii, co może zostać wykorzystane np. do ataku hackerskiego. Po drugie nieprawidłowo działające oprogramowanie fatalnie wpływa na wizerunek i zaufanie użytkowników. Bez względu na to, jak szybko usterka zostanie naprawiona, jej wystąpienie u klientów, a zwłaszcza sytuacja, w której to oni zgłaszają nieprawidłowość, wskazuje na poważne niedociągnięcia w zarządzaniu procesem wytwarzania oprogramowania.
Rysunek 2 przedstawia aplikację przygotowaną przy pomocy platformy .NET 8. Program umożliwia odbiorcom przegląd planu zajęć oraz komunikację w czasie rzeczywistym z innymi użytkownikami. Do dyspozycji administratora oddano funkcjonalność tworzenia nowych użytkowników, planów zajęć oraz przypisywania uczniów i nauczycieli do poszczególnych klas. Przygotowując automatyzację wdrażania należało mieć na uwadze przede wszystkim rozdzielenie środowiska testowego oraz produkcyjnego. W optymalnej konfiguracji środowiska deweloperskie i produkcyjne powinny różnić się tylko poziomem zabezpieczeń. Zmniejsza to ryzyko wystąpienia błędów, które zostają wykryte dopiero po udostępnieniu oprogramowania klientom, na przykład z powodu braku odpowiednich zależności.



Rysunek 2. Ekran główny aplikacji

2.3. Architektura proponowanego podejścia

Architektura zaproponowanego rozwiązania skupia się na automatyzacji procesu wdrażania aplikacji z wykorzystaniem narzędzi takich jak Jenkins, Packer, Terraform, Google Cloud Platform i Splunk. Komponenty te współpracują, aby zapewnić płynne budowanie, testowanie, tworzenie obrazów maszyn oraz zarządzanie infrastrukturą w chmurze. Rysunek 3 przedstawia pełną architekturę proponowanego podejścia z uwzględnieniem zależności między komponentami.

2.3.1 Jenkins

Jenkins jako silnik CI/CD, stanowi rdzeń całego procesu. Automatyczne kompilacje, integracja z chmurą czy raportowanie o wynikach testów to tylko niektóre z jego funkcji, które posłużyły w budowie potoku wdrażania aplikacji:

- Uruchamia proces budowania aplikacji, zapewniając, że najnowsza wersja kodu jest poprawnie skompilowana.
- Uruchamia testy automatyczne z wykorzystaniem biblioteki xUnit. Niepowodzenie w wykonaniu testów skutkuje błędem kompilacji kończącym proces wdrażania.
- Inicjuje narzędzie Packer, aby stworzyć obraz maszyny wirtualnej zawierający aplikację oraz jej zależności. Obraz ten jest podstawą do wdrożenia na wybranym środowisku.
- Umożliwia wdrożenie aplikacji na wybrane środowisko.

2.3.2 Packer

Packer jest narzędziem odpowiedzialnym za tworzenie powtarzalnych i konfigurowalnych obrazów maszyn wirtualnych. W kontekście proponowanego rozwiązania Packer przygotowuje obraz maszyny zawierający skompilowaną aplikację oraz jej wszystkie zależności. Obrazy te są konfigurowalne i zoptymalizowane pod konkretne środowiska. Packer wspiera integrację z Google Cloud Platform, co pozwala na łatwe wdrożenie aplikacji w wybranym środowisku. Tworzenie obrazów za pomocą Packer'a gwarantuje spójność środowiska aplikacyjnego oraz upraszcza procesy wdrażania i aktualizacji.

2.3.3 Terraform

Terraform jest narzędziem typu infrastruktura jako kod (ang. Infrastructure as Code - IaC), które umożliwia zarządzanie infrastrukturą chmurową. W proponowanym rozwiązaniu Terraform jest odpowiedzialny za:

- Tworzenie lub aktualizację instancji serwera w chmurze Google Cloud Platform, który udostępnia aplikację.
- Zapewnienie, że infrastruktura jest zgodna z opisanym stanem docelowym.
- Implementację infrastruktury w zależności od środowiska – testowego lub produkcyjnego – co pozwala na ich rozdzielenie oraz odpowiednie zarządzanie.

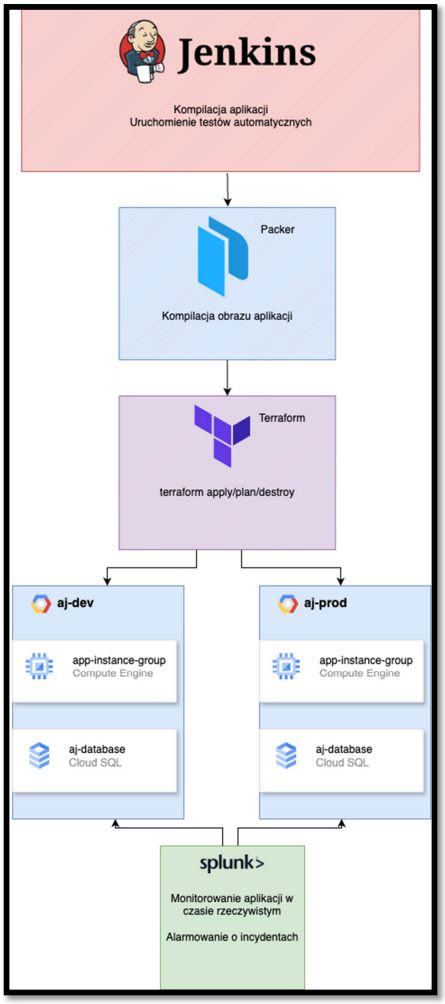
2.3.4 Google Cloud Platform

Google Cloud Platform (GCP) w proponowanym rozwiązaniu pełni rolę dostawcy infrastruktury chmurowej. Obrazy aplikacji zbudowane przez Packer są wykorzystywane przez Terraform do wdrożenia na serwerach w chmurze. GCP zapewnia skalowalność, bezpieczeństwo oraz wiele udogodnień pozwalających na prostsze zarządzanie infrastrukturą aplikacji. Wśród narzędzi GCP wymienić można Cloud SQL, które posłużył do przygotowania serwera bazy danych programu czy Cloud Secret Manager, który przechowuje wszelkie poufne dane, takie jak łańcuch połączenia (ang. connection string), tokeny uwierzytelniające, czy hasła.

2.3.5 Splunk

Splunk jest wykorzystywany do monitorowania aplikacji oraz analizowania logów w czasie rzeczywistym. Umożliwia tworzenie paneli monitorujących, które wspierają zarządzanie wydajnością i incydentami. Splunk jest również używany do tworzenia alarmów, które pomagają w szybkim wykrywaniu i naprawie problemów związanych z działaniem aplikacji. Dzięki integracji ze Splunk, aplikacja ma możliwość proaktywnego monitorowania i analizowania logów, co poprawia jej niezawodność oraz ułatwia diagnozowanie potencjalnych problemów.

DOI: https://doi.org/10.53052/9788367652292.32



Rysunek 3. Architektura potoku wdrażania

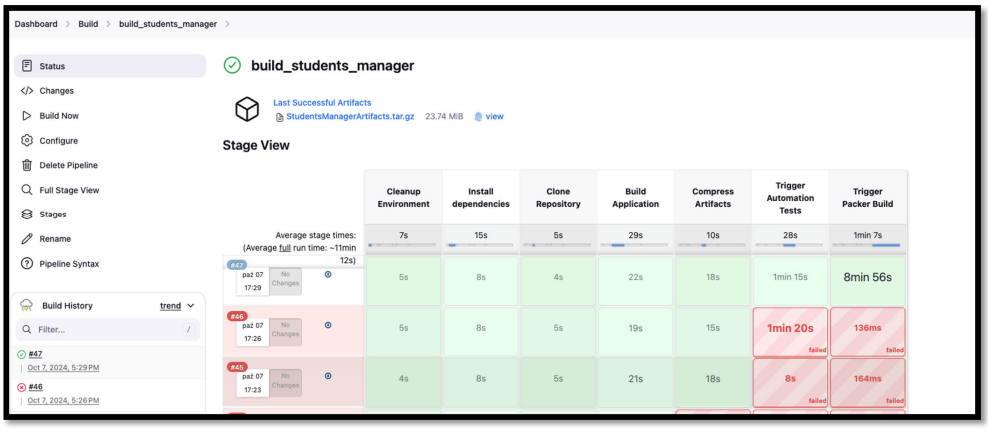
2.4. Proces wdrażania

Proces wdrażania aplikacji składa się z kilku kluczowych etapów, które gwarantują jej poprawne uruchomienie oraz stabilne działanie w środowisku produkcyjnym. Każdy krok tego procesu jest zautomatyzowany, aby zminimalizować błędy ludzkie oraz przyspieszyć czas wdrożenia.

2.4.1 Inicjalizacja przez Jenkins

Proces wdrażania rozpoczyna się od inicjalizacji w Jenkinsie, który oferuje intuicyjny panel użytkownika, umożliwiający łatwe tworzenie i uruchamianie automatycznych kompilacji (rysunek 4). Po wyborze odpowiednich parametrów, Jenkins pobiera najnowszą wersję kodu z repozytorium, a następnie przystępuje do budowania i testowania aplikacji. Na rysunku 5 przedstawiono szczegółowy widok procesu kompilacji, w którym wszystkie etapy – od pobrania kodu, przez budowanie, aż po testy – są monitorowane w czasie rzeczywistym.

Rysunek 4. Widok głównej kompilacji aplikacji oraz kompilacji narzędzia Packer.

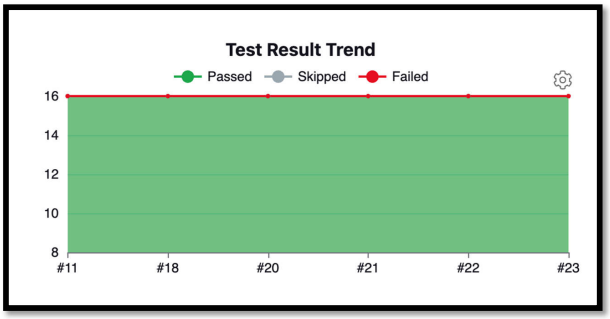


Rysunek 5. Widok panelu użytkownika głównej kompilacji aplikacji.

2.4.2 Testy automatyczne

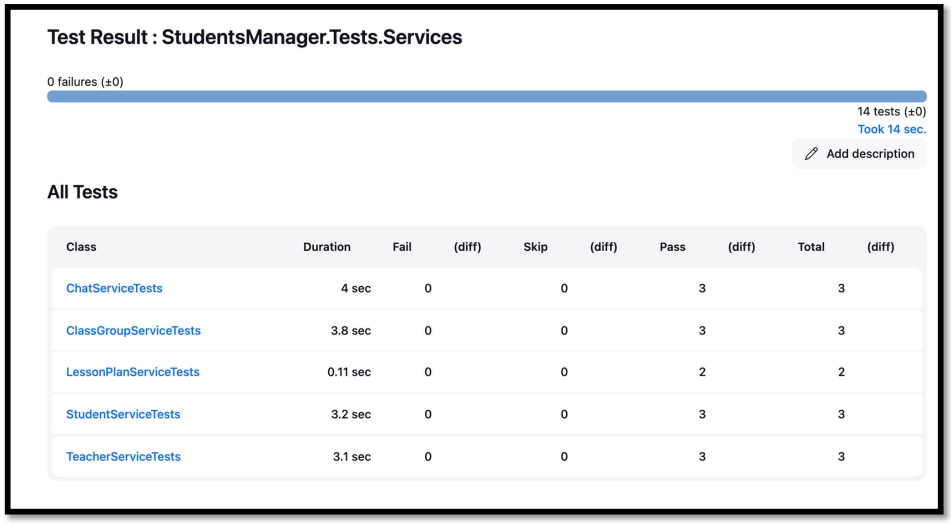
Proces bezpiecznego wdrażania aplikacji nie może odbyć się bez wcześniejszej fazy testowania. W proponowanym rozwiązaniu jednym z etapów wdrażania projektu było uruchomienie testów automatycznych, które testują podstawowe funkcje działania aplikacji, takie jak wysyłanie wiadomości, tworzenie i usuwanie użytkowników czy planów zajęć. Tego typu testy pozwalają na wstępną walidację nowego kodu, ponieważ zakończyłyby się niepowodzeniem i przerwaniem kompilacji w sytuacji, gdy któraś z funkcjonalności nie działa. Na rysunkach 6 i 7 przedstawiono rezultaty testów w formie graficznej, co pozwala na łatwiejszą identyfikację problemów.

Z komentarzem [TZ1]: Tegu typu



Rysunek 6. Wyniki testów w formie wykresu.

Z komentarzem [TZ2]: Brak odwołania

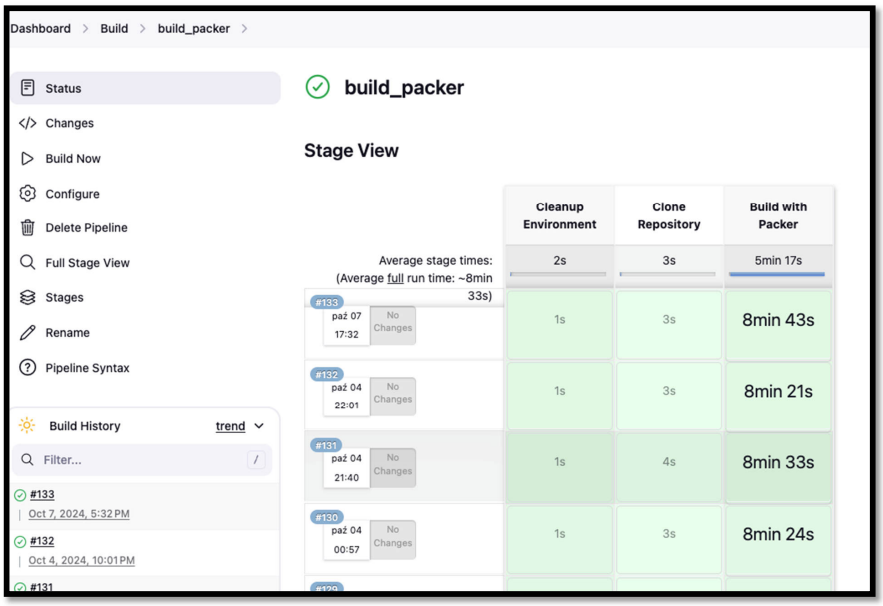


Rysunek 7. Szczegółowe wyniki testów są zapisywane po każdej udanej kompilacji.

Z komentarzem [TZ3]: Brak odwołania

2.4.3 Tworzenie obrazu za pomocą narzędzia Packer

Następnie, Jenkins uruchamia narzędzie Packer, aby stworzyć obraz maszyny wirtualnej. Każda kompilacja pobiera najnowszy kod aplikacji z repozytorium Git po czym przystępuje do tworzenia obrazu (rysunki 8 i 9). Obraz ten zawiera skompilowaną aplikację oraz wszystkie wymagane zależności.



Rysunek 8. Widok panelu użytkownika buildu narzędzia Packer.

Z komentarzem [TZ4]: Brak odwołania

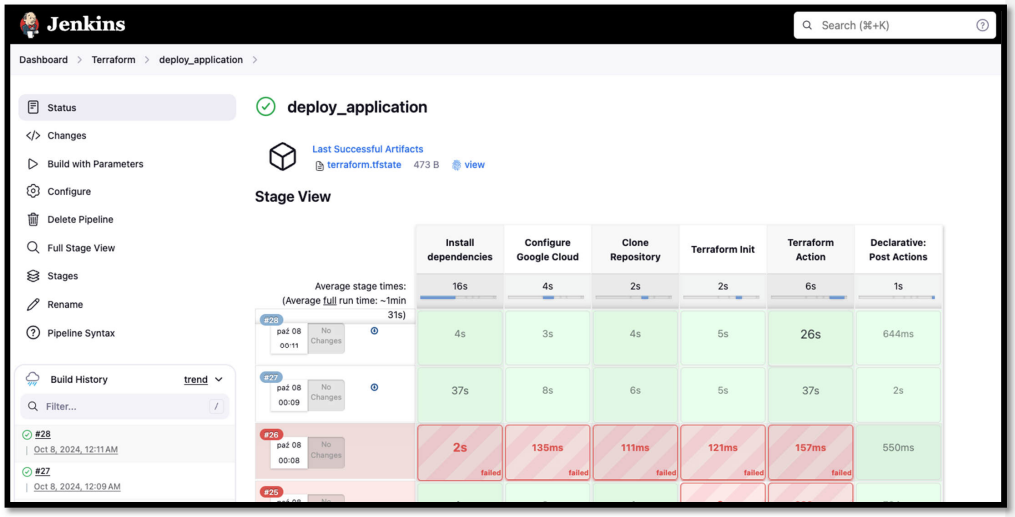
```
==> Wait completed after 8 minutes 32 seconds

==> Builds finished. The artifacts of successful builds are:
--> googlecompute.default: A disk image was created in the 'aj-dev-434320' project: dotnet-app-20241007033220
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Rysunek 9. Widok wyjścia konsolowego narzędzia Packer.

Z komentarzem [TZ5]: Brak odwołania

2.4.4 Wdrożenie infrastruktury przez Terraform

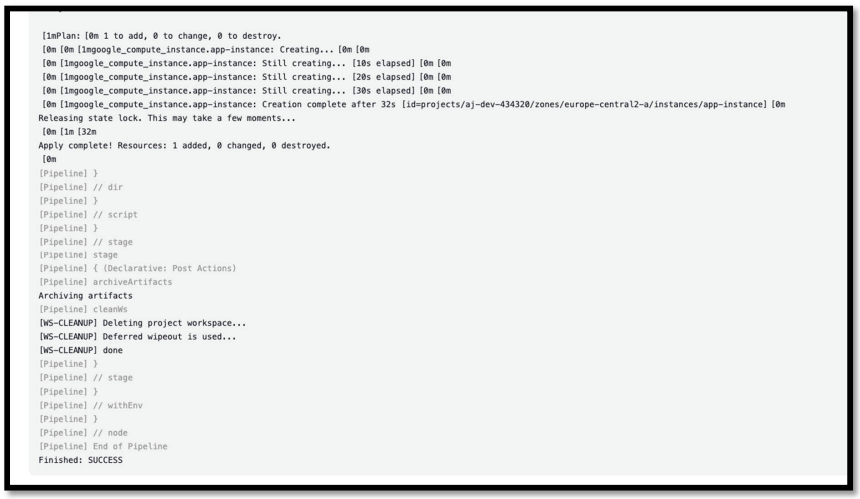
Po utworzeniu obrazu, Terraform automatycznie tworzy lub aktualizuje instancję w chmurze GCP, w zależności od wybranego środowiska. Dzięki temu narzędziu infrastruktura jest zarządzana w sposób spójny i zgodny ze stanem docelowym, co zapewnia powtarzalność i przewidywalność wdrożeń. Proces kompilacji pobiera pliki Terraform z repozytorium Git, a następnie rozpoczyna wdrażanie na wskazane środowisko (rysunek 10). Użytkownik ma możliwość wyboru środowiska docelowego, co umożliwia elastyczne zarządzanie operacjami wdrożeniowymi (rysunek 11).



Rysunek 10. Widok panelu użytkownika kompilacji narzędzia Terraform.

Z komentarzem [TZ6]: Brak odwołania

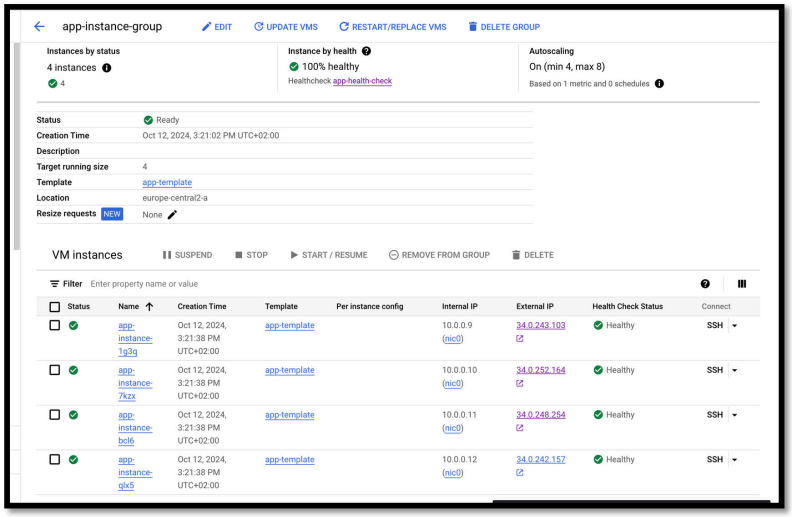
Rysunek 11. Wybór środowiska do wdrożenia aplikacji.



Rysunek 12. Widok wyjścia konsolowego kompilacji narzędzia Terraform – narzędzie utworzyło nową instancję aplikacji w chmurze GCP.

Z komentarzem [TZ7]: Brak odwołania

Po przetestowaniu kodu i zakończeniu wdrażania przez Terraform (rysunek 12), instancje z aplikacją są wdrożone na produkcję. Auto-skalowanie jest włączone, by zapewnić większą przepustowość w sytuacji wzmożonego ruchu. Mechanizm zdrowia aplikacji (*ang. health check*) jest odpowiedzialny za monitorowanie działania aplikacji na wszystkich serwerach w grupie instancji (rysunek 13).

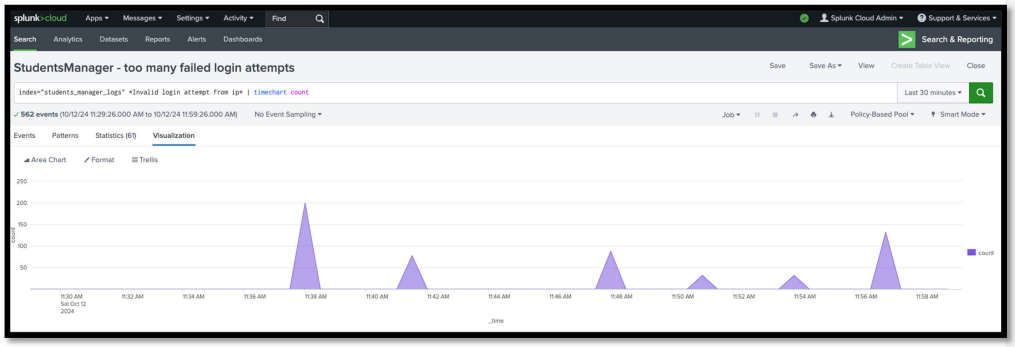


Rysunek 13. Widok grupy instancji w konsoli GCP.

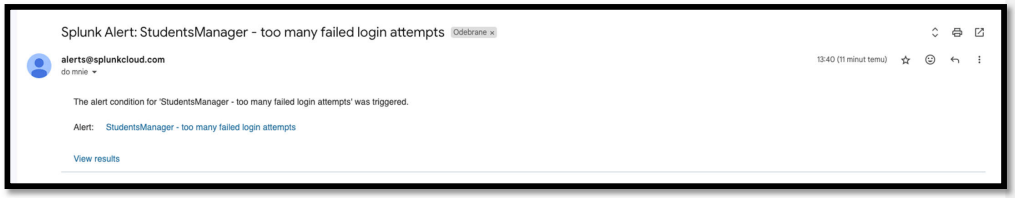
Z komentarzem [TZ8]: Brak odwołania

2.4.5 Monitorowanie za pomocą Splunk

Po wdrożeniu aplikacji, Splunk jest używany do monitorowania stanu aplikacji. Pozwala to na szybkie wykrywanie i reagowanie na wszelkie problemy w działaniu serwisu. Dla przykładu wysoka ilość nieudanych logowań w krótkim czasie może świadczyć o przeprowadzanym ataku typu brute force polegającym na podjęciu dużej ilość prób logowania do aplikacji z różną kombinacją haseł użytkownika, co powinno zaalarmować osoby odpowiedzialne za bezpieczeństwo aplikacji w celu podjęcia działań zaradczych (rysunek 14). Uruchomiony alarm w Splunk wysyła email do odpowiedniej osoby lub osób w celu powiadomienia o problemie (rysunek 15).



Rysunek 14. Wykres wyszukiwania nieudanych prób logowania przez alarm Splunk.



Rysunek 15. Email wysłany przez alarm Splunk.

3. Rezultaty

Wiele etapów wdrażania oprogramowania zostało uproszczonych i zautomatyzowanych. W tabelach 1 i 2 przedstawiono porównanie proponowanego podejścia do manualnego wdrażania aplikacji. Pod względem bezpieczeństwa automatyzacja procesów eliminuje z powodzeniem wiele zagrożeń wynikających z czynnika ludzkiego. Testy aplikacji są uruchamiane automatycznie, bez konieczności wprowadzania parametrów przez użytkownika. Nie da się ich pominąć - wszystkie muszą zakończyć się powodzeniem, by kompilacja została odpowiednio przygotowana do wdrożenia. Jeśli chodzi o szybkość działania – tutaj również można zaobserwować poprawę. Aplikacja może zostać wdrożona na środowisko testowe lub produkcyjne w chmurze w mniej niż piętnaście minut.

Tabela 1. Porównanie aspektów proponowanego rozwiązania oraz ręcznego wdrażania aplikacji.

Aspekt	Proponowane podejście	Ręczne wdrażanie aplikacji
Testy automatyczne	Pełna integracja testów automatycznych w procesie CI/CD.	Brak lub ograniczone testy automatyczne.
Automatyzacja procesów	Wykorzystanie narzędzi takich jak Jenkins, Packer, Terraform.	Duża zależność od manualnych interwencji.
	Eliminacja błędów ludzkich poprzez automatyzację powtarzalnych zadań.	Wysokie ryzyko błędów wynikających z pomyłek.
	Standaryzacja i powtarzalność procesu wdrażania.	Brak spójności w procesie wdrażania.
Szybkość działania	Średni czas wdrożenia: 12 minut i 10 sekund od momentu napisania kodu do jego wdrożenia na produkcję.	Proces wdrażania może trwać kilka godzin lub dni.
	Szybkie reagowanie na zmiany i aktualizacje.	Opóźnienia wynikające z ręcznych kroków i weryfikacji.
	Automatyczne skalowanie zasobów w chmurze.	Trudności w szybkim reagowaniu na problemy.
Koszty operacyjne	Mniejsze koszty długoterminowe dzięki automatyzacji i redukcji błędów.	Wyższe koszty związane z koniecznością angażowania personelu do ręcznych zadań.

Tabela 1. Porównanie aspektów proponowanego rozwiązania oraz ręcznego wdrażania aplikacji.

Aspekt	Proponowane podejście	Ręczne wdrażanie aplikacji
Skalowalność i dostępność	Automatyczne skalowanie w chmurze (Google Cloud Platform) zapewnia wysoką dostępność.	Skalowanie wymaga ręcznej interwencji i jest czasochłonne.
	Możliwość szybkiego dostosowania się do wzrostu liczby użytkowników.	Ryzyko przestojów przy nagłym wzroście obciążenia.
	Wbudowane mechanizmy zdrowia aplikacji (*ang. health checks*).	Brak automatycznych mechanizmów monitorujących stan aplikacji.

Bezpieczeństwo	Wczesne wykrywanie luk i błędów dzięki automatycznym testom i monitorowaniu.	Błędy bezpieczeństwa mogą pozostać niewykryte przed wdrożeniem
	Szybka reakcja na incydenty poprzez systemy alarmów	Reakcja na incydenty opóźniona ze względu na brak automatycznego monitorowania
	Izolacja środowisk testowych i produkcyjnych z zachowaniem spójności konfiguracji.	Niejednolita konfiguracja środowisk zwiększa ryzyko podatności na błędy

4. Podsumowanie

W artykule przedstawione zostało nowoczesne podejście do inżynierii oprogramowania poprzez ukazanie tworzenia procesu wdrażania aplikacji internetowej z wykorzystaniem metodologii DevOps oraz praktyk ciągłej integracji i dostarczania (CI/CD). Podkreślono znaczenie automatyzacji w eliminacji błędów ludzkich, zwiększeniu efektywności pracy zespołów programistycznych oraz zapewnieniu wysokiej jakości i stabilności oprogramowania.
Proponowane rozwiązanie opisuje architekturę potoku wdrażania, który implementuje narzędzia ciągłej integracji i wdrażania. Udowodniono, że powtarzalny i zautomatyzowany proces ma lepszy wpływ na bezpieczeństwo i szybkość wdrażania aplikacji niż manualne aktualizacje.

Z komentarzem [TZ9]: Dodałbym akapit

Artykuł zwraca uwagę na istotność skalowalności i elastyczności infrastruktury w procesie wdrażania aplikacji. Dzięki wykorzystaniu narzędzi typu infrastruktura jako kod oraz chmury obliczeniowej, możliwe jest dynamiczne dostosowywanie zasobów do aktualnych potrzeb aplikacji, co zapewnia optymalizację kosztów oraz wydajność systemu. Wpasowuje się to w nowoczesny model biznesowy przedsiębiorstw technologicznych, który zakłada optymalizację kosztów przy jednoczesnym zapewnieniu stabilności i skalowalności.

Literatura

1. D. Farley: Nowoczesna inżynieria oprogramowania. Stosowanie skutecznych technik szybkiego rozwoju oprogramowania wyższej jakości. 2023 Helion SA.
2. B. Beyer, C. Jones, J. Petoff, N. R. Murphy: Site Reliability Engineering. Jak Google zarządza systemami produkcyjnymi. 2017 Helion SA.
3. Spacelift.io, https://spacelift.io/blog/terraform-jenkins, data dostępu: 15.10.2024
4. Dev.io, https://dev.to/ismailg/the-ultimate-guide-to-building-an-efficient-cicd-pipeline-23pg, data dostępu: 15.10.2024
5. Github.com, https://github.com/ArmstrongLiwox/Implementing-CICD-Pipeline-for-Terraform-using-Jenkins, data dostępu: 15.10.2024
6. Jenkins.io, https://www.jenkins.io/doc/tutorials/tutorials-for-installing-jenkins-on-Google-Cloud/, data dostępu: 15.10.2024
7. Splunk.com, https://www.splunk.com/en_us/blog/partners/getting-started-with-splunk-on-google-cloud.html, data dostępu: 15.10.2024
8. Toxigon.com, https://toxigon.com/best-practices-ci-cd-2024, data dostępu: 18.10.2024
9. Datacamp.com, https://www.datacamp.com/blog/what-is-gcp, data dostępu: 18.10.2024
10. Divya Kumar, K.K. Mishra: The Impacts of Test Automation on Software's Cost, Quality and Time to Market.