Przedmiot: Technologie chmury obliczeniowej	*Data: 06-02-2026*
Przykład wykorzystania technologii CICD w Cloud - TCO	
Wersja doc ver 1.1	Wykonawca: Paweł Grabacki 15939

Współczesne systemy informatyczne wymagają szybkiego, powtarzalnego i niezawodnego procesu dostarczania oprogramowania. W odpowiedzi na te potrzeby coraz powszechniej stosowane są praktyki **DevOps** oraz mechanizmy **CI/CD** (Continuous Integration / Continuous Deployment), które automatyzują procesy budowania, testowania i wdrażania aplikacji.

Celem niniejszego projektu było zaprojektowanie oraz implementacja kompletnego pipeline'u **CI/CD** dla aplikacji webowej napisanej w języku **Python** z wykorzystaniem frameworka **Flask**, działającej w środowisku chmurowym **Google Cloud Platform (GCP)**. Do realizacji zadania wykorzystano narzędzia: **Git** jako system kontroli wersji, Jenkins jako serwer automatyzacji, Docker Hub jako rejestr obrazów, Docker do konteneryzacji aplikacji oraz instancję **Compute Engine (GCP)** do jej uruchomienia.

Projekt ma charakter praktyczny i demonstracyjny, a jego celem jest przedstawienie uproszczonego procesu dostarczania oprogramowania - od wprowadzenia zmiany w kodzie źródłowym aż do wdrożenia aplikacji w środowisku chmurowym, tak aby była dostępna dla użytkownika końcowego.

Uwagi do projektu:

2. Część teoretyczna

2.1 Continuous Integration i Continuous Deployment

(CI) Cointinuus integration to praktyka polegająca na częstym/automatycznym integrowaniu zmian w kodzie źródłowym do wspólnego repozytorium. Każda zmiana jest budowana i testowana, co pozwala szybko wykrywać błędy.
Continuous Deployment (CD) rozszerza CI o automatyczne wdrażanie aplikacji do środowiska docelowego, np. testowego lub produkcyjnego, bez konieczności ręcznej interwencji.

2.2 Git - system kontroli wersji

Git jest rozproszonym systemem kontroli wersji, który umożliwia:

- Śledzenie zmian w kodzie,
- Pracę zespołową,
- Łatwy rollback do poprzednich wersji,
- Integrację z narzędziami CI/CD.
- Repozytorium Git stanowi centralny punkt pipeline CI/CD.

2.3 Jenkins jako narzędzie CI/CD

Jenkins jest serwerem automatyzacji typu open-source, umożliwiającym:

- Definiowanie pipeline'ów jako kod Groovy (Jenkinsfile),
- Integrację z Git,
- Uruchamianie testów, skryptów i narzędzi zewnętrznych,
- Automatyczne wdrożenia aplikacji.
- Pipeline w Jenkinsie składa się z etapów (stages), które są wykonywane sekwencyjnie.

2.4 Dockerhub

Jest publicznym (jednym z kilku rejestrem do przechowywania utworzonych obrazów:

- Umożliwia łatwy/uniwersalny dostęp do zbudowanych obrazów
- Stanowi bardzo uproszczony system kontroli wersji dla zbudowanych obrazów

2.5 Docker i konteneryzacja

Docker umożliwia pakowanie aplikacji wraz z jej zależnościami w postaci obrazu kontenera. Dzięki temu:

- Aplikacja działa identycznie w każdym środowisku,
- Upraszcza się proces wdrożeń,
- Łatwiej skalować aplikację.
- Kontener zawiera kompletną powtarzalną konfigurację potrzebną do uruchomienia aplikacji (w tym wypadku Python/Flask.)

3. Część praktyczna

3.1 Opis aplikacji

Aplikacja webowa została napisana w Pythonie z użyciem frameworka Flask. Jej funkcją jest prezentacja prostego endpointu HTTP, który umożliwia weryfikację poprawności wdrożenia

Aplikacja udostępnia:

- Stronę główną (/) zwracającą komunikat tekstowy, numer zbudowanego/użytego obrazu.
- Endpoint zdrowia (/health) wykorzystywany do sprawdzania stanu aplikacji.

3.2 Struktura projektu

Projekt został uporządkowany w następujący sposób:

```
python-demo-app/
│   .flake8
│   Dockerfile
│   Jenkinsfile
│   LICENSE
│   README.md
│   requirements.txt
│
├──app
│       main.py
│       __init__.py
│
└──tests
         test_main.py
```

3.3 Opis pipeline CI/CD

Pipeline CI/CD realizuje następujące kroki:

1. Pobranie kodu z repozytorium Git
2. Instalacja zależności
3. Budowa obrazu Docker
4. Publikacja obrazu do Artifact Registry/Dockerhub
5. Wdrożenie/aktualizacjia aplikacji w instancji Compute Eengine GCP

4. Dokumentacja wykonania części praktycznej

4.1W ramach przygotowania środowiska wdrożeniowego w Google Cloud Platform wykonano następujące działania:

- Utworzono projekt w Google Cloud Platform
- Skonfigurowano instancję Compute Engine, na której uruchamiana jest aplikacja w kontenerze Docker,
- Utworzono konto serwisowe (service account) oraz nadano mu wymagane uprawnienia do zarządzania zasobami Compute Engine,
- Wygenerowano klucz dostępu w formacie JSON, wykorzystywany w Jenkins jako poświadczenie,
- Do automatyzacji operacji administracyjnych wykorzystano narzędzie Google Cloud CLI (gcloud), które umożliwia autoryzację oraz tworzenie/aktualizację instancji i reguł firewall w ramach pipeline.

4.2 Jenkins został skonfigurowany do realizacji procesu CI/CD poprzez:

- Pobieranie kodu z repozytorium Git (GitHub)
- Przechowywanie danych uwierzytelniających (Docker Hub oraz GCP),
- Budowanie, tagowanie i publikację obrazów Docker,
- Zarządzanie wdrożeniem w Google Cloud Platform za pomocą narzędzia **gcloud CLI**,
- Uruchamianie pipeline zdefiniowanego w pliku **Jenkinsfile** umieszczonym w repozytorium projektu (na osobnym branchu).

4.3 Budowa i publikacja obrazu Docker

Podczas wykonywania pipeline:

- Aplikacja jest pakowana w obraz Docker,
- Obraz otrzymuje unikalny tag BUILD NUMBER, (numer ten jest wyświetlany w gotowej aplikacji)
- Obraz jest wysyłany do rejestru Dockerhub. , z którego następnie pobierany jest podczas wdrożenia na instancję GCE.

4.4 Wdrożenie do GCP (Google Cloud Platform)

Wdrożenie aplikacji odbywa się poprzez:

- Utworzenie reguły firewall umożliwiającej ruch przychodzący na porcie 80 (jeżeli nie istnieje),
- Utworzenie instancji **Compute Engine** uruchamiającej kontener Docker (jeżeli instancja jeszcze nie istnieje),
- Aktualizację uruchomionego kontenera na instancji Compute Engine do nowej wersji obrazu (w przypadku Kolejnych wdrożeń),
- Wyświetlenie zewnętrznego adresu IP instancji, umożliwiającego dostęp do aplikacji z poziomu przeglądarki.

5. Wnioski końcowe

W ramach projektu zrealizowano kompletny proces **CI/CD** dla aplikacji webowej napisanej w **Python/Flask.** Konfiguracja oparta na Jenkins umożliwiła automatyczne pobieranie kodu z repozytorium Git, budowanie obrazu Docker oraz publikację gotowej wersji aplikacji do repozytorium Docker Hub.

Wdrożenie do Google **Cloud Platform** zostało zautomatyzowane z wykorzystaniem narzędzia **Google Cloud CLI** (gcloud), co pozwoliło na tworzenie i aktualizację instancji **Compute Engine** uruchamiającej aplikację w kontenerze. Zastosowanie unikalnych tagów dla kolejnych buildów zwiększyło kontrolę nad wersjonowaniem oraz umożliwiło łatwiejszą identyfikację wdrożonej wersji aplikacji.

Przyjęte rozwiązanie jest proste w utrzymaniu, a jednocześnie spełnia podstawowe wymagania procesu wdrożeniowego. Dodatkowo stanowi dobrą bazę do dalszego rozwoju projektu, np. poprzez rozbudowę monitoringu, testów automatycznych, użycie lintera lub migrację wdrożenia do środowiska Kubernetes (GKE).

5.1 Repozytoria użyte w projekcie

- pawelgrabacki/python-demo-app
- docker/pawelgrabacki/python-demo-app