

# Kompletny Podręcznik DevOps

Git • Jenkins • Python • Terraform • AWS • Kubernetes

---

## Spis Treści

### Część I: Podstawy DevOps i Git

1. [Wprowadzenie do DevOps](#)
2. [Git - Podstawy](#)
3. [Git - Zaawansowane funkcje](#)
4. [Najlepsze praktyki Git w DevOps](#)

### Część II: Continuous Integration/Continuous Deployment

5. [Wprowadzenie do CI/CD](#)
6. [Jenkins - Instalacja i konfiguracja](#)
7. [Tworzenie pipeline'ów Jenkins](#)
8. [Integracja Jenkins z Git](#)

### Część III: Python w DevOps

9. [Python dla DevOps - podstawy](#)
10. [Automatyzacja z Python](#)
11. [Testowanie aplikacji Python](#)
12. [Skrypty DevOps w Python](#)

### Część IV: Infrastructure as Code

13. [Wprowadzenie do Infrastructure as Code](#)
14. [Terraform - Instalacja i podstawy](#)
15. [Tworzenie infrastruktury AWS z Terraform](#)
16. [Zarządzanie stanem Terraform](#)

### Część V: Amazon Web Services

17. [Wprowadzenie do AWS](#)

- 18. [Kluczowe usługi AWS dla DevOps](#)
- 19. [Bezpieczeństwo w AWS](#)
- 20. [Monitoring i logging w AWS](#)

## **Część VI: Kubernetes w AWS**

- 21. [Wprowadzenie do Kubernetes](#)
- 22. [Amazon EKS - konfiguracja](#)
- 23. [Deployment aplikacji na EKS](#)
- 24. [Zarządzanie klastrem Kubernetes](#)

## **Część VII: Projekt praktyczny**

- 25. [Projekt end-to-end](#)
  - 26. [Konfiguracja kompletnego pipeline'u](#)
  - 27. [Deployment i monitoring](#)
  - 28. [Najlepsze praktyki i optymalizacja](#)
- 

# **Wymagania wstępne**

Przed rozpoczęciem pracy z tym podręcznikiem upewnij się, że masz:

- Komputer z systemem Linux, macOS lub Windows (z WSL2)
- Dostęp do internetu
- Konto AWS (z możliwością tworzenia zasobów)
- Podstawową znajomość linii poleceń
- Edytor tekstu lub IDE (np. VS Code, PyCharm)

# **Struktura podręcznika**

Każdy rozdział zawiera:

- **Teorię** - wyjaśnienie koncepcji i narzędzi

- **Instrukcje krok po kroku** - dokładne polecenia do wykonania
  - **Przykłady praktyczne** - rzeczywiste scenariusze użycia
  - **Zadania do wykonania** - ćwiczenia utrwalające wiedzę
  - **Rozwiązywanie problemów** - najczęstsze błędy i ich rozwiązania
- 

# 1. Wprowadzenie do DevOps

## Czym jest DevOps?

DevOps to filozofia, kultura i zestaw praktyk, które łączą rozwój oprogramowania (Development) z operacjami IT (Operations). Głównym celem DevOps jest skrócenie cyklu życia rozwoju systemów przy jednoczesnym dostarczaniu funkcji, poprawek i aktualizacji w wysokiej jakości i z dużą częstotliwością, zgodnie z celami biznesowymi.

Termin "DevOps" został po raz pierwszy użyty przez Patricka Debois w 2009 roku podczas pierwszej konferencji DevOpsDays w Belgii. Od tego czasu DevOps ewoluował z prostej idei współpracy między zespołami rozwoju i operacji w kompleksowy zestaw praktyk, narzędzi i filozofii, które fundamentalnie zmieniły sposób, w jaki organizacje dostarczają oprogramowanie.

Tradycyjnie zespoły rozwoju i operacji działały w silosach. Deweloperzy koncentrowali się na tworzeniu nowych funkcji i szybkim dostarczaniu kodu, podczas gdy zespoły operacyjne priorytetowo traktowały stabilność i niezawodność systemów produkcyjnych. Ta separacja często prowadziła do konfliktów, opóźnień i problemów z jakością. DevOps przełamuje te bariery, promując współpracę, komunikację i wspólną odpowiedzialność za cały cykl życia aplikacji.

## Kluczowe zasady DevOps

### Współpraca i komunikacja

Fundamentem DevOps jest przełamanie silosów organizacyjnych i stworzenie kultury współpracy między wszystkimi zespołami zaangażowanymi w dostarczanie oprogramowania. Oznacza to nie tylko współpracę między deweloperami a administratorami systemów, ale także włączenie zespołów QA, bezpieczeństwa, architektów i innych interesariuszy w jeden spójny proces.

Efektywna komunikacja w środowisku DevOps wymaga ustanowienia jasnych kanałów komunikacji, regularnych spotkań między zespołami i wspólnych narzędzi do śledzenia postępów i problemów. Zespoły muszą dzielić się wiedzą, doświadczeniami i odpowiedzialnością za sukces całego projektu, a nie tylko za swoje indywidualne obszary.

## Automatyzacja

Automatyzacja jest sercem DevOps i obejmuje wszystkie aspekty cyklu życia oprogramowania, od kompilacji i testowania po deployment i monitoring. Celem automatyzacji jest eliminacja powtarzalnych, manualnych zadań, które są podatne na błędy ludzkie i spowalniają proces dostarczania oprogramowania.

Kluczowe obszary automatyzacji w DevOps obejmują:

**Continuous Integration (CI)** - automatyczne kompilowanie, testowanie i integrowanie zmian w kodzie. Każda zmiana w repozytorium kodu automatycznie uruchamia proces budowania i testowania, co pozwala na szybkie wykrycie problemów.

**Continuous Deployment (CD)** - automatyczne wdrażanie aplikacji do środowisk testowych i produkcyjnych po pomyślnym przejściu przez wszystkie etapy pipeline'u CI/CD.

**Infrastructure as Code (IaC)** - zarządzanie infrastrukturą poprzez kod, co pozwala na wersjonowanie, testowanie i automatyczne wdrażanie zmian w infrastrukturze.

**Automatyzacja testów** - uruchamianie testów jednostkowych, integracyjnych i end-to-end w sposób automatyczny, co zapewnia wysoką jakość kodu i szybkie feedback.

## Monitoring i feedback

Ciągły monitoring aplikacji i infrastruktury jest kluczowy dla sukcesu DevOps. Monitoring nie ogranicza się tylko do sprawdzania, czy aplikacja działa, ale obejmuje także zbieranie

metryk wydajności, analizę logów, śledzenie doświadczeń użytkowników i monitorowanie bezpieczeństwa.

Efektywny monitoring w DevOps charakteryzuje się:

**Proaktywnym podejściem** - systemy monitoringu powinny wykrywać problemy zanim wpłyną na użytkowników końcowych.

**Kompleksowością** - monitoring powinien obejmować wszystkie warstwy aplikacji, od infrastruktury przez aplikację po doświadczenie użytkownika.

**Actionable insights** - zebrane dane muszą być przekształcane w konkretne działania i rekomendacje dla zespołów.

**Kulturą uczenia się** - błędy i incydenty powinny być traktowane jako okazje do nauki i poprawy procesów, a nie jako powody do obwiniania.

## Korzyści z wdrożenia DevOps

### Szybsze dostarczanie oprogramowania

Organizacje wdrażające praktyki DevOps doświadczają znacznego przyspieszenia w dostarczaniu nowych funkcji i poprawek. Według raportu "State of DevOps" z 2021 roku, organizacje o wysokiej wydajności DevOps wdrażają kod 973 razy częściej niż organizacje o niskiej wydajności, przy jednoczesnym skróceniu czasu od commitu do wdrożenia z miesięcy do godzin lub nawet minut.

To przyspieszenie wynika z automatyzacji procesów, eliminacji manualnych kroków zatwierdzania i lepszej współpracy między zespołami. Gdy procesy są zautomatyzowane i dobrze zdefiniowane, zespoły mogą skupić się na tworzeniu wartości biznesowej zamiast na powtarzalnych zadaniach operacyjnych.

### Wyższa jakość oprogramowania

Paradoksalnie, mimo szybszego tempa dostarczania, DevOps prowadzi do wyższej jakości oprogramowania. Dzieje się tak dzięki:

**Automatyzacji testów** - każda zmiana w kodzie jest automatycznie testowana na różnych poziomach, co pozwala na wczesne wykrycie błędów.

**Continuous feedback** - szybkie feedback z środowisk testowych i produkcyjnych pozwala na natychmiastowe reagowanie na problemy.

**Małe, częste zmiany** - zamiast dużych, rzadkich wydań, DevOps promuje małe, inkrementalne zmiany, które są łatwiejsze do testowania i debugowania.

**Shift-left testing** - testowanie przesuwane się w lewo w cyklu rozwoju, co oznacza wcześniejsze wykrywanie i naprawianie błędów.

## Lepsza współpraca zespołowa

DevOps fundamentalnie zmienia sposób, w jaki zespoły współpracują. Zamiast działac w izolacji, zespoły rozwoju, operacji, QA i bezpieczeństwa pracują razem nad wspólnymi celami. Ta współpraca prowadzi do:

**Lepszego zrozumienia** - deweloperzy lepiej rozumieją wyzwania operacyjne, podczas gdy zespoły operacyjne lepiej rozumieją wymagania biznesowe.

**Wspólnej odpowiedzialności** - wszyscy członkowie zespołu czują się odpowiedzialni za sukces całego produktu, a nie tylko za swój fragment.

**Szybszego rozwiązywania problemów** - gdy problemy wystąpią, zespoły mogą szybko współpracować nad ich rozwiązaniem, wykorzystując wiedzę z różnych obszarów.

## Zwiększona niezawodność

Mimo szybszego tempa zmian, organizacje DevOps osiągają wyższą niezawodność systemów. Według badań, organizacje o wysokiej wydajności DevOps mają 2,6 razy niższy wskaźnik awarii i 2,6 razy szybszy czas odzyskiwania po awarii.

Ta zwiększona niezawodność wynika z:

**Lepszego monitoringu** - systemy są lepiej monitorowane, co pozwala na szybsze wykrycie problemów.

**Automatyzacji odzyskiwania** - procesy odzyskiwania po awarii są zautomatyzowane i regularnie testowane.

**Infrastruktury jako kodu** - infrastruktura jest wersjonowana i może być szybko odtworzona w przypadku problemów.

**Kultury uczenia się** - organizacje uczą się z każdego incydentu i poprawiają swoje procesy.

## Wyzwania we wdrażaniu DevOps

### Zmiana kultury organizacyjnej

Największym wyzwaniem we wdrażaniu DevOps jest zmiana kultury organizacyjnej. DevOps wymaga przejścia od kultury silosów i obwiniania do kultury współpracy i wspólnej odpowiedzialności. Ta zmiana może być szczególnie trudna w dużych, tradycyjnych organizacjach z ugruntowanymi strukturami i procesami.

Kluczowe aspekty zmiany kulturowej obejmują:

**Przywództwo** - liderzy muszą aktywnie wspierać i promować praktyki DevOps, dając przykład współpracy i otwartości na zmiany.

**Edukacja i szkolenia** - zespoły muszą być edukowane nie tylko w zakresie nowych narzędzi, ale także nowych sposobów myślenia i współpracy.

**Stopniowe wprowadzanie zmian** - radykalne zmiany mogą spotkać się z oporem, dlatego lepiej wprowadzać DevOps stopniowo, zaczynając od małych projektów pilotazowych.

**Mierzenie i komunikowanie sukcesu** - ważne jest mierzenie postępów i komunikowanie sukcesów, aby budować momentum dla dalszych zmian.

### Kompleksowość narzędzi

Ekosystem narzędzi DevOps jest bardzo bogaty i stale ewoluuje. Wybór odpowiednich narzędzi i ich integracja może być przytłaczająca, szczególnie dla organizacji dopiero rozpoczynających swoją podróż z DevOps.

Główne kategorie narzędzi DevOps obejmują:

**Kontrola wersji** - Git, SVN, Mercurial

**CI/CD** - Jenkins, GitLab CI, GitHub Actions, Azure DevOps

**Konteneryzacja** - Docker, Podman, containerd

**Orkiestracja** - Kubernetes, Docker Swarm, OpenShift

**Infrastructure as Code** - Terraform, CloudFormation, Ansible

**Monitoring** - Prometheus, Grafana, ELK Stack, Datadog

**Bezpieczeństwo** - SonarQube, OWASP ZAP, Aqua Security

Kluczem do sukcesu jest rozpoczęcie od podstawowych narzędzi i stopniowe rozszerzanie toolchain'u w miarę dojrzewania praktyk DevOps w organizacji.

## Bezpieczeństwo

Szybkie tempo zmian w DevOps może stwarzać wyzwania dla bezpieczeństwa. Tradycyjne podejścia do bezpieczeństwa, które polegają na manualnych przeglądach i długich procesach zatwierdzania, nie są kompatybilne z szybkim tempem DevOps.

Rozwiązaniem jest DevSecOps - podejście, które integruje bezpieczeństwo w każdy etap pipeline'u DevOps. Kluczowe praktyki DevSecOps obejmują:

**Security as Code** - polityki bezpieczeństwa są definiowane jako kod i automatycznie egzekwowane.

**Automated security testing** - testy bezpieczeństwa są zautomatyzowane i uruchamiane w ramach pipeline'u CI/CD.

**Continuous compliance** - zgodność z regulacjami jest monitorowana w sposób ciągły, a nie tylko podczas audytów.

**Security monitoring** - systemy bezpieczeństwa są zintegrowane z narzędziami monitoringu i alertowania.

## Metryki DevOps

Pomiar postępów w DevOps jest kluczowy dla sukcesu. Istnieją cztery kluczowe metryki DevOps, znane jako "Four Keys", które zostały zidentyfikowane przez Google w ramach badań DORA (DevOps Research and Assessment):



## Lead Time for Changes

Lead Time for Changes mierzy czas od momentu, gdy kod jest zacommitowany do repozytorium, do momentu, gdy jest wdrożony na produkcji. Ta metryka pokazuje, jak szybko organizacja może dostarczać wartość biznesową.

Organizacje o wysokiej wydajności mają Lead Time for Changes mierzony w godzinach lub dniach, podczas gdy organizacje o niskiej wydajności mogą mieć Lead Time mierzony w tygodniach lub miesiącach.

## Deployment Frequency

Deployment Frequency mierzy, jak często organizacja wdraża kod na produkcję. Wyzsza częstotliwość wdrożeń zazwyczaj koreluje z lepszą wydajnością DevOps i większą zdolnością do szybkiego reagowania na zmiany biznesowe.

Organizacje o wysokiej wydajności wdrażają kod wiele razy dziennie, podczas gdy organizacje o niskiej wydajności mogą wdrażać kod raz na kilka miesięcy.

## Mean Time to Recovery (MTTR)

MTTR mierzy średni czas potrzebny do odzyskania sprawności systemu po awarii. Ta metryka jest kluczowa, ponieważ awarie są nieuniknione, ale szybkość odzyskiwania może znacząco wpłynąć na doświadczenie użytkowników i koszty biznesowe.

Organizacje o wysokiej wydajności mają MTTR mierzony w godzinach, podczas gdy organizacje o niskiej wydajności mogą potrzebować dni lub tygodni na odzyskanie sprawności.

## Change Failure Rate

Change Failure Rate mierzy procent wdrożeń, które powodują awarie lub wymagają natychmiastowych poprawek. Ta metryka pokazuje jakość procesów DevOps i efektywność testowania.

Organizacje o wysokiej wydajności mają Change Failure Rate poniżej 15%, podczas gdy organizacje o niskiej wydajności mogą mieć wskaźnik powyżej 30%.

# Dojrzałość DevOps

Wdrażanie DevOps to proces ewolucyjny, który można podzielić na kilka poziomów dojrzałości:

## Poziom 1: Początkowy

Na tym poziomie organizacja dopiero zaczyna swoją podróż z DevOps. Charakteryzuje się:

- Manualnymi procesami deployment'u
- Ograniczoną automatyzacją
- Silosami między zespołami
- Rzadkimi wydaniem
- Reaktywnym podejściem do problemów

## Poziom 2: Zarządzany

Organizacja zaczyna wprowadzać podstawowe praktyki DevOps:

- Podstawowa automatyzacja CI/CD
- Regularne wydania
- Początkowa współpraca między zespołami
- Podstawowy monitoring
- Standardowe procesy

## Poziom 3: Zdefiniowany

Praktyki DevOps są dobrze zdefiniowane i konsekwentnie stosowane:

- Zaawansowana automatyzacja
- Infrastructure as Code

- Comprehensive testing
- Proaktywny monitoring
- Kultura współpracy

## Poziom 4: Kwantytatywnie zarządzany

Organizacja używa metryk do zarządzania i optymalizacji procesów:

- Metryki wydajności są regularnie mierzone
- Procesy są optymalizowane na podstawie danych
- Predictive analytics
- Zaawansowane praktyki bezpieczeństwa

## Poziom 5: Optymalizujący

Organizacja ciągle się uczy i optymalizuje:

- Continuous improvement
- Innowacyjne praktyki
- Eksperymentowanie
- Sharing knowledge z społecznością

# 2. Git - Podstawy

## Wprowadzenie do Git

Git to rozproszony system kontroli wersji stworzony przez Linusa Torvaldsa w 2005 roku do zarządzania kodem źródłowym jądra Linux. Od tego czasu Git stał się de facto standardem w branży IT i jest fundamentalnym narzędziem w każdym środowisku DevOps.

W przeciwieństwie do scentralizowanych systemów kontroli wersji, takich jak SVN czy CVS, Git jest systemem rozproszonym. Oznacza to, że każdy deweloper ma pełną kopię całej

historii projektu na swoim lokalnym komputerze. Ta architektura zapewnia większą elastyczność, niezawodność i możliwość pracy offline.

Git przechowuje dane jako serie migawek (snapshots) całego systemu plików, a nie jako zmiany względem poprzedniej wersji. Każda migawka zawiera referencje do wszystkich plików w projekcie w danym momencie. Jeśli plik nie zmienił się między migawkami, Git przechowuje tylko referencję do poprzedniej wersji pliku.

## Instalacja Git

### Linux (Ubuntu/Debian)

Otwórz terminal i wykonaj następujące polecenia:

Bash

```
# Aktualizacja listy pakietów
sudo apt update

# Instalacja Git
sudo apt install git

# Weryfikacja instalacji
git --version
```

### Linux (CentOS/RHEL/Fedora)

Bash

```
# CentOS/RHEL
sudo yum install git

# Fedora
sudo dnf install git

# Weryfikacja instalacji
git --version
```

### macOS

Najłatwiejszym sposobem instalacji Git na macOS jest użycie Homebrew:

Bash

```
# Instalacja Homebrew (jeśli nie jest zainstalowane)
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

# Instalacja Git
brew install git

# Weryfikacja instalacji
git --version
```

Alternatywnie, możesz pobrać installer z oficjalnej strony Git lub zainstalować Xcode Command Line Tools:

Bash

```
xcode-select --install
```

## Windows

1. Pobierz installer z oficjalnej strony: <https://git-scm.com/download/win>
2. Uruchom pobrany plik .exe
3. Postępuj zgodnie z instrukcjami instalatora (zalecane są domyślne ustawienia)
4. Po instalacji otwórz Git Bash lub Command Prompt i sprawdź instalację:

Bash

```
git --version
```

## Pierwsza konfiguracja Git

Po instalacji Git należy skonfigurować podstawowe informacje, które będą używane w commitach:

---

Bash

```
# Ustawienie nazwy użytkownika
git config --global user.name "Twoje Imię Nazwisko"

# Ustawienie adresu email
git config --global user.email "twoj.email@example.com"

# Ustawienie domyślnego edytora (opcjonalne)
git config --global core.editor "code --wait" # VS Code
# lub
git config --global core.editor "nano" # Nano
# lub
git config --global core.editor "vim" # Vim

# Ustawienie domyślnej nazwy głównej gałęzi
git config --global init.defaultBranch main

# Sprawdzenie konfiguracji
git config --list
```

Flaga `--global` oznacza, że ustawienia będą stosowane do wszystkich repozytoriów Git na tym komputerze. Możesz także skonfigurować ustawienia tylko dla konkretnego repozytorium, pomijając flagę `--global`.

## Podstawowe koncepcje Git

### Repository (Repozytorium)

Repozytorium Git to katalog zawierający wszystkie pliki projektu oraz całą historię zmian. Repozytorium może być lokalne (na Twoim komputerze) lub zdalne (na serwerze, np. GitHub, GitLab).

### Working Directory (Katalog roboczy)

To katalog na Twoim komputerze, w którym aktualnie pracujesz nad plikami projektu. Zawiera aktualne wersje plików, które możesz edytować.

### Staging Area (Index)

Staging Area to pośredni obszar między katalogiem roboczym a repozytorium. Pliki dodane do Staging Area są przygotowane do następnego commita.

## Commit

Commit to migawka stanu projektu w określonym momencie. Każdy commit ma unikalny identyfikator (hash SHA-1) i zawiera informacje o autorze, dacie oraz opisie zmian.

## Branch (Gałąź)

Branch to niezależna linia rozwoju w projekcie. Domyślna gałąź nazywa się zwykle `main` lub `master`. Gałęzie pozwalają na równoległą pracę nad różnymi funkcjami.

## HEAD

HEAD to wskaźnik na aktualnie aktywną gałąź i commit. Pokazuje, gdzie jesteś w historii projektu.

# Tworzenie pierwszego repozytorium

## Inicjalizacja nowego repozytorium

Bash

```
# Utwórz nowy katalog dla projektu
mkdir moj-pierwszy-projekt
cd moj-pierwszy-projekt

# Inicjalizuj repozytorium Git
git init

# Sprawdź status repozytorium
git status
```

Po wykonaniu `git init` zostanie utworzony ukryty katalog `.git`, który zawiera wszystkie metadane i historię repozytorium.

## Tworzenie pierwszego pliku i commita

---

Bash

```
# Utwórz plik README.md
echo "# Mój pierwszy projekt" > README.md

# Sprawdź status
git status

# Dodaj plik do Staging Area
git add README.md

# Sprawdź status ponownie
git status

# Wykonaj pierwszy commit
git commit -m "Dodaj plik README.md"

# Sprawdź historię commitów
git log
```

## Klonowanie istniejącego repozytorium

Bash

```
# Klonowanie repozytorium z GitHub
git clone https://github.com/username/repository-name.git

# Klonowanie do konkretnego katalogu
git clone https://github.com/username/repository-name.git moj-katalog

# Klonowanie konkretnej gałęzi
git clone -b branch-name https://github.com/username/repository-name.git
```

## Podstawowe operacje Git

### Sprawdzanie statusu

Bash

```
# Sprawdź status repozytorium
git status
```



```
# Krótki format statusu
git status -s
```

Status pokazuje:

- Pliki zmodyfikowane (modified)
- Pliki dodane do Staging Area (staged)
- Pliki niesłędzone (untracked)

## Dodawanie plików do Staging Area

Bash

```
# Dodaj konkretny plik
git add nazwa-pliku.txt

# Dodaj wszystkie pliki w katalogu
git add .

# Dodaj wszystkie pliki z rozszerzeniem .py
git add *.py

# Dodaj wszystkie pliki w konkretnym katalogu
git add src/

# Interaktywne dodawanie
git add -i

# Dodawanie fragmentów pliku
git add -p nazwa-pliku.txt
```

## Wykonywanie commitów

Bash

```
# Commit z wiadomością
git commit -m "Opis zmian"

# Commit z dłuższym opisem
git commit -m "Krótki opis" -m "Dłuższy opis zmian"
```

```
# Commit wszystkich zmodyfikowanych plików (pomija nowe pliki)
git commit -am "Opis zmian"

# Modyfikacja ostatniego commita
git commit --amend -m "Nowy opis"

# Commit bez wiadomości (otworzy edytor)
git commit
```

## Przeglądanie historii

Bash

```
# Podstawowa historia
git log

# Kompaktowa historia
git log --oneline

# Historia z grafem gałęzi
git log --graph --oneline --all

# Historia ostatnich 5 commitów
git log -5

# Historia z różnicami
git log -p

# Historia konkretnego pliku
git log nazwa-pliku.txt

# Historia z filtrowaniem po autorze
git log --author="Imię Nazwisko"

# Historia z filtrowaniem po dacie
git log --since="2023-01-01" --until="2023-12-31"
```

## Sprawdzanie różnic

Bash

```
# Różnice między katalogiem roboczym a Staging Area
git diff
```

```
# Różnice między Staging Area a ostatnim commitem
git diff --staged

# Różnice między dwoma commitami
git diff commit1 commit2

# Różnice w konkretnym pliku
git diff nazwa-pliku.txt

# Różnice między gałęziami
git diff branch1 branch2
```

## Praca z gałęziami

### Tworzenie i przełączanie gałęzi

Bash

```
# Lista gałęzi
git branch

# Lista wszystkich gałęzi (lokalnych i zdalnych)
git branch -a

# Tworzenie nowej gałęzi
git branch nazwa-galezi

# Przełączenie na gałąź
git checkout nazwa-galezi

# Tworzenie i przełączenie na nową gałąź
git checkout -b nazwa-galezi

# Nowsza składnia (Git 2.23+)
git switch nazwa-galezi
git switch -c nazwa-galezi
```

### Łączenie gałęzi

Bash

```
# Przełącz na gałąź docelową (np. main)
git checkout main
```

```
# Połącz gałąź
git merge nazwa-galezi

# Merge z utworzeniem merge commit
git merge --no-ff nazwa-galezi

# Merge z squash (wszystkie commity z gałęzi jako jeden)
git merge --squash nazwa-galezi
```

## Usuwanie gałęzi

Bash

```
# Usuń gałąź (bezpieczne - tylko jeśli została zmergowana)
git branch -d nazwa-galezi

# Wymuś usunięcie gałęzi
git branch -D nazwa-galezi

# Usuń zdalną gałąź
git push origin --delete nazwa-galezi
```

## Praca ze zdalnymi repozytoriami

### Dodawanie zdalnych repozytoriów

Bash

```
# Dodaj zdalne repozytorium
git remote add origin https://github.com/username/repository.git

# Lista zdalnych repozytoriów
git remote -v

# Szczegóły zdalnego repozytorium
git remote show origin

# Zmiana URL zdalnego repozytorium
git remote set-url origin https://github.com/username/new-repository.git
```

## Pobieranie zmian

Bash

```
# Pobierz zmiany bez mergowania  
git fetch origin
```

```
# Pobierz i zmerguj zmiany  
git pull origin main
```

```
# Pull z rebase  
git pull --rebase origin main
```

## Wysyłanie zmian

Bash

```
# Wyślij zmiany do zdalnego repozytorium  
git push origin main
```

```
# Wyślij nową gałąź  
git push -u origin nazwa-galezi
```

```
# Wyślij wszystkie gałęzie  
git push --all origin
```

```
# Wyślij tagi  
git push --tags origin
```

## Cofanie zmian

### Cofanie zmian w katalogu roboczym

Bash

```
# Cofnij zmiany w konkretnym pliku  
git checkout -- nazwa-pliku.txt
```

```
# Cofnij wszystkie zmiany w katalogu roboczym  
git checkout -- .
```

```
# Nowsza składnia
```

```
git restore nazwa-pliku.txt
git restore .
```

## Usuwanie plików ze Staging Area

Bash

```
# Usuń plik ze Staging Area (pozostaw w katalogu roboczym)
git reset HEAD nazwa-pliku.txt

# Nowsza składnia
git restore --staged nazwa-pliku.txt
```

## Cofanie commitów

Bash

```
# Cofnij ostatni commit (zachowaj zmiany w katalogu roboczym)
git reset --soft HEAD~1

# Cofnij ostatni commit (zachowaj zmiany w katalogu roboczym, usuń ze Staging Area)
git reset HEAD~1

# Cofnij ostatni commit (usuń wszystkie zmiany)
git reset --hard HEAD~1

# Cofnij do konkretnego commita
git reset --hard commit-hash

# Utwórz nowy commit cofający zmiany (bezpieczniejsze)
git revert HEAD
git revert commit-hash
```

## Praca z tagami

Tagi służą do oznaczania ważnych punktów w historii projektu, takich jak wydania (releases).

Bash

```
# Lista tagów
git tag

# Tworzenie lekkiego tagu
git tag v1.0.0

# Tworzenie tagu z adnotacją
git tag -a v1.0.0 -m "Wersja 1.0.0"

# Tagowanie konkretnego commita
git tag -a v1.0.0 commit-hash -m "Wersja 1.0.0"

# Wyświetlanie informacji o tagu
git show v1.0.0

# Wysyłanie tagów do zdalnego repozytorium
git push origin v1.0.0
git push origin --tags

# Usuwanie tagu
git tag -d v1.0.0
git push origin --delete v1.0.0
```

## Plik .gitignore

Plik `.gitignore` określa, które pliki i katalogi Git powinien ignorować

Bash

```
# Utwórz plik .gitignore
touch .gitignore
```

Przykładowa zawartość pliku `.gitignore` :

Plain Text

```
# Pliki tymczasowe
*.tmp
*.log
*.swp

# Katalogi
node_modules/
```

```
dist/  
build/  
.vscode/  
  
# Pliki systemowe  
.DS_Store  
Thumbs.db  
  
# Pliki środowiska  
.env  
.env.local  
  
# Pliki IDE  
*.iml  
.idea/  
  
# Pliki Python  
__pycache__/  
*.pyc  
*.pyo  
*.egg-info/  
venv/  
.pytest_cache/  
  
# Pliki Java  
*.class  
target/  
  
# Pliki C/C++  
*.o  
*.exe  
*.dll
```

## Praktyczne ćwiczenie

Wykonaj następujące kroki, aby przećwiczyć podstawowe operacje Git:

### Krok 1: Przygotowanie środowiska

Bash

```
# Utwórz katalog dla ćwiczenia  
mkdir git-cwiczenie  
cd git-cwiczenie
```



```
# Inicjalizuj repozytorium
git init

# Sprawdź status
git status
```

## Krok 2: Tworzenie pierwszych plików

Bash

```
# Utwórz plik README.md
echo "# Projekt ćwiczeniowy Git" > README.md

# Utwórz plik main.py
cat > main.py << EOF
def hello_world():
    print("Hello, World!")

if __name__ == "__main__":
    hello_world()
EOF

# Sprawdź status
git status
```

## Krok 3: Pierwszy commit

Bash

```
# Dodaj pliki do Staging Area
git add README.md main.py

# Sprawdź status
git status

# Wykonaj commit
git commit -m "Dodaj podstawowe pliki projektu"

# Sprawdź historię
git log --oneline
```

## Krok 4: Modyfikacja plików

Bash

```
# Zmodyfikuj plik main.py
cat > main.py << EOF
def hello_world():
    print("Hello, World!")

def goodbye_world():
    print("Goodbye, World!")

if __name__ == "__main__":
    hello_world()
    goodbye_world()
EOF

# Sprawdź różnice
git diff

# Dodaj i commituj zmiany
git add main.py
git commit -m "Dodaj funkcję goodbye_world"
```

## Krok 5: Praca z gałęziami

Bash

```
# Utwórz nową gałąź
git checkout -b feature/new-function

# Dodaj nowy plik
cat > utils.py << EOF
def calculate_sum(a, b):
    return a + b

def calculate_product(a, b):
    return a * b
EOF

# Commituj nowy plik
git add utils.py
git commit -m "Dodaj funkcje matematyczne"

# Przełącz na główną gałąź
git checkout main

# Sprawdź, czy plik utils.py istnieje
```

```
ls -la

# Przełącz z powrotem na gałąź feature
git checkout feature/new-function

# Sprawdź ponownie
ls -la
```

## Krok 6: Łączenie gałęzi

Bash

```
# Przełącz na główną gałąź
git checkout main

# Połącz gałąź feature
git merge feature/new-function

# Sprawdź historię
git log --oneline --graph

# Usuń gałąź feature
git branch -d feature/new-function
```

## Krok 7: Symulacja konfliktu

Bash

```
# Utwórz dwie gałęzie z konfliktującymi zmianami
git checkout -b branch1
echo "Zmiana z branch1" >> README.md
git add README.md
git commit -m "Zmiana z branch1"

git checkout main
git checkout -b branch2
echo "Zmiana z branch2" >> README.md
git add README.md
git commit -m "Zmiana z branch2"

# Spróbuj zmergować branch1 do branch2
git merge branch1

# Rozwiąż konflikt ręcznie
```

```
# Edytuj plik README.md, usuń markery konfliktu
# Następnie:
git add README.md
git commit -m "Rozwiąż konflikt merge"
```

To ćwiczenie wprowadza Cię w podstawowe operacje Git, które będą fundamentem dla bardziej zaawansowanych technik DevOps opisanych w kolejnych rozdziałach.

## 3. Git - Zaawansowane funkcje

### Git Rebase

Rebase to jedna z najważniejszych i jednocześnie najbardziej kontrowersyjnych funkcji Git. Pozwala na przepisanie historii commitów, co może być bardzo przydatne do utrzymania czystej i linearnej historii projektu.

#### Podstawy rebase

Bash

```
# Podstawowy rebase na główną gałąź
git checkout feature-branch
git rebase main

# Interaktywny rebase ostatnich 3 commitów
git rebase -i HEAD~3

# Rebase z konkretnego commita
git rebase -i commit-hash
```

#### Interaktywny rebase

Interaktywny rebase pozwala na modyfikację historii commitów. Podczas interaktywnego rebase możesz:

- **pick** - zachować commit bez zmian
- **reword** - zmienić wiadomość commita

- **edit** - zatrzymać się na commicie do modyfikacji
- **squash** - połączyć commit z poprzednim
- **fixup** - jak squash, ale odrzucić wiadomość commita
- **drop** - usunąć commit

Bash

```
# Przykład interaktywnego rebase
git rebase -i HEAD~4

# W edytorze zobaczysz coś takiego:
# pick 1234567 Dodaj funkcję A
# pick 2345678 Popraw błąd w funkcji A
# pick 3456789 Dodaj funkcję B
# pick 4567890 Aktualizuj dokumentację

# Możesz zmienić na:
# pick 1234567 Dodaj funkcję A
# squash 2345678 Popraw błąd w funkcji A
# pick 3456789 Dodaj funkcję B
# reword 4567890 Aktualizuj dokumentację
```

## Rebase vs Merge

**Merge** zachowuje historię i tworzy merge commit:

Bash

```
git checkout main
git merge feature-branch
```

**Rebase** przepisuje historię i tworzy linearną linię commitów:

Bash

```
git checkout feature-branch
git rebase main
git checkout main
git merge feature-branch # Fast-forward merge
```

### Kiedy używać **rebase**:

- Gdy chcesz zachować czystą, linearną historię
- Przed mergowaniem feature branch do main
- Do czyszczenia lokalnej historii przed push

### Kiedy **NIE** używać **rebase**:

- Na commitach, które już zostały wypchnięte do zdalnego repozytorium
- Na gałęziach współdzielonych z innymi deweloperami
- Gdy historia merge jest ważna dla kontekstu

## Git Stash

Stash pozwala na tymczasowe zapisanie zmian bez tworzenia commita. Jest to przydatne, gdy musisz szybko przełączyć się na inną gałąź, ale nie chcesz commitować niedokończonej pracy.

Bash

```
# Zapisz zmiany w stash
git stash

# Zapisz z opisem
git stash save "Praca nad funkcją X"

# Lista stash'y
git stash list

# Zastosuj ostatni stash
git stash apply

# Zastosuj konkretny stash
git stash apply stash@{2}

# Zastosuj i usuń stash
git stash pop

# Usuń stash
```

```
git stash drop stash@{1}

# Usuń wszystkie stash'e
git stash clear

# Pokaż zawartość stash
git stash show -p stash@{0}
```

## Zaawansowane operacje stash

Bash

```
# Stash tylko śledzonych plików
git stash --keep-index

# Stash z nieśledzonymi plikami
git stash -u

# Stash wszystkich plików (włącznie z ignorowanymi)
git stash -a

# Interaktywny stash
git stash -p

# Tworzenie gałęzi ze stash
git stash branch nazwa-galezi stash@{1}
```

## Git Cherry-pick

Cherry-pick pozwala na zastosowanie konkretnego commita z jednej gałęzi do innej, bez mergowania całej gałęzi.

Bash

```
# Zastosuj konkretny commit
git cherry-pick commit-hash

# Zastosuj zakres commitów
git cherry-pick commit1..commit3

# Cherry-pick bez automatycznego commita
git cherry-pick -n commit-hash
```

```
# Cherry-pick z zachowaniem oryginalnego autora
git cherry-pick -x commit-hash

# Rozwiązywanie konfliktów podczas cherry-pick
git cherry-pick commit-hash
# Rozwiąż konflikty
git add .
git cherry-pick --continue

# Anulowanie cherry-pick
git cherry-pick --abort
```

## Git Bisect

Bisect to narzędzie do znajdowania commita, który wprowadził błąd, używając algorytmu wyszukiwania binarnego.

Bash

```
# Rozpocznij bisect
git bisect start

# Oznacz aktualny commit jako zły
git bisect bad

# Oznacz dobry commit (np. ostatni działający)
git bisect good commit-hash

# Git automatycznie przełączy na commit w środku
# Przetestuj aplikację i oznacz jako good lub bad
git bisect good # lub git bisect bad

# Kontynuuj proces aż do znalezienia problematycznego commita

# Zakończ bisect
git bisect reset
```

## Automatyczny bisect

Bash

```
# Automatyczny bisect z skrypcem testowym
git bisect start HEAD v1.0
```



```
git bisect run ./test-script.sh

# Skrypt test-script.sh powinien zwracać:
# 0 - test przeszedł (good)
# 1-124, 126-127 - test nie przeszedł (bad)
# 125 - nie można przetestować (skip)
```

## Git Hooks

Hooks to skrypty, które Git automatycznie uruchamia w określonych momentach. Znajdują się w katalogu `.git/hooks/`.

### Rodzaje hooks

#### Client-side hooks:

- `pre-commit` - uruchamiany przed commitem
- `prepare-commit-msg` - uruchamiany przed edytorem wiadomości commita
- `commit-msg` - uruchamiany po wprowadzeniu wiadomości commita
- `post-commit` - uruchamiany po commicie
- `pre-push` - uruchamiany przed push
- `pre-rebase` - uruchamiany przed rebase

#### Server-side hooks:

- `pre-receive` - uruchamiany przed przyjęciem push
- `update` - uruchamiany dla każdej gałęzi podczas push
- `post-receive` - uruchamiany po przyjęciu push

### Przykład pre-commit hook

```
Bash
```

```
#!/bin/sh
# .git/hooks/pre-commit

# Sprawdź składnię Python
python -m py_compile *.py
if [ $? -ne 0 ]; then
    echo "Błąd składni Python. Commit anulowany."
    exit 1
fi

# Uruchom testy
python -m pytest tests/
if [ $? -ne 0 ]; then
    echo "Testy nie przeszły. Commit anulowany."
    exit 1
fi

echo "Wszystkie sprawdzenia przeszły pomyślnie."
exit 0
```

Bash

```
# Uczyń hook wykonywalnym
chmod +x .git/hooks/pre-commit
```

## Git Submodules

Submoduły pozwalają na włączenie jednego repozytorium Git jako podkatalog innego repozytorium.

Bash

```
# Dodaj submoduł
git submodule add https://github.com/user/repo.git path/to/submodule

# Inicjalizuj submoduły po klonowaniu
git submodule init
git submodule update

# Lub w jednym poleceniu
git submodule update --init --recursive
```

```
# Klonowanie z submodulejami
git clone --recursive https://github.com/user/main-repo.git

# Aktualizacja submodulew
git submodule update --remote

# Usunięcie submodule
git submodule deinit path/to/submodule
git rm path/to/submodule
rm -rf .git/modules/path/to/submodule
```

## Git Worktree

Worktree pozwala na jednoczesną pracę z wieloma gałęziami w różnych katalogach.

Bash

```
# Lista worktree
git worktree list

# Dodaj nowy worktree
git worktree add ../feature-branch feature-branch

# Dodaj worktree z nową gałęzią
git worktree add -b new-feature ../new-feature

# Usuń worktree
git worktree remove ../feature-branch

# Wyczyść nieistniejące worktree
git worktree prune
```

## Zaawansowane konfiguracje Git

### Aliasy Git

Bash

```
# Podstawowe aliasy
git config --global alias.co checkout
git config --global alias.br branch
```

```
git config --global alias.ci commit
git config --global alias.st status

# Zaawansowane aliasy
git config --global alias.unstage 'reset HEAD --'
git config --global alias.last 'log -1 HEAD'
git config --global alias.visual '!gitk'

# Alias dla ładnego loga
git config --global alias.lg "log --color --graph --
pretty=format: '%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold
blue)<%an>%Creset' --abbrev-commit"

# Alias dla statusu z krótkim formatem
git config --global alias.s 'status -s'
```

## Konfiguracja edytora i narzędzi diff/merge

Bash

```
# Ustawienie VS Code jako edytor
git config --global core.editor "code --wait"

# Ustawienie narzędzia diff
git config --global diff.tool vimdiff

# Ustawienie narzędzia merge
git config --global merge.tool vimdiff

# Konfiguracja dla VS Code jako narzędzie merge
git config --global merge.tool vscode
git config --global mergetool.vscode.cmd 'code --wait $MERGED'

# Automatyczne usuwanie plików .orig po merge
git config --global mergetool.keepBackup false
```

## Konfiguracja push i pull

Bash

```
# Ustawienie domyślnego zachowania push
git config --global push.default simple

# Automatyczne ustawienie upstream dla nowych gałęzi
```

```
git config --global push.autoSetupRemote true

# Ustawienie domyślnego zachowania pull
git config --global pull.rebase true

# Automatyczne przycinanie zdalnych gałęzi podczas fetch
git config --global fetch.prune true
```

## Git Flow

Git Flow to model rozgałęziania, który definiuje strukturę gałęzi dla projektów z regularnymi wydaniem.

### Instalacja Git Flow

Bash

```
# Ubuntu/Debian
sudo apt install git-flow

# macOS
brew install git-flow

# Inicjalizacja w projekcie
git flow init
```

### Główne gałęzie Git Flow

- **main/master** - stabilna wersja produkcyjna
- **develop** - główna gałąź rozwoju
- **feature/** - nowe funkcje
- **release/** - przygotowanie wydań
- **hotfix/** - szybkie poprawki produkcyjne

### Praca z feature branches

Bash

```
# Rozpocznij nową funkcję
git flow feature start nazwa-funkcji

# Zakończ funkcję
git flow feature finish nazwa-funkcji

# Opublikuj funkcję
git flow feature publish nazwa-funkcji

# Pobierz funkcję
git flow feature pull origin nazwa-funkcji
```

## Praca z release branches

Bash

```
# Rozpocznij release
git flow release start 1.0.0

# Zakończ release
git flow release finish 1.0.0
```

## Praca z hotfix branches

Bash

```
# Rozpocznij hotfix
git flow hotfix start nazwa-hotfix

# Zakończ hotfix
git flow hotfix finish nazwa-hotfix
```

## Rozwiązywanie problemów

### Odzyskiwanie usuniętych commitów

Bash

```
# Pokaż historię wszystkich operacji
git reflog

# Odzyskaj commit
git checkout commit-hash
git checkout -b recovered-branch

# Lub użyj reset
git reset --hard commit-hash
```

## Naprawianie błędnych commitów

Bash

```
# Zmień wiadomość ostatniego commita
git commit --amend -m "Nowa wiadomość"

# Dodaj pliki do ostatniego commita
git add forgotten-file.txt
git commit --amend --no-edit

# Podziel ostatni commit
git reset --soft HEAD~1
git add file1.txt
git commit -m "Pierwszy commit"
git add file2.txt
git commit -m "Drugi commit"
```

## Czyszczenie repozytorium

Bash

```
# Usuń nieśledzone pliki
git clean -f

# Usuń nieśledzone pliki i katalogi
git clean -fd

# Pokaż co zostanie usunięte (dry run)
git clean -n

# Usuń pliki ignorowane przez .gitignore
git clean -fx
```

```
# Usuń wszystko nieśledzone
git clean -fx
```

## Optymalizacja repozytorium

Bash

```
# Kompresja i optymalizacja
git gc

# Agresywna optymalizacja
git gc --aggressive

# Sprawdzenie integralności
git fsck

# Statystyki repozytorium
git count-objects -v
```

## Najlepsze praktyki zaawansowane

### Konwencje nazewnictwa gałęzi

Plain Text

```
feature/JIRA-123-user-authentication
bugfix/JIRA-456-login-error
hotfix/JIRA-789-security-patch
release/v1.2.0
```

### Konwencje wiadomości commitów

Używaj formatu Conventional Commits:

Plain Text

```
type(scope): description

[optional body]
```



[optional footer]

Przykłady:

Plain Text

```
feat(auth): add user authentication
fix(api): resolve null pointer exception
docs(readme): update installation instructions
style(css): fix button alignment
refactor(utils): extract common functions
test(auth): add unit tests for login
chore(deps): update dependencies
```

## Strategia mergowania

Bash

```
# Dla feature branches - squash merge
git checkout main
git merge --squash feature-branch
git commit -m "feat: add new feature"

# Dla release branches - merge commit
git checkout main
git merge --no-ff release/v1.0.0

# Dla hotfixes - fast-forward merge
git checkout main
git merge hotfix/critical-bug
```

## Automatyzacja z Git hooks

Przykład kompleksowego pre-commit hook:

Bash

```
#!/bin/sh
# .git/hooks/pre-commit

set -e
```

```
echo "Uruchamianie pre-commit checks..."

# Sprawdź formatowanie kodu
if command -v black &> /dev/null; then
    echo "Sprawdzanie formatowania Python..."
    black --check .
fi

# Sprawdź linting
if command -v flake8 &> /dev/null; then
    echo "Sprawdzanie linting Python..."
    flake8 .
fi

# Sprawdź testy
if [ -f "pytest.ini" ] || [ -f "setup.cfg" ] || [ -f "pyproject.toml" ]; then
    echo "Uruchamianie testów..."
    python -m pytest tests/ -x
fi

# Sprawdź bezpieczeństwo
if command -v bandit &> /dev/null; then
    echo "Sprawdzanie bezpieczeństwa..."
    bandit -r . -f json -o bandit-report.json
fi

echo "Wszystkie sprawdzenia przeszły pomyślnie!"
```

To kończy zaawansowane funkcje Git. W następnym rozdziale omówimy najlepsze praktyki Git w kontekście DevOps.

## 4. Najlepsze praktyki Git w DevOps

### Strategia rozgałęziania w DevOps

Wybór odpowiedniej strategii rozgałęziania jest kluczowy dla sukcesu implementacji DevOps. Różne strategie sprawdzają się w różnych kontekstach organizacyjnych i projektowych.

#### GitHub Flow

GitHub Flow to prosta strategia idealna dla projektów z ciągłym wdrażaniem:

Bash

```
# 1. Utwórz gałąź feature z main
git checkout main
git pull origin main
git checkout -b feature/new-functionality

# 2. Pracuj nad funkcją, commituj często
git add .
git commit -m "feat: implement user registration"
git push -u origin feature/new-functionality

# 3. Otwórz Pull Request
# 4. Po review i testach, zmerguj do main
# 5. Wdróż main na produkcję
# 6. Usuń gałąź feature
git branch -d feature/new-functionality
git push origin --delete feature/new-functionality
```

### Zalety GitHub Flow:

- Prostota i łatwość zrozumienia
- Szybkie feedback i wdrożenia
- Idealne dla małych zespołów
- Wspiera continuous deployment

### Wady GitHub Flow:

- Może być problematyczne dla dużych zespołów
- Brak wsparcia dla multiple environments
- Trudności z hotfixami

## GitLab Flow

GitLab Flow łączy prostotę GitHub Flow z elastycznością Git Flow:

Bash

```
# Environment branches
main -> pre-production -> production

# Feature development
git checkout -b feature/user-dashboard main
# Praca nad funkcją
git push origin feature/user-dashboard
# Merge do main po review

# Release process
git checkout -b release/v1.2.0 main
# Testy i stabilizacja
git checkout pre-production
git merge release/v1.2.0
# Po testach na pre-production
git checkout production
git merge pre-production
```

## Trunk-based Development

Strategia promowana przez Google i inne organizacje high-performance:

Bash

```
# Wszyscy pracują na main/trunk
git checkout main
git pull origin main

# Krótkotrwałe gałęzie (max 1-2 dni)
git checkout -b short-lived-feature
# Szybka implementacja
git push origin short-lived-feature
# Natychmiastowy merge po review

# Feature flags dla większych funkcji
if feature_flag_enabled('new_dashboard'):
    render_new_dashboard()
else:
    render_old_dashboard()
```

### Kluczowe zasady Trunk-based Development:

- Gałęzie feature żyją maksymalnie 1-2 dni

- Częste commity do main (kilka razy dziennie)
- Używanie feature flags dla większych funkcji
- Bardzo szybkie code review (< 24h)
- Automatyczne testy na każdy commit

## Continuous Integration z Git

### Konfiguracja CI pipeline

Przykład konfiguracji GitHub Actions:

YAML

```
# .github/workflows/ci.yml
name: CI Pipeline

on:
  push:
    branches: [ main, develop ]
  pull_request:
    branches: [ main ]

jobs:
  test:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        python-version: [3.8, 3.9, 3.10]

    steps:
      - uses: actions/checkout@v3

      - name: Set up Python ${ matrix.python-version }
        uses: actions/setup-python@v3
        with:
          python-version: ${ matrix.python-version }

      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install -r requirements.txt
```

```

    pip install -r requirements-dev.txt

- name: Lint with flake8
  run: |
    flake8 . --count --select=E9,F63,F7,F82 --show-source --statistics
    flake8 . --count --exit-zero --max-complexity=10 --max-line-
length=127 --statistics

- name: Test with pytest
  run: |
    pytest tests/ --cov=src --cov-report=xml

- name: Upload coverage to Codecov
  uses: codecov/codecov-action@v3
  with:
    file: ./coverage.xml

security:
  runs-on: ubuntu-latest
  steps:
    - uses: actions/checkout@v3

- name: Run security checks
  run: |
    pip install bandit safety
    bandit -r src/
    safety check

```

## Pre-commit hooks dla CI

Bash

```

# Instalacja pre-commit
pip install pre-commit

# Konfiguracja .pre-commit-config.yaml
cat > .pre-commit-config.yaml << EOF
repos:
  - repo: https://github.com/pre-commit/pre-commit-hooks
    rev: v4.4.0
    hooks:
      - id: trailing-whitespace
      - id: end-of-file-fixer
      - id: check-yaml
      - id: check-added-large-files
      - id: check-merge-conflict

```

```
- repo: https://github.com/psf/black
  rev: 22.10.0
  hooks:
    - id: black

- repo: https://github.com/pycqa/flake8
  rev: 5.0.4
  hooks:
    - id: flake8

- repo: https://github.com/pycqa/isort
  rev: 5.10.1
  hooks:
    - id: isort

- repo: https://github.com/pre-commit/mirrors-mypy
  rev: v0.991
  hooks:
    - id: mypy
EOF

# Instalacja hooks
pre-commit install

# Uruchomienie na wszystkich plikach
pre-commit run --all-files
```

## Semantic Versioning i Git Tags

### Automatyczne wersjonowanie

Bash

```
# Instalacja semantic-release
npm install -g semantic-release

# Konfiguracja .releaserc.json
cat > .releaserc.json << EOF
{
  "branches": ["main"],
  "plugins": [
    "@semantic-release/commit-analyzer",
    "@semantic-release/release-notes-generator",
    "@semantic-release/changelog",
```

```
"@semantic-release/npm",
"@semantic-release/git",
"@semantic-release/github"
]
}
EOF
```

## Konwencje commitów dla automatycznego wersjonowania

Bash

```
# PATCH version (1.0.0 -> 1.0.1)
git commit -m "fix: resolve memory leak in user service"

# MINOR version (1.0.0 -> 1.1.0)
git commit -m "feat: add user profile management"

# MAJOR version (1.0.0 -> 2.0.0)
git commit -m "feat!: redesign authentication system

BREAKING CHANGE: authentication API has been completely redesigned"
```

## Ręczne tagowanie wydań

Bash

```
# Tworzenie tagu z adnotacją
git tag -a v1.2.0 -m "Release version 1.2.0

Features:
- User authentication
- Dashboard improvements
- Performance optimizations

Bug fixes:
- Fixed login redirect issue
- Resolved memory leak in cache"

# Wysyłanie tagów
git push origin v1.2.0
git push origin --tags
```



```
# Tworzenie release notes
git log v1.1.0..v1.2.0 --pretty=format:"- %s" > RELEASE_NOTES.md
```

## Zarządzanie konfliktami w zespole

### Strategie rozwiązywania konfliktów

Bash

```
# 1. Rebase strategy (preferowana dla feature branches)
git checkout feature-branch
git rebase main

# Jeśli wystąpią konflikty:
# Edytuj pliki, usuń markery konfliktu
git add .
git rebase --continue

# 2. Merge strategy (dla długotrwałych gałęzi)
git checkout main
git merge feature-branch

# 3. Three-way merge tool
git mergetool

# 4. Ręczne rozwiązywanie
git status
# Edytuj pliki z konfliktami
git add resolved-file.py
git commit
```

### Zapobieganie konfliktom

Bash

```
# Częste synchronizowanie z main
git checkout feature-branch
git fetch origin
git rebase origin/main

# Małe, atomowe commity
git add specific-file.py
git commit -m "feat: add user validation function"
```

```
# Komunikacja w zespole o zmianach w wspólnych plikach
# Używanie CODEOWNERS file
cat > .github/CODEOWNERS << EOF
# Global owners
* @team-leads

# Frontend
/frontend/ @frontend-team

# Backend API
/api/ @backend-team

# Database migrations
/migrations/ @database-team @team-leads

# CI/CD configuration
/.github/ @devops-team
/.gitlab-ci.yml @devops-team
/Dockerfile @devops-team
EOF
```

## Bezpieczeństwo w Git

### Podpisywanie commitów GPG

Bash

```
# Generowanie klucza GPG
gpg --full-generate-key

# Lista kluczy
gpg --list-secret-keys --keyid-format LONG

# Konfiguracja Git
git config --global user.signingkey YOUR_KEY_ID
git config --global commit.gpgsign true

# Podpisywanie konkretnego commita
git commit -S -m "feat: add secure authentication"

# Eksport klucza publicznego (do GitHub/GitLab)
gpg --armor --export YOUR_KEY_ID
```

## Skanowanie sekretów

Bash

```
# Instalacja git-secrets
git clone https://github.com/awslabs/git-secrets.git
cd git-secrets
make install

# Konfiguracja w repozytorium
git secrets --install
git secrets --register-aws

# Dodanie własnych wzorców
git secrets --add 'password\s*=\s*.*+'
git secrets --add 'api[_-]?key\s*=\s*.*+'

# Skanowanie
git secrets --scan
git secrets --scan-history
```

## .gitignore dla bezpieczeństwa

Bash

```
# Pliki z sekretami
.env
.env.local
.env.production
secrets.yml
config/secrets.yml

# Klucze i certyfikaty
*.pem
*.key
*.crt
*.p12
*.pfx

# Pliki konfiguracyjne z danymi wrażliwymi
config/database.yml
config/production.rb
aws-credentials.json

# IDE i edytory
```

```
.vscode/settings.json
.idea/
*.swp
*.swO

# Logi mogące zawierać dane wrażliwe
*.log
logs/
npm-debug.log*

# Pliki backup
*.bak
*.backup
*.old
```

## Monitoring i metryki Git

### Git analytics

Bash

```
# Statystyki commitów
git shortlog -sn --all
git log --pretty=format:@"%h %an %ad %s" --date=short --since="1 month ago"

# Analiza aktywności
git log --author="John Doe" --since="1 week ago" --oneline
git log --grep="fix" --oneline --since="1 month ago"

# Statystyki plików
git log --stat --since="1 month ago"
git whatchanged --since="1 week ago" --oneline
```

### Automatyczne raporty

Python

```
#!/usr/bin/env python3
# git-report.py

import subprocess
import json
from datetime import datetime, timedelta
```

```

def get_git_stats(since_days=30):
    since_date = (datetime.now() - timedelta(days=since_days)).strftime('%Y-%m-%d')

    # Commits per author
    cmd = f'git shortlog -sn --since="{since_date}"'
    result = subprocess.run(cmd, shell=True, capture_output=True, text=True)

    commits_per_author = {}
    for line in result.stdout.strip().split('\n'):
        if line:
            count, author = line.strip().split('\t', 1)
            commits_per_author[author] = int(count)

    # Files changed
    cmd = f'git log --name-only --pretty=format: --since="{since_date}" | sort | uniq -c | sort -nr'
    result = subprocess.run(cmd, shell=True, capture_output=True, text=True)

    files_changed = {}
    for line in result.stdout.strip().split('\n')[:10]: # Top 10
        if line.strip():
            count, filename = line.strip().split(None, 1)
            files_changed[filename] = int(count)

    return {
        'period': f'Last {since_days} days',
        'commits_per_author': commits_per_author,
        'most_changed_files': files_changed
    }

if __name__ == '__main__':
    stats = get_git_stats()
    print(json.dumps(stats, indent=2))

```

## Integracja z narzędziami DevOps

### Git hooks dla automatyzacji

Bash

```

# Post-receive hook dla automatycznego deployment
#!/bin/bash
# .git/hooks/post-receive

```

```

while read oldrev newrev refname; do
    if [[ $refname == "refs/heads/main" ]]; then
        echo "Deploying to production..."

        # Checkout to working directory
        git --git-dir=/path/to/repo.git --work-tree=/path/to/production
        checkout -f main

        # Restart services
        sudo systemctl restart myapp

        # Notify team
        curl -X POST -H 'Content-type: application/json' \
            --data '{"text": "New deployment to production completed"}' \
            $SLACK_WEBHOOK_URL
    fi
done

```

## Integracja z Jenkins

Plain Text

```

// Jenkinsfile
pipeline {
    agent any

    environment {
        GIT_COMMIT_SHORT = sh(
            script: "git rev-parse --short HEAD",
            returnStdout: true
        ).trim()
    }

    stages {
        stage('Checkout') {
            steps {
                checkout scm
                script {
                    env.GIT_COMMIT_MSG = sh(
                        script: "git log -1 --pretty=format:'%s'",
                        returnStdout: true
                    ).trim()
                }
            }
        }
    }
}

```

```

stage('Test') {
    steps {
        sh 'python -m pytest tests/'
    }
    post {
        always {
            publishTestResults testResultsPattern: 'test-results.xml'
        }
    }
}

stage('Build') {
    when {
        branch 'main'
    }
    steps {
        sh 'docker build -t myapp:${GIT_COMMIT_SHORT} .'
        sh 'docker tag myapp:${GIT_COMMIT_SHORT} myapp:latest'
    }
}

stage('Deploy') {
    when {
        branch 'main'
    }
    steps {
        sh 'docker push myapp:${GIT_COMMIT_SHORT}'
        sh 'kubectl set image deployment/myapp
myapp=myapp:${GIT_COMMIT_SHORT}'
    }
}

post {
    success {
        slackSend(
            color: 'good',
            message: "✅ Deployment successful: ${env.GIT_COMMIT_MSG}
(${env.GIT_COMMIT_SHORT})"
        )
    }
    failure {
        slackSend(
            color: 'danger',
            message: "❌ Deployment failed: ${env.GIT_COMMIT_MSG}
(${env.GIT_COMMIT_SHORT})"
        )
    }
}

```

```
}  
}  
}
```

## Backup i disaster recovery

### Backup strategia

Bash

```
# Backup wszystkich gałęzi i tagów  
git clone --mirror https://github.com/user/repo.git repo-backup.git  
  
# Aktualizacja backup  
cd repo-backup.git  
git remote update  
  
# Backup do różnych lokalizacji  
git push --mirror backup-remote-1  
git push --mirror backup-remote-2  
  
# Automatyczny backup script  
#!/bin/bash  
# backup-git-repos.sh  
  
BACKUP_DIR="/backup/git-repos"  
REPOS=(  
    "https://github.com/company/app1.git"  
    "https://github.com/company/app2.git"  
    "https://github.com/company/infrastructure.git"  
)  
  
for repo in "${REPOS[@]"; do  
    repo_name=$(basename "$repo" .git)  
    backup_path="$BACKUP_DIR/$repo_name.git"  
  
    if [ -d "$backup_path" ]; then  
        echo "Updating backup for $repo_name..."  
        cd "$backup_path"  
        git remote update  
    else  
        echo "Creating backup for $repo_name..."  
        git clone --mirror "$repo" "$backup_path"  
    fi  
done
```



```
# Kompresja i archiwizacja
tar -czf "$BACKUP_DIR/git-backup-$(date +%Y%m%d).tar.gz" -C "$BACKUP_DIR" .
```

## Recovery procedures

Bash

```
# Odzyskiwanie z mirror backup
git clone backup-repo.git recovered-repo
cd recovered-repo
git remote set-url origin https://github.com/user/repo.git

# Odzyskiwanie usuniętej gałęzi
git reflog --all | grep branch-name
git checkout -b recovered-branch commit-hash

# Odzyskiwanie z bundle
git bundle create repo-backup.bundle --all
# Na innym systemie:
git clone repo-backup.bundle recovered-repo
```

To kończy sekcję o najlepszych praktykach Git w DevOps. W następnym rozdziale przejdziemy do Jenkins i Continuous Integration/Continuous Deployment.

## 5. Wprowadzenie do CI/CD

### Czym jest CI/CD?

Continuous Integration/Continuous Deployment (CI/CD) to zestaw praktyk i narzędzi, które automatyzują proces integracji kodu, testowania i wdrażania aplikacji. CI/CD stanowi fundament nowoczesnych praktyk DevOps i umożliwia zespołom dostarczanie oprogramowania szybciej, bezpieczniej i z wyższą jakością.

Termin CI/CD faktycznie obejmuje trzy powiązane praktyki: Continuous Integration (CI), Continuous Delivery (CD) i Continuous Deployment (również CD). Chociaż często używane zamiennie, każda z tych praktyk ma swoje specyficzne cele i implementacje.

### Continuous Integration (CI)

Continuous Integration to praktyka, w której deweloperzy regularnie (często kilka razy dziennie) integrują swoje zmiany kodu z główną gałęzią repozytorium. Każda integracja jest automatycznie weryfikowana przez zautomatyzowane buildy i testy, co pozwala na szybkie wykrycie problemów integracyjnych.

Kluczowe elementy CI obejmują:

**Częste commity** - deweloperzy commitują kod do głównej gałęzi co najmniej raz dziennie, a preferowane jest jeszcze częstsze commitowanie. To minimalizuje konflikty merge i ułatwia identyfikację źródła problemów.

**Automatyczne buildy** - każdy commit automatycznie uruchamia proces budowania aplikacji. Build powinien być szybki (idealnie poniżej 10 minut) i obejmować kompilację, linkowanie i pakowanie aplikacji.

**Automatyczne testowanie** - każdy build automatycznie uruchamia zestaw testów, które weryfikują poprawność kodu. Testy powinny być szybkie, niezawodne i zapewniać odpowiednie pokrycie kodu.

**Szybkie feedback** - zespół otrzymuje natychmiastowe informacje o statusie buildu i testów. Jeśli build się nie powiedzie, jest to traktowane jako najwyższy priorytet do naprawienia.

## Continuous Delivery (CD)

Continuous Delivery rozszerza CI o automatyzację procesu przygotowania kodu do wydania. W Continuous Delivery kod jest zawsze w stanie gotowym do wdrożenia na produkcję, ale faktyczne wdrożenie wymaga manualnego zatwierdzenia.

Kluczowe aspekty Continuous Delivery:

**Deployment pipeline** - zautomatyzowany proces, który przenosi kod przez różne środowiska (development, testing, staging) aż do momentu, gdy jest gotowy na produkcję.

**Automatyzacja środowisk** - wszystkie środowiska są tworzone i konfigurowane automatycznie, zapewniając spójność między środowiskami.

**Automatyzacja testów** - oprócz testów jednostkowych, pipeline obejmuje testy integracyjne, testy wydajności, testy bezpieczeństwa i testy akceptacyjne.

**Artefakty wersjonowane** - każdy build tworzy wersjonowane artefakty, które mogą być wdrożone w dowolnym momencie.

## Continuous Deployment

Continuous Deployment to najwyższy poziom automatyzacji, gdzie każda zmiana, która przejdzie przez pipeline, jest automatycznie wdrażana na produkcję bez manualnej interwencji.

Charakterystyki Continuous Deployment:

**Pełna automatyzacja** - nie ma manualnych kroków zatwierdzania; jeśli kod przejdzie przez wszystkie testy, jest automatycznie wdrażany.

**Wysokie zaufanie do testów** - organizacja musi mieć bardzo wysokie zaufanie do swojego zestawu testów automatycznych.

**Monitoring i rollback** - silne systemy monitoringu i możliwość szybkiego rollback są kluczowe.

**Kultura DevOps** - wymaga dojrzałej kultury DevOps z wysokim poziomem współpracy między zespołami.

## Korzyści z implementacji CI/CD

### Szybsze dostarczanie wartości biznesowej

CI/CD dramatycznie skraca czas od pomysłu do wdrożenia na produkcję. Tradycyjne cykle wydawnicze, które mogły trwać miesiące, zostają skrócone do dni lub nawet godzin. To pozwala organizacjom szybciej reagować na zmiany rynkowe, feedback użytkowników i nowe możliwości biznesowe.

Według raportu "State of DevOps" z 2021 roku, organizacje o wysokiej wydajności wdrażają kod 973 razy częściej niż organizacje o niskiej wydajności. Lead time dla zmian w organizacjach high-performance wynosi mniej niż godzinę, podczas gdy w organizacjach low-performance może wynosić od jednego do sześciu miesięcy.

## Wyzsza jakość oprogramowania

Paradoksalnie, mimo szybszego tempa dostarczania, CI/CD prowadzi do wyższej jakości oprogramowania. Dzieje się tak z kilku powodów:

**Wczesne wykrywanie błędów** - automatyczne testy uruchamiane przy każdym commicie pozwalają na wykrycie problemów w momencie ich wprowadzenia, gdy kontekst jest jeszcze świeży w umyśle dewelopera.

**Małe, inkrementalne zmiany** - zamiast dużych, monolitycznych wydań, CI/CD promuje małe, częste zmiany, które są łatwiejsze do testowania, debugowania i rollback.

**Spójność środowisk** - automatyzacja deployment zapewnia, że wszystkie środowiska są identyczne, eliminując problemy typu "działa na moim komputerze".

**Continuous feedback** - szybkie feedback z różnych środowisk i od użytkowników pozwala na natychmiastowe reagowanie na problemy.

## Redukcja ryzyka

CI/CD znacząco redukuje ryzyko związane z wdrożeniami:

**Mniejsze zmiany** - małe, częste zmiany niosą mniejsze ryzyko niż duże, rzadkie wydania.

**Szybki rollback** - możliwość szybkiego powrotu do poprzedniej wersji minimalizuje wpływ problemów na użytkowników.

**Testowanie w produkcji** - techniki takie jak blue-green deployment, canary releases i feature flags pozwalają na bezpieczne testowanie w środowisku produkcyjnym.

**Automatyzacja** - eliminacja manualnych kroków redukuje ryzyko błędów ludzkich.

## Lepsza współpraca zespołowa

CI/CD promuje lepszą współpracę między różnymi zespołami:

**Wspólna odpowiedzialność** - wszyscy członkowie zespołu są odpowiedzialni za jakość i stabilność pipeline'u.

**Transparentność** - status buildów i deploymentów jest widoczny dla wszystkich, co promuje otwartą komunikację o problemach.

**Shared ownership** - kod, testy i infrastruktura są wspólną własnością zespołu, a nie indywidualnych deweloperów.

## Komponenty pipeline'u CI/CD

### Source Control

Wszystko zaczyna się od systemu kontroli wersji, najczęściej Git. Source control nie obejmuje tylko kodu aplikacji, ale także:

**Infrastructure as Code** - definicje infrastruktury w plikach Terraform, CloudFormation lub Ansible.

**Configuration as Code** - konfiguracje aplikacji, baz danych i innych komponentów.

**Pipeline as Code** - definicje samych pipeline'ów CI/CD w plikach takich jak Jenkinsfile, .gitlab-ci.yml czy GitHub Actions workflows.

**Documentation as Code** - dokumentacja przechowywana razem z kodem, często w formacie Markdown.

### Build Stage

Etap budowania kompiluje kod źródłowy w artefakty gotowe do wdrożenia:

**Kompilacja** - dla języków kompilowanych (Java, C#, Go) lub transpilacja dla języków takich jak TypeScript.

**Dependency management** - pobieranie i zarządzanie zależnościami zewnętrznymi.

**Asset processing** - optymalizacja obrazów, minifikacja CSS/JavaScript, bundling.

**Packaging** - tworzenie artefaktów deployment (JAR, WAR, Docker images, ZIP archives).

### Test Stage

Etap testowania weryfikuje jakość i poprawność kodu:

**Unit Tests** - testy jednostkowe weryfikujące pojedyncze komponenty w izolacji.

**Integration Tests** - testy sprawdzające interakcje między komponentami.

**Contract Tests** - testy weryfikujące kontrakty API między serwisami.

**End-to-End Tests** - testy sprawdzające kompletne scenariusze użytkownika.

**Performance Tests** - testy wydajności i obciążenia.

**Security Tests** - skanowanie podatności i testy bezpieczeństwa.

## Quality Gates

Quality Gates to automatyczne kontrole jakości, które muszą być spełnione, aby kod mógł przejść dalej w pipeline:

**Code Coverage** - minimalny procent pokrycia kodu testami (np. 80%).

**Code Quality** - metryki jakości kodu takie jak cykliczność złożoność, duplikacja kodu.

**Security Scan Results** - brak krytycznych podatności bezpieczeństwa.

**Performance Benchmarks** - aplikacja musi spełniać określone kryteria wydajności.

## Deployment Stages

Pipeline może obejmować wdrożenia do różnych środowisk:

**Development Environment** - środowisko dla deweloperów do testowania integracji.

**Testing Environment** - środowisko dla zespołów QA do przeprowadzania testów manualnych.

**Staging Environment** - środowisko identyczne z produkcją do finalnych testów.

**Production Environment** - środowisko produkcyjne dla użytkowników końcowych.

## Strategie deployment

## Blue-Green Deployment

Blue-Green Deployment to strategia, w której utrzymywane są dwa identyczne środowiska produkcyjne:

Bash

```
# Przykład Blue-Green deployment z Docker
# Aktualne środowisko (Blue)
docker run -d --name app-blue -p 8080:8080 myapp:v1.0

# Nowe środowisko (Green)
docker run -d --name app-green -p 8081:8080 myapp:v2.0

# Testowanie Green
curl http://localhost:8081/health

# Przełączenie ruchu (load balancer configuration)
# Blue -> Green
# Po weryfikacji, usunięcie Blue
docker stop app-blue
docker rm app-blue
```

### Zalety:

- Natychmiastowy rollback
- Zero downtime deployment
- Pełne testowanie przed przełączeniem

### Wady:

- Podwojone koszty infrastruktury
- Kompleksowość zarządzania danymi
- Wymagania dotyczące load balancera

## Canary Deployment

Canary Deployment stopniowo kieruje ruch do nowej wersji:

## YAML

```
# Kubernetes Canary Deployment
apiVersion: argoproj.io/v1alpha1
kind: Rollout
metadata:
  name: myapp-rollout
spec:
  replicas: 10
  strategy:
    canary:
      steps:
        - setWeight: 10    # 10% ruchu do nowej wersji
        - pause: {duration: 30s}
        - setWeight: 30    # 30% ruchu
        - pause: {duration: 30s}
        - setWeight: 50    # 50% ruchu
        - pause: {duration: 30s}
        - setWeight: 100   # 100% ruchu
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
        - name: myapp
          image: myapp:v2.0
```

## Zalety:

- Minimalne ryzyko
- Stopniowe wykrywanie problemów
- Możliwość szybkiego rollback

## Wady:

- Dłuższy czas deployment
- Kompleksowość monitoringu
- Wymagania dotyczące traffic splitting



# Rolling Deployment

Rolling Deployment stopniowo zastępuje stare instancje nowymi:

YAML

```
# Kubernetes Rolling Update
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-deployment
spec:
  replicas: 6
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
        - name: myapp
          image: myapp:v2.0
```

## Zalety:

- Nie wymaga dodatkowej infrastruktury
- Stopniowe wdrożenie
- Wbudowane w Kubernetes

## Wady:

- Możliwe problemy z kompatybilnością wersji
- Dłuższy czas rollback
- Potencjalne problemy z sesją użytkownika

# Metryki CI/CD

## DORA Metrics

DevOps Research and Assessment (DORA) zidentyfikowało cztery kluczowe metryki:

**Deployment Frequency** - jak często organizacja wdraża kod na produkcję.

- Elite: Wiele razy dziennie
- High: Raz dziennie do raz w tygodniu
- Medium: Raz w tygodniu do raz w miesiącu
- Low: Raz w miesiącu do raz na sześć miesięcy

**Lead Time for Changes** - czas od commita do wdrożenia na produkcję.

- Elite: Mniej niż godzina
- High: Jeden dzień do jeden tydzień
- Medium: Jeden tydzień do jeden miesiąc
- Low: Jeden miesiąc do sześć miesięcy

**Change Failure Rate** - procent deploymentów powodujących awarie.

- Elite: 0-15%
- High: 16-30%
- Medium: 16-30%
- Low: 16-30%

**Time to Restore Service** - czas potrzebny na przywrócenie usługi po awarii.

- Elite: Mniej niż godzina
- High: Jeden dzień
- Medium: Jeden dzień do jeden tydzień

- Low: Jeden tydzień do jeden miesiąc

## Pipeline Metrics

**Build Success Rate** - procent buildów, które kończą się sukcesem.

Bash

```
# Przykład kalkulacji
successful_builds = 95
total_builds = 100
success_rate = (successful_builds / total_builds) * 100 # 95%
```

**Build Duration** - średni czas trwania buildu.

Bash

```
# Cel: build poniżej 10 minut
# Monitoring: średnia z ostatnich 30 buildów
```

**Test Coverage** - procent kodu pokrytego testami.

Bash

```
# Przykład z pytest-cov
pytest --cov=src tests/ --cov-report=term-missing
```

**Deployment Success Rate** - procent udanych deploymentów.

**Mean Time Between Deployments** - średni czas między deploymentami.

## Quality Metrics

**Code Quality Score** - metryki z narzędzi takich jak SonarQube.

**Security Vulnerabilities** - liczba wykrytych podatności bezpieczeństwa.

**Technical Debt** - szacowany czas potrzebny na naprawę problemów jakości kodu.

**Defect Escape Rate** - procent błędów wykrytych na produkcji zamiast w testach.

# Wyzwania implementacji CI/CD

## Kultura organizacyjna

Największym wyzwaniem jest często zmiana kultury organizacyjnej. CI/CD wymaga:

**Zaufania** - zespoły muszą ufać automatyzacji i być gotowe na częste zmiany.

**Współpracy** - przełamanie silosów między deweloperami, testerami i operatorami.

**Continuous Learning** - gotowość do uczenia się z błędów i ciągłego doskonalenia.

**Risk Tolerance** - akceptacja, że częste zmiany mogą czasami powodować problemy.

## Kompleksowość techniczna

**Legacy Systems** - integracja starszych systemów z nowoczesnymi praktykami CI/CD.

**Microservices Complexity** - zarządzanie pipeline'ami dla dziesiątek lub setek serwisów.

**Data Management** - synchronizacja danych między środowiskami.

**Security Integration** - włączenie bezpieczeństwa w każdy etap pipeline'u.

## Testowanie

**Test Automation** - tworzenie i utrzymanie kompleksowego zestawu testów automatycznych.

**Test Data Management** - zarządzanie danymi testowymi w różnych środowiskach.

**Performance Testing** - integracja testów wydajności w pipeline.

**Flaky Tests** - radzenie sobie z niestabilnymi testami, które czasami przechodzą, a czasami nie.

## Monitoring i observability

**Distributed Tracing** - śledzenie requestów przez wiele serwisów.

**Metrics Collection** - zbieranie i analiza metryk z różnych komponentów.

**Log Aggregation** - centralizacja i analiza logów z wielu źródeł.

**Alerting** - konfiguracja inteligentnych alertów, które minimalizują false positives.

To wprowadzenie do CI/CD stanowi fundament dla zrozumienia Jenkins i innych narzędzi, które omówimy w kolejnych rozdziałach. Praktyki CI/CD są kluczowe dla sukcesu każdej organizacji dążącej do implementacji DevOps.

## 6. Jenkins - Instalacja i konfiguracja

### Wprowadzenie do Jenkins

Jenkins to open-source'owy serwer automatyzacji napisany w Javie, który umożliwia deweloperom budowanie, testowanie i wdrażanie aplikacji. Stworzony przez Kohsuke Kawaguchi w 2004 roku jako Hudson, Jenkins stał się jednym z najpopularniejszych narzędzi CI/CD na świecie.

Jenkins wyróżnia się swoją elastycznością i rozszerzalnością. Dzięki architekturze opartej na pluginach, Jenkins może być dostosowany do praktycznie każdego workflow'u DevOps. Obecnie dostępnych jest ponad 1800 pluginów, które rozszerzają funkcjonalność Jenkins o integracje z różnymi narzędziami, platformami chmurowymi i systemami.

Kluczowe cechy Jenkins:

**Rozszerzalność** - bogaty ekosystem pluginów pozwala na integrację z praktycznie każdym narzędziem w łańcuchu DevOps.

**Distributed builds** - możliwość uruchamiania buildów na wielu maszynach jednocześnie, co zwiększa wydajność i skalowalność.

**Pipeline as Code** - definicja pipeline'ów w kodzie (Jenkinsfile), co umożliwia wersjonowanie i współdzielenie konfiguracji.

**Web-based interface** - intuicyjny interfejs webowy do zarządzania jobami, monitorowania buildów i konfiguracji systemu.

**Community support** - aktywna społeczność i bogata dokumentacja.

# Wymagania systemowe

Przed instalacją Jenkins należy upewnić się, że system spełnia minimalne wymagania:

## Wymagania sprzętowe

### Minimalne wymagania:

- RAM: 256 MB (zalecane 512 MB+)
- Przechyżnia dyskowa: 1 GB (zalecane 50 GB+ dla workspace i artefaktów)
- CPU: Brak specjalnych wymagań

### Zalecane dla środowiska produkcyjnego:

- RAM: 4 GB+ (w zależności od liczby równoczesnych buildów)
- Przechyżnia dyskowa: 100 GB+ na SSD
- CPU: 4+ rdzenie
- Sieć: Stabilne połączenie internetowe

## Wymagania programowe

**Java** - Jenkins wymaga Java 8 lub nowszej (zalecane Java 11 LTS):

Bash

```
# Sprawdzenie wersji Java
java -version

# Jeśli Java nie jest zainstalowana:
# Ubuntu/Debian
sudo apt update
sudo apt install openjdk-11-jdk

# CentOS/RHEL
sudo yum install java-11-openjdk-devel

# Ustawienie JAVA_HOME
```

```
echo 'export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64' >> ~/.bashrc
source ~/.bashrc
```

# Instalacja Jenkins

## Instalacja na Ubuntu/Debian

Bash

```
# Aktualizacja systemu
sudo apt update

# Instalacja Java (jeśli nie jest zainstalowana)
sudo apt install openjdk-11-jdk

# Dodanie klucza GPG Jenkins
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo tee \
  /usr/share/keyrings/jenkins-keyring.asc > /dev/null

# Dodanie repozytorium Jenkins
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null

# Aktualizacja listy pakietów
sudo apt update

# Instalacja Jenkins
sudo apt install jenkins

# Uruchomienie i włączenie Jenkins
sudo systemctl start jenkins
sudo systemctl enable jenkins

# Sprawdzenie statusu
sudo systemctl status jenkins
```

## Instalacja na CentOS/RHEL

Bash

```
# Instalacja Java
sudo yum install java-11-openjdk-devel
```

```
# Dodanie repozytorium Jenkins
sudo wget -O /etc/yum.repos.d/jenkins.repo \
    https://pkg.jenkins.io/redhat-stable/jenkins.repo

# Import klucza GPG
sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io.key

# Instalacja Jenkins
sudo yum install jenkins

# Uruchomienie i włączenie Jenkins
sudo systemctl start jenkins
sudo systemctl enable jenkins

# Sprawdzenie statusu
sudo systemctl status jenkins
```

## Instalacja za pomocą Docker

Bash

```
# Utworzenie wolumenu dla danych Jenkins
docker volume create jenkins_home

# Uruchomienie Jenkins w kontenerze
docker run -d \
    --name jenkins \
    -p 8080:8080 \
    -p 50000:50000 \
    -v jenkins_home:/var/jenkins_home \
    -v /var/run/docker.sock:/var/run/docker.sock \
    jenkins/jenkins:lts

# Sprawdzenie logów
docker logs jenkins

# Pobranie hasła administratora
docker exec jenkins cat /var/jenkins_home/secrets/initialAdminPassword
```

## Instalacja za pomocą Docker Compose

YAML



```
# docker-compose.yml
version: '3.8'

services:
  jenkins:
    image: jenkins/jenkins:lts
    container_name: jenkins
    restart: unless-stopped
    ports:
      - "8080:8080"
      - "50000:50000"
    volumes:
      - jenkins_home:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
    environment:
      - JAVA_OPTS=-Djenkins.install.runSetupWizard=false
    networks:
      - jenkins_network

  jenkins-agent:
    image: jenkins/ssh-agent:jdk11
    container_name: jenkins-agent
    restart: unless-stopped
    environment:
      - JENKINS_AGENT_SSH_PUBKEY=ssh-rsa AAAAB3NzaC1yc2E...
    networks:
      - jenkins_network

volumes:
  jenkins_home:

networks:
  jenkins_network:
    driver: bridge
```

## Bash

```
# Uruchomienie
docker-compose up -d

# Sprawdzenie statusu
docker-compose ps
```

## Instalacja na Windows

1. Pobierz plik jenkins.war z oficjalnej strony: <https://jenkins.io/download/>
2. Otwórz Command Prompt jako Administrator
3. Uruchom Jenkins:

Plain Text

```
java -jar jenkins.war --httpPort=8080
```

Alternatywnie, możesz pobrać installer Windows (.msi) i zainstalować Jenkins jako usługę Windows.

## Pierwsza konfiguracja

### Dostęp do interfejsu webowego

Po instalacji Jenkins jest dostępny pod adresem:

Plain Text

```
http://localhost:8080
```

Lub jeśli instalujesz na zdalnym serwerze:

Plain Text

```
http://your-server-ip:8080
```

## Odblokowanie Jenkins

Przy pierwszym uruchomieniu Jenkins wymaga odblokowania za pomocą hasła administratora:

Bash

```
# Lokalizacja hasła (instalacja systemowa)
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

```
# Lokalizacja hasła (Docker)
docker exec jenkins cat /var/jenkins_home/secrets/initialAdminPassword
```

Skopiuj hasło i wklej je w interfejsie webowym.

## Instalacja pluginów

Jenkins oferuje dwie opcje instalacji pluginów:

**Install suggested plugins** - instaluje zestaw najczęściej używanych pluginów (zalecane dla początkujących).

**Select plugins to install** - pozwala na wybór konkretnych pluginów.

Zalecane pluginy dla środowiska DevOps:

### Source Code Management:

- Git plugin
- GitHub plugin
- GitLab plugin

### Build Tools:

- Maven Integration plugin
- Gradle plugin
- NodeJS plugin
- Python plugin

### Deployment:

- Deploy to container plugin
- SSH plugin
- Kubernetes plugin
- AWS plugins

## Notifications:

- Email Extension plugin
- Slack Notification plugin
- Microsoft Teams plugin

## Security:

- Role-based Authorization Strategy
- LDAP plugin
- SAML plugin

## Pipeline:

- Pipeline plugin (zwykle już zainstalowany)
- Blue Ocean plugin
- Pipeline Stage View plugin

## Tworzenie użytkownika administratora

Po instalacji pluginów utwórz konto administratora:

1. Username: admin (lub inna nazwa)
2. Password: silne hasło
3. Full name: Twoje imię i nazwisko
4. E-mail address: Twój adres email

## Konfiguracja URL Jenkins

Ustaw URL Jenkins, który będzie używany w powiadomieniach i linkach:

Plain Text

<http://your-jenkins-server:8080/>

## Konfiguracja podstawowa

### Konfiguracja globalna

Przejdź do **Manage Jenkins > Configure System** aby skonfigurować podstawowe ustawienia:

#### Jenkins Location:

- Jenkins URL: <http://your-server:8080/>
- System Admin e-mail address: [admin@yourcompany.com](mailto:admin@yourcompany.com)

#### Global properties:

Bash

```
# Przykładowe zmienne środowiskowe
JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
MAVEN_HOME=/usr/share/maven
DOCKER_HOST=unix:///var/run/docker.sock
```

#### E-mail Notification:

Plain Text

```
SMTP server: smtp.gmail.com
Default user e-mail suffix: @yourcompany.com
Use SMTP Authentication: true
User Name: your-email@gmail.com
Password: your-app-password
Use SSL: true
SMTP Port: 465
```

## Konfiguracja bezpieczeństwa

Przejdź do **Manage Jenkins > Configure Global Security**:

## Security Realm:

- Jenkins' own user database
- Allow users to sign up: false (dla środowiska produkcyjnego)

## Authorization:

- Matrix-based security (dla zaawansowanego zarządzania uprawnieniami)
- Role-Based Strategy (jeśli zainstalowany plugin)

## CSRF Protection:

- Enable CSRF Protection: true

## Agent protocols:

- Disable deprecated protocols
- Enable only: Java Web Start Agent Protocol/4 (TLS encryption)

## Konfiguracja narzędzi

Przejdź do **Manage Jenkins > Global Tool Configuration:**

### JDK installations:

Plain Text

```
Name: JDK-11
JAVA_HOME: /usr/lib/jvm/java-11-openjdk-amd64
Install automatically: false
```

### Git installations:

Plain Text

```
Name: Default
Path to Git executable: git
Install automatically: false
```

## Maven installations:

Plain Text

```
Name: Maven-3.8
MAVEN_HOME: /usr/share/maven
Install automatically: true
Version: 3.8.6
```

## Docker installations:

Plain Text

```
Name: Docker
Installation root: /usr/bin/docker
Install automatically: false
```

# Konfiguracja agentów (nodes)

## Master-Agent architektura

Jenkins może działać w architekturze master-agent, gdzie:

**Master (Controller)** - główny serwer Jenkins, który:

- Zarządza interfejsem webowym
- Przechowuje konfigurację
- Planuje i dystrybuje buildy
- Monitoruje agentów

**Agent (Node)** - maszyny wykonujące buildy:

- Wykonują zadania przydzielone przez master
- Mogą być dedykowane dla konkretnych typów buildów
- Mogą działać na różnych systemach operacyjnych

## Dodawanie agenta SSH

1. Przejdź do **Manage Jenkins > Manage Nodes and Clouds**
2. Kliknij **New Node**
3. Wprowadź nazwę i wybierz **Permanent Agent**

Konfiguracja agenta SSH:

Plain Text

```
Name: linux-agent-01
Description: Linux build agent
Number of executors: 2
Remote root directory: /home/jenkins
Labels: linux docker maven
Usage: Use this node as much as possible
Launch method: Launch agents via SSH
Host: 192.168.1.100
Credentials: jenkins-ssh-key
Host Key Verification Strategy: Known hosts file Verification Strategy
```

## Przygotowanie agenta SSH

Na maszynie agenta:

Bash

```
# Utworzenie użytkownika jenkins
sudo useradd -m -s /bin/bash jenkins

# Przełączenie na użytkownika jenkins
sudo su - jenkins

# Generowanie klucza SSH (na master)
ssh-keygen -t rsa -b 4096 -C "jenkins@master"

# Kopiowanie klucza publicznego na agenta
ssh-copy-id jenkins@agent-ip

# Instalacja Java na agencie
sudo apt install openjdk-11-jdk
```



```
# Instalacja Docker (jeśli potrzebne)
sudo apt install docker.io
sudo usermod -aG docker jenkins

# Instalacja innych narzędzi
sudo apt install maven nodejs npm python3 python3-pip
```

## Dodawanie agenta Docker

Konfiguracja agenta Docker w **Manage Jenkins > Manage Nodes and Clouds > Configure Clouds**:

YAML

```
# Docker Cloud configuration
Docker Host URI: unix:///var/run/docker.sock
Enabled: true
Expose DOCKER_HOST: true

# Docker Agent Template
Labels: docker-agent
Docker Image: jenkins/agent:latest
Instance Capacity: 10
Remote File System Root: /home/jenkins
Usage: Only build jobs with label expressions matching this node
```

## Agent jako Docker container

Bash

```
# Uruchomienie agenta jako kontener
docker run -d \
  --name jenkins-agent \
  -e JENKINS_URL=http://jenkins-master:8080 \
  -e JENKINS_SECRET=your-agent-secret \
  -e JENKINS_AGENT_NAME=docker-agent \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v jenkins-agent-workspace:/home/jenkins \
  jenkins/agent:latest
```

## Backup i przywracanie

## Backup Jenkins

Bash

```
#!/bin/bash
# jenkins-backup.sh

JENKINS_HOME="/var/lib/jenkins"
BACKUP_DIR="/backup/jenkins"
DATE=$(date +%Y%m%d_%H%M%S)
BACKUP_FILE="jenkins-backup-$DATE.tar.gz"

# Utworzenie katalogu backup
mkdir -p $BACKUP_DIR

# Zatrzymanie Jenkins (opcjonalne dla pełnego backup)
sudo systemctl stop jenkins

# Tworzenie archiwum
sudo tar -czf "$BACKUP_DIR/$BACKUP_FILE" \
  --exclude="$JENKINS_HOME/workspace/*" \
  --exclude="$JENKINS_HOME/builds/*/archive" \
  --exclude="$JENKINS_HOME/logs/*" \
  --exclude="$JENKINS_HOME/.m2/repository/*" \
  -C / var/lib/jenkins

# Uruchomienie Jenkins
sudo systemctl start jenkins

# Usunięcie starych backupów (starszych niż 30 dni)
find $BACKUP_DIR -name "jenkins-backup-*.tar.gz" -mtime +30 -delete

echo "Backup completed: $BACKUP_DIR/$BACKUP_FILE"
```

## Automatyczny backup z cron

Bash

```
# Edycja crontab
sudo crontab -e

# Dodanie zadania (backup codziennie o 2:00)
0 2 * * * /usr/local/bin/jenkins-backup.sh
```

## Przywracanie z backup

Bash

```
#!/bin/bash
# jenkins-restore.sh

BACKUP_FILE="/backup/jenkins/jenkins-backup-20231201_020000.tar.gz"
JENKINS_HOME="/var/lib/jenkins"

# Zatrzymanie Jenkins
sudo systemctl stop jenkins

# Backup aktualnej konfiguracji
sudo mv $JENKINS_HOME $JENKINS_HOME.old

# Przywrócenie z backup
sudo tar -xzf $BACKUP_FILE -C /

# Ustawienie uprawnień
sudo chown -R jenkins:jenkins $JENKINS_HOME

# Uruchomienie Jenkins
sudo systemctl start jenkins

echo "Restore completed from: $BACKUP_FILE"
```

## Monitoring i logowanie

### Lokalizacja logów

Bash

```
# Logi systemowe Jenkins
sudo journalctl -u jenkins -f

# Logi aplikacji Jenkins
tail -f /var/log/jenkins/jenkins.log

# Logi w Docker
docker logs -f jenkins
```

# Konfiguracja logowania

W **Manage Jenkins > System Log**:

Plain Text

Logger: hudson.model.Run  
Level: FINE

Logger: hudson.plugins.git.GitSCM  
Level: FINE

Logger: jenkins.plugins.git.GitStep  
Level: FINE

## Monitoring zasobów

Bash

```
# Skrypt monitoringu Jenkins
#!/bin/bash
# jenkins-monitor.sh

JENKINS_URL="http://localhost:8080"
JENKINS_USER="admin"
JENKINS_TOKEN="your-api-token"

# Sprawdzenie statusu Jenkins
curl -s -u $JENKINS_USER:$JENKINS_TOKEN \
  "$JENKINS_URL/api/json?pretty=true" | \
  jq '.mode, .numExecutors, .quietingDown'

# Sprawdzenie aktywnych buildów
curl -s -u $JENKINS_USER:$JENKINS_TOKEN \
  "$JENKINS_URL/computer/api/json?pretty=true" | \
  jq '.computer[] | {displayName, idle, offline}'

# Sprawdzenie kolejki buildów
curl -s -u $JENKINS_USER:$JENKINS_TOKEN \
  "$JENKINS_URL/queue/api/json?pretty=true" | \
  jq '.items | length'
```

## Alerting

Konfiguracja alertów w **Manage Jenkins > Configure System**:

Bash

```
# Email notification dla failed builds
Post-build Actions:
- E-mail Notification
  Recipients: devops-team@company.com
  Send e-mail for every unstable build: true
  Send separate e-mails to individuals who broke the build: true
```

## Optymalizacja wydajności

### Konfiguracja JVM

Edycja pliku konfiguracyjnego Jenkins:

Bash

```
# Ubuntu/Debian: /etc/default/jenkins
# CentOS/RHEL: /etc/sysconfig/jenkins

JAVA_ARGS="-Djava.awt.headless=true"
JENKINS_ARGS="--webroot=/var/cache/$NAME/war --httpPort=$HTTP_PORT"

# Optymalizacja pamięci
JAVA_ARGS="$JAVA_ARGS -Xms2g -Xmx4g"
JAVA_ARGS="$JAVA_ARGS -XX:+UseG1GC"
JAVA_ARGS="$JAVA_ARGS -XX:+UseStringDeduplication"
JAVA_ARGS="$JAVA_ARGS -XX:+DisableExplicitGC"

# Optymalizacja dla Docker
JAVA_ARGS="$JAVA_ARGS -XX:+UnlockExperimentalVMOptions"
JAVA_ARGS="$JAVA_ARGS -XX:+UseCGroupMemoryLimitForHeap"
```

### Czyszczenie workspace

Plain Text

```
// Pipeline script dla czyszczenia
pipeline {
  agent any
```

```

options {
    // Zachowaj tylko ostatnie 10 buildów
    buildDiscarder(logRotator(numToKeepStr: '10'))

    // Timeout dla całego pipeline
    timeout(time: 1, unit: 'HOURS')

    // Wyczyść workspace przed buildem
    skipDefaultCheckout(true)
}

stages {
    stage('Cleanup') {
        steps {
            cleanWs()
        }
    }

    stage('Checkout') {
        steps {
            checkout scm
        }
    }
}
}

```

## Optymalizacja pluginów

### Bash

```

# Regularne aktualizacje pluginów
# Manage Jenkins > Manage Plugins > Updates

# Usuwanie nieużywanych pluginów
# Manage Jenkins > Manage Plugins > Installed

# Monitoring użycia pluginów
# Manage Jenkins > Plugin Usage

```

To kończy rozdział o instalacji i konfiguracji Jenkins. W następnym rozdziale omówimy tworzenie pipeline'ów Jenkins i praktyczne przykłady ich implementacji.

## 7. Tworzenie pipeline'ów Jenkins

# Wprowadzenie do Jenkins Pipeline

Jenkins Pipeline to zestaw pluginów, które wspierają implementację i integrację continuous delivery pipeline'ów w Jenkins. Pipeline pozwala na definiowanie całego procesu budowania, testowania i wdrażania aplikacji jako kod, który może być wersjonowany, przeglądany i współdzielony.

Pipeline oferuje kilka kluczowych korzyści:

**Pipeline as Code** - definicja pipeline'u jest przechowywana w repozytorium kodu razem z aplikacją, co zapewnia wersjonowanie i możliwość code review.

**Durability** - pipeline'y mogą przetrwać planowane i nieplanowane restarty Jenkins master.

**Pausable** - pipeline'y mogą opcjonalnie zatrzymać się i czekać na input użytkownika lub zatwierdzenie.

**Versatile** - pipeline'y wspierają złożone wymagania real-world CD, włączając możliwość fork/join, loop i wykonywania pracy równoległej.

**Extensible** - Pipeline plugin wspiera niestandardowe rozszerzenia do swojego DSL i wiele opcji integracji z innymi pluginami.

## Typy Pipeline'ów

### Declarative Pipeline

Declarative Pipeline to nowsza i zalecana składnia, która oferuje bogatszy model składniowy dla tworzenia pipeline'ów. Jest łatwiejsza do nauki i pisania.

Plain Text

```
pipeline {
  agent any

  stages {
    stage('Build') {
      steps {
        echo 'Building..'
      }
    }
  }
}
```

```

    }
    stage('Test') {
        steps {
            echo 'Testing..'
        }
    }
    stage('Deploy') {
        steps {
            echo 'Deploying....'
        }
    }
}

```

## Scripted Pipeline

Scripted Pipeline to pierwotna składnia pipeline'u, oparta na Groovy. Oferuje większą elastyczność, ale jest bardziej skomplikowana.

Plain Text

```

node {
    stage('Build') {
        echo 'Building..'
    }
    stage('Test') {
        echo 'Testing..'
    }
    stage('Deploy') {
        echo 'Deploying....'
    }
}

```

## Tworzenie pierwszego Pipeline'u

### Krok 1: Utworzenie nowego Job'a

1. W Jenkins dashboard kliknij **New Item**
2. Wprowadź nazwę: `my-first-pipeline`
3. Wybierz **Pipeline** i kliknij **OK**



## Krok 2: Konfiguracja Pipeline'u

W sekcji **Pipeline** wybierz **Pipeline script** i wprowadź:

Plain Text

```
pipeline {
  agent any

  environment {
    // Zmienne środowiskowe dostępne w całym pipeline
    APP_NAME = 'my-application'
    VERSION = '1.0.0'
  }

  stages {
    stage('Checkout') {
      steps {
        echo 'Checking out source code...'
        // W rzeczywistym projekcie:
        // git 'https://github.com/user/repo.git'
      }
    }

    stage('Build') {
      steps {
        echo "Building ${APP_NAME} version ${VERSION}"
        // Przykład budowania aplikacji Java
        // sh 'mvn clean compile'
      }
    }

    stage('Test') {
      steps {
        echo 'Running tests...'
        // sh 'mvn test'
      }
    }

    stage('Package') {
      steps {
        echo 'Packaging application...'
        // sh 'mvn package'
      }
    }

    stage('Deploy') {
```

```
        steps {
            echo 'Deploying to staging...'
            // sh 'docker build -t ${APP_NAME}:${VERSION} .'
            // sh 'docker push registry/${APP_NAME}:${VERSION}'
        }
    }
}

post {
    always {
        echo 'Pipeline completed!'
    }
    success {
        echo 'Pipeline succeeded!'
    }
    failure {
        echo 'Pipeline failed!'
    }
}
}
```

## Krok 3: Uruchomienie Pipeline'u

1. Kliknij **Save**
2. Kliknij **Build Now**
3. Obserwuj wykonanie w **Console Output**

## Pipeline z Git Integration

### Jenkinsfile w repozytorium

Utwórz plik `Jenkinsfile` w głównym katalogu repozytorium:

Plain Text

```
pipeline {
    agent any

    tools {
        maven 'Maven-3.8'
        jdk 'JDK-11'
    }
}
```

```

}

environment {
    DOCKER_REGISTRY = 'your-registry.com'
    IMAGE_NAME = 'my-app'
    KUBECONFIG = credentials('kubeconfig')
}

stages {
    stage('Checkout') {
        steps {
            checkout scm
            script {
                env.GIT_COMMIT_SHORT = sh(
                    script: "git rev-parse --short HEAD",
                    returnStdout: true
                ).trim()
                env.BUILD_VERSION =
"${env.BUILD_NUMBER}-${env.GIT_COMMIT_SHORT}"
            }
        }
    }

    stage('Build') {
        steps {
            sh 'mvn clean compile'
        }
    }

    stage('Unit Tests') {
        steps {
            sh 'mvn test'
        }
        post {
            always {
                publishTestResults testResultsPattern: 'target/surefire-
reports/*.xml'
                publishCoverage adapters:
[jacocoAdapter('target/site/jacoco/jacoco.xml')]
            }
        }
    }

    stage('Code Quality') {
        steps {
            withSonarQubeEnv('SonarQube') {
                sh 'mvn sonar:sonar'
            }
        }
    }
}

```

```

    }
}

stage('Package') {
    steps {
        sh 'mvn package -DskipTests'
        archiveArtifacts artifacts: 'target/*.jar', fingerprint: true
    }
}

stage('Docker Build') {
    steps {
        script {
            def image =
docker.build("${DOCKER_REGISTRY}/${IMAGE_NAME}:${BUILD_VERSION}")
            docker.withRegistry("https://${DOCKER_REGISTRY}",
'docker-registry-credentials') {
                image.push()
                image.push('latest')
            }
        }
    }
}

stage('Deploy to Staging') {
    steps {
        sh """
            kubectl set image deployment/${IMAGE_NAME} \

${IMAGE_NAME}=${DOCKER_REGISTRY}/${IMAGE_NAME}:${BUILD_VERSION} \
            --namespace=staging
        """
        sh 'kubectl rollout status deployment/${IMAGE_NAME} --
namespace=staging'
    }
}

stage('Integration Tests') {
    steps {
        sh 'mvn verify -Pintegration-tests'
    }
}

stage('Deploy to Production') {
    when {
        branch 'main'
    }
    steps {

```

```

        input message: 'Deploy to production?', ok: 'Deploy'
        sh """
            kubectl set image deployment/${IMAGE_NAME} \

${IMAGE_NAME}=${DOCKER_REGISTRY}/${IMAGE_NAME}:${BUILD_VERSION} \
            --namespace=production
        """
        sh 'kubectl rollout status deployment/${IMAGE_NAME} --
namespace=production'
    }
}
}

post {
    always {
        cleanWs()
    }
    success {
        slackSend(
            color: 'good',
            message: "✅ Pipeline succeeded: ${env.JOB_NAME} -
${env.BUILD_NUMBER}"
        )
    }
    failure {
        slackSend(
            color: 'danger',
            message: "❌ Pipeline failed: ${env.JOB_NAME} -
${env.BUILD_NUMBER}"
        )
    }
}
}
}

```

## Konfiguracja Pipeline z SCM

1. W konfiguracji job'a wybierz **Pipeline script from SCM**
2. Wybierz **Git** jako SCM
3. Wprowadź URL repozytorium
4. Skonfiguruj credentials jeśli potrzebne
5. Ustaw **Script Path** na `Jenkinsfile`

# Zaawansowane funkcje Pipeline

## Parallel Execution

Plain Text

```
pipeline {
  agent any

  stages {
    stage('Parallel Tests') {
      parallel {
        stage('Unit Tests') {
          steps {
            sh 'mvn test'
          }
        }
        stage('Integration Tests') {
          steps {
            sh 'mvn verify -Pintegration'
          }
        }
        stage('Security Tests') {
          steps {
            sh 'mvn verify -Psecurity'
          }
        }
      }
    }
  }
}
```

## Matrix Builds

Plain Text

```
pipeline {
  agent none

  stages {
    stage('Test') {
      matrix {
        axes {
          axis {
```

```

        name 'JAVA_VERSION'
        values '8', '11', '17'
    }
    axis {
        name 'OS'
        values 'linux', 'windows'
    }
}
stages {
    stage('Build and Test') {
        agent {
            label "${OS}"
        }
        tools {
            jdk "JDK-${JAVA_VERSION}"
        }
        steps {
            sh 'mvn clean test'
        }
    }
}
}
}
}
```

## Conditional Execution

Plain Text

```
pipeline {
  agent any

  stages {
    stage('Deploy to Dev') {
      when {
        branch 'develop'
      }
      steps {
        echo 'Deploying to development environment'
      }
    }

    stage('Deploy to Staging') {
      when {
        branch 'release/*'
      }
    }
  }
}
```

```

    }
    steps {
        echo 'Deploying to staging environment'
    }
}

stage('Deploy to Production') {
    when {
        allof {
            branch 'main'
            environment name: 'DEPLOY_TO_PROD', value: 'true'
        }
    }
    steps {
        echo 'Deploying to production environment'
    }
}

stage('Hotfix Deploy') {
    when {
        anyOf {
            branch 'hotfix/*'
            changeRequest target: 'main'
        }
    }
    steps {
        echo 'Deploying hotfix'
    }
}
}
}
}

```

## Input Steps

Plain Text

```

pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                echo 'Building application...'
            }
        }
    }
}

```



```

stage('Deploy Approval') {
    steps {
        script {
            def userInput = input(
                id: 'userInput',
                message: 'Deploy to production?',
                parameters: [
                    choice(
                        choices: ['Deploy', 'Abort'],
                        description: 'Choose action',
                        name: 'ACTION'
                    ),
                    string(
                        defaultValue: 'v1.0.0',
                        description: 'Version to deploy',
                        name: 'VERSION'
                    )
                ]
            )

            if (userInput.ACTION == 'Deploy') {
                env.DEPLOY_VERSION = userInput.VERSION
            } else {
                error('Deployment aborted by user')
            }
        }
    }
}

stage('Deploy') {
    steps {
        echo "Deploying version ${env.DEPLOY_VERSION}"
    }
}
}

```

## Pipeline dla różnych technologii

### Python Application Pipeline

Plain Text

```

pipeline {
    agent any
}

```

```

tools {
    // Wymaga Python plugin
    python 'Python-3.9'
}

environment {
    PYTHONPATH = "${WORKSPACE}"
    VIRTUAL_ENV = "${WORKSPACE}/venv"
}

stages {
    stage('Setup') {
        steps {
            sh '''
                python -m venv venv
                . venv/bin/activate
                pip install --upgrade pip
                pip install -r requirements.txt
                pip install -r requirements-dev.txt
            '''
        }
    }

    stage('Lint') {
        steps {
            sh '''
                . venv/bin/activate
                flake8 src/ tests/
                black --check src/ tests/
                isort --check-only src/ tests/
            '''
        }
    }

    stage('Test') {
        steps {
            sh '''
                . venv/bin/activate
                pytest tests/ --junitxml=test-results.xml --cov=src --
cov-report=xml
            '''
        }
        post {
            always {
                publishTestResults testResultsPattern: 'test-results.xml'
                publishCoverage adapters:
[coberturaAdapter('coverage.xml')]
            }
        }
    }
}

```

```

    }
  }
}

stage('Security Scan') {
  steps {
    sh '''
      . venv/bin/activate
      bandit -r src/ -f json -o bandit-report.json
      safety check --json --output safety-report.json
    '''
  }
}

stage('Build') {
  steps {
    sh '''
      . venv/bin/activate
      python setup.py sdist bdist_wheel
    '''
    archiveArtifacts artifacts: 'dist/*', fingerprint: true
  }
}

stage('Docker Build') {
  steps {
    script {
      def image = docker.build("python-
app:${env.BUILD_NUMBER}")
      image.push()
    }
  }
}
}
}

```

## Node.js Application Pipeline

Plain Text

```

pipeline {
  agent any

  tools {
    nodejs 'NodeJS-16'
  }
}

```

```
environment {
    NODE_ENV = 'test'
}

stages {
    stage('Install Dependencies') {
        steps {
            sh 'npm ci'
        }
    }

    stage('Lint') {
        steps {
            sh 'npm run lint'
        }
    }

    stage('Test') {
        steps {
            sh 'npm test'
        }
        post {
            always {
                publishTestResults testResultsPattern: 'test-results.xml'
                publishCoverage adapters:
[istanbulCoberturaAdapter('coverage/cobertura-coverage.xml')]
            }
        }
    }

    stage('Build') {
        steps {
            sh 'npm run build'
        }
    }

    stage('Security Audit') {
        steps {
            sh 'npm audit --audit-level moderate'
        }
    }

    stage('Docker Build') {
        steps {
            script {
                def image = docker.build("nodejs-
app:${env.BUILD_NUMBER}")
            }
        }
    }
}
```

```

    image.push()
  }
}
}
}
}

```

## .NET Application Pipeline

Plain Text

```

pipeline {
  agent any

  tools {
    dotnet 'DotNet-6'
  }

  stages {
    stage('Restore') {
      steps {
        sh 'dotnet restore'
      }
    }

    stage('Build') {
      steps {
        sh 'dotnet build --configuration Release --no-restore'
      }
    }

    stage('Test') {
      steps {
        sh '''
          dotnet test --no-build --configuration Release \
            --logger trx --results-directory TestResults \
            --collect:"XPlat Code Coverage"
        '''
      }
      post {
        always {
          publishTestResults testResultsPattern:
'TestResults/*.trx'
          publishCoverage adapters:
[coberturaAdapter('TestResults/*/coverage.cobertura.xml')]
        }
      }
    }
  }
}

```

```

    }
  }

  stage('Publish') {
    steps {
      sh 'dotnet publish --configuration Release --output
./publish'
      archiveArtifacts artifacts: 'publish/**', fingerprint: true
    }
  }

  stage('Docker Build') {
    steps {
      script {
        def image = docker.build("dotnet-
app:${env.BUILD_NUMBER}")
        image.push()
      }
    }
  }
}

```

## Shared Libraries

Shared Libraries pozwalają na współdzielenie kodu pipeline między różnymi projektami.

### Struktura Shared Library

Plain Text

```

vars/
  buildApp.groovy
  deployApp.groovy
  notifySlack.groovy
src/
  com/
    company/
      pipeline/
        Utils.groovy
resources/
  scripts/
    deploy.sh

```

## Przykład Shared Library

Plain Text

```
// vars/buildApp.groovy
def call(Map config) {
    pipeline {
        agent any

        stages {
            stage('Build') {
                steps {
                    script {
                        if (config.language == 'java') {
                            sh 'mvn clean package'
                        } else if (config.language == 'nodejs') {
                            sh 'npm run build'
                        } else if (config.language == 'python') {
                            sh 'python setup.py build'
                        }
                    }
                }
            }

            stage('Test') {
                steps {
                    script {
                        if (config.language == 'java') {
                            sh 'mvn test'
                        } else if (config.language == 'nodejs') {
                            sh 'npm test'
                        } else if (config.language == 'python') {
                            sh 'pytest'
                        }
                    }
                }
            }

            stage('Deploy') {
                when {
                    branch 'main'
                }
                steps {
                    deployApp([
                        environment: config.environment,
                        image: config.image
                    ])
                }
            }
        }
    }
}
```

```

    }
  }
}

post {
  always {
    notifySlack([
      status: currentBuild.result,
      channel: config.slackChannel
    ])
  }
}
}
}
}

```

#### Plain Text

```

// vars/deployApp.groovy
def call(Map config) {
  sh """
    kubectl set image deployment/${config.image} \
      ${config.image}=${config.image}:${env.BUILD_NUMBER} \
      --namespace=${config.environment}
  """

  sh """
    kubectl rollout status deployment/${config.image} \
      --namespace=${config.environment}
  """
}

```

## Użycie Shared Library

#### Plain Text

```

// Jenkinsfile w projekcie
@Library('company-pipeline-library') _

buildApp([
  language: 'java',
  environment: 'staging',
  image: 'my-java-app',
  slackChannel: '#deployments'
])

```



# Monitoring i debugging Pipeline

## Pipeline Logs

Plain Text

```
pipeline {
  agent any

  stages {
    stage('Debug Info') {
      steps {
        script {
          echo "Build Number: ${env.BUILD_NUMBER}"
          echo "Job Name: ${env.JOB_NAME}"
          echo "Workspace: ${env.WORKSPACE}"
          echo "Git Commit: ${env.GIT_COMMIT}"
          echo "Git Branch: ${env.GIT_BRANCH}"

          // Wyświetl wszystkie zmienne środowiskowe
          sh 'printenv | sort'
        }
      }
    }
  }
}
```

## Error Handling

Plain Text

```
pipeline {
  agent any

  stages {
    stage('Risky Operation') {
      steps {
        script {
          try {
            sh 'some-command-that-might-fail'
          } catch (Exception e) {
            echo "Command failed: ${e.getMessage()}"
            currentBuild.result = 'UNSTABLE'
          }
        }
      }
    }
  }
}
```

```

    }
  }
}

stage('Retry Logic') {
  steps {
    retry(3) {
      sh 'flaky-command'
    }
  }
}

stage('Timeout') {
  steps {
    timeout(time: 5, unit: 'MINUTES') {
      sh 'long-running-command'
    }
  }
}
}
}

```

## Pipeline Metrics

Plain Text

```

pipeline {
  agent any

  stages {
    stage('Metrics Collection') {
      steps {
        script {
          def startTime = System.currentTimeMillis()

          // Wykonaj operację
          sh 'mvn test'

          def duration = System.currentTimeMillis() - startTime
          echo "Test execution took: ${duration}ms"

          // Wyślij metryki do systemu monitoringu
          sh """
              curl -X POST http://metrics-server/api/metrics \
                -d
            'test_duration_ms=${duration}&job=${env.JOB_NAME}&build=${env.BUILD_NUMBER}'
          """
        }
      }
    }
  }
}

```

```
    """"  
    }  
    }  
    }  
    }  
    }
```

To kończy rozdział o tworzeniu pipeline'ów Jenkins. W następnym rozdziale omówimy integrację Jenkins z Git i zaawansowane scenariusze CI/CD.

## 8. Integracja Jenkins z Git

### Podstawy integracji Git z Jenkins

Integracja Jenkins z Git stanowi fundament nowoczesnych pipeline'ów CI/CD. Jenkins może automatycznie wykrywać zmiany w repozytoriach Git, uruchamiać buildy w odpowiedzi na commity, pull requesty i inne wydarzenia w repozytorium. Ta integracja umożliwia implementację prawdziwego continuous integration, gdzie każda zmiana kodu jest automatycznie budowana i testowana.

Jenkins oferuje kilka sposobów integracji z Git:

**Polling SCM** - Jenkins regularnie sprawdza repozytorium w poszukiwaniu zmian.

**Webhooks** - Git repository powiadamia Jenkins o zmianach w czasie rzeczywistym.

**Manual triggers** - buildy uruchamiane ręcznie przez użytkowników.

**Scheduled builds** - buildy uruchamiane według harmonogramu.

**Pipeline triggers** - buildy uruchamiane przez inne pipeline'y.

### Konfiguracja Git w Jenkins

#### Instalacja Git Plugin

Git plugin jest zwykle instalowany domyślnie, ale można go zainstalować ręcznie:

1. Przejdź do **Manage Jenkins > Manage Plugins**
2. W zakładce **Available** wyszukaj "Git plugin"
3. Zaznacz plugin i kliknij **Install without restart**

## Konfiguracja globalna Git

W **Manage Jenkins > Global Tool Configuration:**

Plain Text

```
Git installations:  
Name: Default  
Path to Git executable: git  
Install automatically: false
```

Dla zaawansowanej konfiguracji:

Plain Text

```
Name: Git-2.40  
Path to Git executable: /usr/local/bin/git  
Install automatically: true  
Version: 2.40.0
```

## Konfiguracja credentials

Przejdź do **Manage Jenkins > Manage Credentials:**

**Username/Password credentials:**

Plain Text

```
Kind: Username with password  
Scope: Global  
Username: your-git-username  
Password: your-git-password-or-token  
ID: git-credentials  
Description: Git repository credentials
```

## SSH Key credentials:

Plain Text

```
Kind: SSH Username with private key
Scope: Global
Username: git
Private Key: Enter directly (paste your private key)
ID: git-ssh-key
Description: Git SSH key
```

## GitHub Personal Access Token:

Plain Text

```
Kind: Secret text
Scope: Global
Secret: ghp_your_personal_access_token
ID: github-token
Description: GitHub Personal Access Token
```

# Podstawowa konfiguracja Job'a z Git

## Freestyle Project z Git

1. Utwórz nowy **Freestyle project**
2. W sekcji **Source Code Management** wybierz **Git**
3. Skonfiguruj repozytorium:

Plain Text

```
Repository URL: https://github.com/user/repository.git
Credentials: git-credentials
Branches to build: */main
Repository browser: github
URL: https://github.com/user/repository
```

## Zaawansowana konfiguracja Git

Plain Text

Additional Behaviours:

- Clean before checkout
- Clean after checkout
- Checkout to specific local branch: main
- Merge before build
- Prune stale remote-tracking branches
- Shallow clone (depth: 1)
- Advanced clone behaviours:
  - Honor refs spec on initial clone: true
  - Shallow clone depth: 50

## Build Triggers

### Poll SCM:

Plain Text

Schedule: H/5 \* \* \* \* # Sprawdzaj co 5 minut

### GitHub hook trigger for GITScm polling:

- Zaznacz tę opcję dla webhook integration

### Build when a change is pushed to GitHub:

- Wymaga GitHub plugin

## Webhooks Configuration

### GitHub Webhooks

1. W repozytorium GitHub przejdź do **Settings > Webhooks**
2. Kliknij **Add webhook**
3. Skonfiguruj webhook:

Plain Text

```
Payload URL: http://your-jenkins-server:8080/github-webhook/  
Content type: application/json  
Secret: (opcjonalne, ale zalecane)  
Which events: Just the push event  
Active: true
```

## GitLab Webhooks

1. W projekcie GitLab przejdź do **Settings > Webhooks**
2. Skonfiguruj webhook:

Plain Text

```
URL: http://your-jenkins-server:8080/project/your-job-name  
Secret Token: your-secret-token  
Trigger: Push events, Tag push events, Merge request events  
SSL verification: Enable (jeśli używasz HTTPS)
```

## Bitbucket Webhooks

1. W repozytorium Bitbucket przejdź do **Settings > Webhooks**
2. Dodaj webhook:

Plain Text

```
Title: Jenkins CI  
URL: http://your-jenkins-server:8080/bitbucket-hook/  
Status: Active  
Triggers: Repository push
```

## Pipeline z Git Integration

### Podstawowy Jenkinsfile

Plain Text

```

pipeline {
    agent any

    triggers {
        // Poll SCM co 5 minut
        pollSCM('H/5 * * * *')

        // Webhook trigger (wymaga konfiguracji webhook)
        githubPush()
    }

    stages {
        stage('Checkout') {
            steps {
                // Checkout jest automatyczny dla pipeline z SCM
                checkout scm

                // Lub explicit checkout z dodatkowymi opcjami
                checkout([
                    $class: 'GitSCM',
                    branches: [[name: '*/main']],
                    userRemoteConfigs: [[
                        url: 'https://github.com/user/repo.git',
                        credentialsId: 'git-credentials'
                    ]],
                    extensions: [
                        [$class: 'CleanBeforeCheckout'],
                        [$class: 'CloneOption', depth: 1, shallow: true]
                    ]
                ])
            }
        }

        stage('Build Info') {
            steps {
                script {
                    // Pobierz informacje o commit
                    env.GIT_COMMIT_SHORT = sh(
                        script: "git rev-parse --short HEAD",
                        returnStdout: true
                    ).trim()

                    env.GIT_COMMIT_MSG = sh(
                        script: "git log -1 --pretty=format:'%s'",
                        returnStdout: true
                    ).trim()
                }
            }
        }
    }
}

```





## Plain Text

```
pipeline {
  agent any

  environment {
    DOCKER_REGISTRY = 'your-registry.com'
    IMAGE_NAME = 'my-app'
  }

  stages {
    stage('Build') {
      steps {
        sh 'mvn clean compile'
      }
    }

    stage('Test') {
      steps {
        sh 'mvn test'
      }
    }

    stage('Deploy to Dev') {
      when {
        branch 'develop'
      }
      steps {
        echo 'Deploying to development environment'
        sh 'mvn deploy -Pdev'
      }
    }

    stage('Deploy to Staging') {
      when {
        branch 'release/*'
      }
      steps {
        echo 'Deploying to staging environment'
        sh 'mvn deploy -Pstaging'
      }
    }

    stage('Deploy to Production') {
      when {
        branch 'main'
      }
      steps {
        input message: 'Deploy to production?', ok: 'Deploy'
```

```

        echo 'Deploying to production environment'
        sh 'mvn deploy -Pprod'
    }
}

stage('PR Validation') {
    when {
        changeRequest()
    }
    steps {
        echo 'Validating pull request'
        sh 'mvn verify -Ppr-validation'
    }
}

post {
    always {
        // Publikuj wyniki testów
        publishTestResults testResultsPattern: 'target/surefire-
reports/*.xml'
    }

    success {
        script {
            if (env.BRANCH_NAME == 'main') {
                // Tag successful production builds
                sh """
                    git tag -a v${env.BUILD_NUMBER} -m "Release
v${env.BUILD_NUMBER}"
                    git push origin v${env.BUILD_NUMBER}
                """
            }
        }
    }
}
}

```

## Git Flow z Jenkins

### Implementacja Git Flow

Plain Text

```

pipeline {
    agent any

    stages {
        stage('Determine Strategy') {
            steps {
                script {
                    if (env.BRANCH_NAME.startsWith('feature/')) {
                        env.DEPLOY_ENV = 'none'
                        env.RUN_TESTS = 'unit'
                    } else if (env.BRANCH_NAME == 'develop') {
                        env.DEPLOY_ENV = 'dev'
                        env.RUN_TESTS = 'integration'
                    } else if (env.BRANCH_NAME.startsWith('release/')) {
                        env.DEPLOY_ENV = 'staging'
                        env.RUN_TESTS = 'full'
                    } else if (env.BRANCH_NAME == 'main') {
                        env.DEPLOY_ENV = 'production'
                        env.RUN_TESTS = 'smoke'
                    } else if (env.BRANCH_NAME.startsWith('hotfix/')) {
                        env.DEPLOY_ENV = 'staging'
                        env.RUN_TESTS = 'critical'
                    }

                    echo "Branch: ${env.BRANCH_NAME}"
                    echo "Deploy Environment: ${env.DEPLOY_ENV}"
                    echo "Test Suite: ${env.RUN_TESTS}"
                }
            }
        }

        stage('Build') {
            steps {
                sh 'mvn clean compile'
            }
        }

        stage('Unit Tests') {
            when {
                expression { env.RUN_TESTS in ['unit', 'integration', 'full'] }
            }
            steps {
                sh 'mvn test'
            }
        }
    }
}

```

```

stage('Integration Tests') {
    when {
        expression { env.RUN_TESTS in ['integration', 'full'] }
    }
    steps {
        sh 'mvn verify -Pintegration'
    }
}

stage('Full Test Suite') {
    when {
        expression { env.RUN_TESTS == 'full' }
    }
    steps {
        parallel(
            'Performance Tests': {
                sh 'mvn verify -Pperformance'
            },
            'Security Tests': {
                sh 'mvn verify -Psecurity'
            },
            'UI Tests': {
                sh 'mvn verify -Pui'
            }
        )
    }
}

stage('Deploy') {
    when {
        expression { env.DEPLOY_ENV != 'none' }
    }
    steps {
        script {
            if (env.DEPLOY_ENV == 'production') {
                input message: 'Deploy to production?', ok: 'Deploy'
            }

            sh "mvn deploy -P${env.DEPLOY_ENV}"
        }
    }
}
}

```

## Pull Request Integration

# GitHub Pull Request Builder

Instalacja GitHub Pull Request Builder plugin:

1. **Manage Jenkins > Manage Plugins**
2. Zainstaluj "GitHub Pull Request Builder"

Konfiguracja:

Plain Text

```
pipeline {
    agent any

    triggers {
        githubPullRequests(
            triggerMode: 'HEAVY_HOOKS',
            events: [
                pullRequestOpened(),
                pullRequestUpdated(),
                pullRequestSynchronize()
            ]
        )
    }

    stages {
        stage('PR Validation') {
            when {
                changeRequest()
            }
            steps {
                script {
                    // Pobierz informacje o PR
                    env.PR_NUMBER = env.CHANGE_ID
                    env.PR_TITLE = env.CHANGE_TITLE
                    env.PR_AUTHOR = env.CHANGE_AUTHOR
                    env.TARGET_BRANCH = env.CHANGE_TARGET

                    echo "PR #${env.PR_NUMBER}: ${env.PR_TITLE}"
                    echo "Author: ${env.PR_AUTHOR}"
                    echo "Target: ${env.TARGET_BRANCH}"
                }

                // Uruchom walidację PR
                sh 'mvn clean verify -Ppr-validation'
```

```

        // Sprawdź czy nie ma konfliktów
        sh '''
            git fetch origin ${TARGET_BRANCH}
            git merge-base --is-ancestor HEAD origin/${TARGET_BRANCH}

|| {

                echo "Branch is not up to date with target"
                exit 1
            }
        '''
    }
}

stage('Code Quality Check') {
    when {
        changeRequest()
    }
    steps {
        // SonarQube analysis dla PR
        withSonarQubeEnv('SonarQube') {
            sh '''
                mvn sonar:sonar \
                    -Dsonar.pullrequest.key=${PR_NUMBER} \
                    -Dsonar.pullrequest.branch=${BRANCH_NAME} \
                    -Dsonar.pullrequest.base=${TARGET_BRANCH}
            '''
        }
    }
}

post {
    always {
        script {
            if (env.CHANGE_ID) {
                // Dodaj komentarz do PR z wynikami
                def status = currentBuild.result ?: 'SUCCESS'
                def message = ""
                ## Jenkins Build Results

                **Status:** ${status}
                **Build:** [#${env.BUILD_NUMBER}](${env.BUILD_URL})
                **Commit:** ${env.GIT_COMMIT_SHORT}

                ${status == 'SUCCESS' ? '✅ All checks passed!' : '❌
Build failed!'}

                """"

                pullRequest.comment(message)
            }
        }
    }
}

```

```

    }
  }
}

success {
  script {
    if (env.CHANGE_ID) {
      // Ustaw status check na GitHub
      githubNotify(
        status: 'SUCCESS',
        description: 'Jenkins build succeeded',
        context: 'continuous-integration/jenkins'
      )
    }
  }
}

failure {
  script {
    if (env.CHANGE_ID) {
      githubNotify(
        status: 'FAILURE',
        description: 'Jenkins build failed',
        context: 'continuous-integration/jenkins'
      )
    }
  }
}
}
}

```

## Zaawansowane Git operacje w Pipeline

### Git Tagging

Plain Text

```

stage('Tag Release') {
  when {
    branch 'main'
  }
  steps {
    script {
      // Pobierz ostatni tag
      def lastTag = sh(

```



```

        script: "git describe --tags --abbrev=0 2>/dev/null || echo
'v0.0.0'",
        returnStdout: true
    ).trim()

    // Oblicz nowy tag (semantic versioning)
    def (major, minor, patch) = lastTag.replaceAll('v',
'').split('\\.').toList()
    def newTag = "v${major}.${minor}.${patch.toInteger() + 1}"

    // Utwórz tag
    sh """
        git tag -a ${newTag} -m "Release ${newTag} - Build
#${env.BUILD_NUMBER}"
        git push origin ${newTag}
    """

    env.RELEASE_TAG = newTag
}
}
}

```

## Git Merge Operations

Plain Text

```

stage('Auto Merge') {
    when {
        allOf {
            changeRequest()
            expression { pullRequest.mergeable }
        }
    }
    steps {
        script {
            // Sprawdź czy PR może być automatycznie zmergowany
            if (currentBuild.result == 'SUCCESS') {
                // Merge PR
                pullRequest.merge(
                    commitTitle: "Merge PR #${env.CHANGE_ID}:
${env.CHANGE_TITLE}",
                    commitMessage: "Automatically merged by Jenkins",
                    mergeMethod: 'merge' // 'merge', 'squash', or 'rebase'
                )
            }
        }
    }
}

```

```
}  
}
```

## Git Submodules

Plain Text

```
stage('Update Submodules') {  
    steps {  
        sh '''  
            git submodule update --init --recursive  
            git submodule foreach git pull origin main  
        '''  
    }  
}
```

## Monitoring Git Integration

### Git Metrics Collection

Plain Text

```
pipeline {  
    agent any  
  
    stages {  
        stage('Git Metrics') {  
            steps {  
                script {  
                    // Zbierz metryki Git  
                    def commitCount = sh(  
                        script: "git rev-list --count HEAD",  
                        returnStdout: true  
                    ).trim()  
  
                    def lastCommitDate = sh(  
                        script: "git log -1 --format=%ci",  
                        returnStdout: true  
                    ).trim()  
  
                    def changedFiles = sh(  
                        script: "git diff --name-only HEAD~1 HEAD | wc -l",  
                        returnStdout: true  
                    )  
                }  
            }  
        }  
    }  
}
```

```

    ).trim()

    def linesAdded = sh(
        script: "git diff --numstat HEAD~1 HEAD | awk
'{add+=\$1} END {print add+0}'",
        returnStdout: true
    ).trim()

    def linesDeleted = sh(
        script: "git diff --numstat HEAD~1 HEAD | awk
'{del+=\$2} END {print del+0}'",
        returnStdout: true
    ).trim()

    // Wyślij metryki do systemu monitoringu
    sh """
        curl -X POST http://metrics-server/api/git-metrics \
        -H 'Content-Type: application/json' \
        -d '{
            "repository": "${env.JOB_NAME}",
            "branch": "${env.BRANCH_NAME}",
            "commit": "${env.GIT_COMMIT}",
            "commit_count": ${commitCount},
            "last_commit_date": "${lastCommitDate}",
            "files_changed": ${changedFiles},
            "lines_added": ${linesAdded},
            "lines_deleted": ${linesDeleted},
            "build_number": ${env.BUILD_NUMBER}
        }'
    """
}

}

}

}

```

## Git Hook Monitoring

## Bash

```
#!/bin/bash
# post-receive hook w repozytorium Git

while read oldrev newrev refname; do
    branch=$(git rev-parse --symbolic --abbrev-ref $refname)
```

```
# Powiadom Jenkins o push
curl -X POST "http://jenkins-server:8080/git/notifyCommit?url=$(pwd)&branches=${branch}" \
    --user jenkins:api-token

# Wyślij metryki
commit_count=$(git rev-list --count $newrev)
author=$(git log -1 --format=%an $newrev)

curl -X POST http://metrics-server/api/git-push \
    -H 'Content-Type: application/json' \
    -d "{
        \"repository\": \"$(basename $(pwd) .git)\",
        \"branch\": \"${branch}\",
        \"commit\": \"${newrev}\",
        \"author\": \"${author}\",
        \"commit_count\": ${commit_count}
    }"

done
```

## Troubleshooting Git Integration

### Częste problemy i rozwiązania

#### Problem: Git authentication failures

Plain Text

```
// Rozwiązanie: Użyj withCredentials
stage('Secure Git Operations') {
    steps {
        withCredentials([usernamePassword(
            credentialsId: 'git-credentials',
            usernameVariable: 'GIT_USERNAME',
            passwordVariable: 'GIT_PASSWORD'
        )]) {
            sh '''
                git config --global credential.helper store
                echo "https://${GIT_USERNAME}:${GIT_PASSWORD}@github.com" >
~/.git-credentials
                git push origin main
            '''
        }
    }
}
```

```
}  
}
```

## Problem: Large repository clone times

Plain Text

```
// Rozwiązanie: Shallow clone  
checkout([  
  $class: 'GitSCM',  
  branches: [[name: '*/main']],  
  extensions: [  
    [$class: 'CloneOption', depth: 1, shallow: true],  
    [$class: 'SubmoduleOption', disableSubmodules: true]  
  ],  
  userRemoteConfigs: [[url: 'https://github.com/user/repo.git']]  
])
```

## Problem: Git LFS files

Plain Text

```
stage('Git LFS') {  
  steps {  
    sh '''  
      git lfs install  
      git lfs pull  
    '''  
  }  
}
```

## Problem: Merge conflicts in automated merges

Plain Text

```
stage('Safe Merge') {  
  steps {  
    script {  
      try {  
        sh 'git merge origin/main'  
      } catch (Exception e) {  
        echo "Merge conflict detected: ${e.getMessage()}"  
        currentBuild.result = 'UNSTABLE'  
      }  
    }  
  }  
}
```

```
        // Powiadom zespół o konflikcie
        slackSend(
            color: 'warning',
            message: "⚠ Merge conflict in ${env.JOB_NAME} - manual
intervention required"
        )
    }
}
}
```

To kończy rozdział o integracji Jenkins z Git. Ta integracja jest fundamentalna dla każdego nowoczesnego pipeline'u CI/CD i umożliwia automatyzację całego procesu od commita do wdrożenia.

## 9. Python dla DevOps - podstawy

### Dlaczego Python w DevOps?

Python stał się jednym z najważniejszych języków programowania w ekosystemie DevOps ze względu na swoją prostotę, czytelność i bogaty ekosystem bibliotek. Język ten idealnie nadaje się do automatyzacji zadań, tworzenia skryptów deployment, zarządzania infrastrukturą i integracji różnych narzędzi DevOps.

Kluczowe zalety Python w DevOps:

**Prostota składni** - Python charakteryzuje się czytelną i intuicyjną składnią, co sprawia, że skrypty są łatwe do napisania, zrozumienia i utrzymania. To szczególnie ważne w środowiskach DevOps, gdzie kod często jest współdzielony między różnymi zespołami.

**Bogaty ekosystem bibliotek** - Python oferuje tysiące bibliotek dostępnych przez PyPI (Python Package Index), które pokrywają praktycznie każdy aspekt DevOps: od automatyzacji infrastruktury (Ansible, Fabric) przez monitoring (Prometheus client libraries) po deployment (Docker SDK, Kubernetes client).

**Cross-platform compatibility** - Python działa na wszystkich głównych systemach operacyjnych (Linux, Windows, macOS), co jest kluczowe w heterogenicznych środowiskach IT.

**Integracja z narzędziami DevOps** - większość nowoczesnych narzędzi DevOps oferuje Python SDK lub API, co umożliwia łatwą integrację i automatyzację.

**Spółeczność i wsparcie** - Python ma jedną z największych i najbardziej aktywnych społeczności programistycznych, co oznacza bogactwo dokumentacji, tutoriali i gotowych rozwiązań.

## Instalacja i konfiguracja środowiska Python

### Instalacja Python

#### Linux (Ubuntu/Debian):

Bash

```
# Aktualizacja systemu
sudo apt update

# Instalacja Python 3 i pip
sudo apt install python3 python3-pip python3-venv

# Sprawdzenie wersji
python3 --version
pip3 --version

# Utworzenie aliasu (opcjonalne)
echo 'alias python=python3' >> ~/.bashrc
echo 'alias pip=pip3' >> ~/.bashrc
source ~/.bashrc
```

#### Linux (CentOS/RHEL):

Bash

```
# CentOS 8/RHEL 8
sudo dnf install python3 python3-pip

# CentOS 7/RHEL 7
sudo yum install python3 python3-pip

# Sprawdzenie wersji
```

```
python3 --version
pip3 --version
```

## macOS:

Bash

```
# Używając Homebrew (zalecane)
brew install python

# Lub pobierz installer z python.org
# Sprawdzenie wersji
python3 --version
pip3 --version
```

## Windows:

1. Pobierz installer z <https://python.org/downloads/>
2. Uruchom installer i zaznacz "Add Python to PATH"
3. Sprawdź instalację w Command Prompt:

Plain Text

```
python --version
pip --version
```

## Konfiguracja środowiska wirtualnego

Środowiska wirtualne są kluczowe w Python DevOps, ponieważ pozwalają na izolację zależności między projektami:

Bash

```
# Utworzenie środowiska wirtualnego
python3 -m venv devops-env

# Aktywacja środowiska
# Linux/macOS:
source devops-env/bin/activate
```



```
# Windows:
devops-env\Scripts\activate

# Sprawdzenie aktywacji
which python
which pip

# Aktualizacja pip
pip install --upgrade pip

# Deaktywacja środowiska
deactivate
```

## Instalacja kluczowych bibliotek DevOps

Bash

```
# Aktywuj środowisko wirtualne
source devops-env/bin/activate

# Podstawowe biblioteki DevOps
pip install requests          # HTTP requests
pip install paramiko          # SSH connections
pip install fabric            # Remote execution
pip install ansible           # Configuration management
pip install docker            # Docker integration
pip install kubernetes        # Kubernetes client
pip install boto3             # AWS SDK
pip install azure-mgmt        # Azure SDK
pip install google-cloud      # Google Cloud SDK
pip install terraform-python  # Terraform integration
pip install jenkins-python    # Jenkins API
pip install gitpython         # Git operations
pip install pyyaml            # YAML parsing
pip install jinja2            # Templating
pip install click             # CLI creation
pip install rich              # Rich terminal output
pip install loguru            # Advanced logging

# Zapisanie zależności
pip freeze > requirements.txt
```

## Konfiguracja IDE dla DevOps

### Visual Studio Code:

Bash

```
# Instalacja rozszerzeń
code --install-extension ms-python.python
code --install-extension ms-python.pylint
code --install-extension ms-python.black-formatter
code --install-extension ms-vscode.vscode-yaml
code --install-extension redhat.ansible
code --install-extension ms-kubernetes-tools.vscode-kubernetes-tools
```

## PyCharm konfiguracja:

1. File > Settings > Project > Python Interpreter
2. Wybierz interpreter z środowiska wirtualnego
3. Zainstaluj pluginy: Docker, Kubernetes, Ansible

# Podstawy Python dla DevOps

## Praca z plikami i systemem

Python

```
#!/usr/bin/env python3
"""
Podstawowe operacje na plikach i systemie
"""

import os
import sys
import shutil
import subprocess
from pathlib import Path

def read_config_file(file_path):
    """Odczytaj plik konfiguracyjny"""
    try:
        with open(file_path, 'r') as file:
            return file.read()
    except FileNotFoundError:
        print(f"Plik {file_path} nie został znaleziony")
        return None
```

```

except PermissionError:
    print(f"Brak uprawnień do odczytu pliku {file_path}")
    return None

def write_config_file(file_path, content):
    """Zapisz plik konfiguracyjny"""
    try:
        # Utwórz katalog jeśli nie istnieje
        Path(file_path).parent.mkdir(parents=True, exist_ok=True)

        with open(file_path, 'w') as file:
            file.write(content)
        print(f"Plik {file_path} został zapisany")
        return True
    except Exception as e:
        print(f"Błąd podczas zapisywania pliku: {e}")
        return False

def execute_command(command, capture_output=True):
    """Wykonaj polecenie systemowe"""
    try:
        result = subprocess.run(
            command,
            shell=True,
            capture_output=capture_output,
            text=True,
            check=True
        )
        return {
            'success': True,
            'stdout': result.stdout,
            'stderr': result.stderr,
            'returncode': result.returncode
        }
    except subprocess.CalledProcessError as e:
        return {
            'success': False,
            'stdout': e.stdout,
            'stderr': e.stderr,
            'returncode': e.returncode
        }

def check_disk_space(path="/"):
    """Sprawdź dostępne miejsce na dysku"""
    statvfs = os.statvfs(path)

    # Oblicz rozmiary w GB
    total_space = (statvfs.f_frsize * statvfs.f_blocks) / (1024**3)

```

```

free_space = (statvfs.f_frsize * statvfs.f_available) / (1024**3)
used_space = total_space - free_space

return {
    'total_gb': round(total_space, 2),
    'used_gb': round(used_space, 2),
    'free_gb': round(free_space, 2),
    'usage_percent': round((used_space / total_space) * 100, 2)
}

def backup_directory(source_dir, backup_dir):
    """Utwórz backup katalogu"""
    try:
        if os.path.exists(backup_dir):
            shutil.rmtree(backup_dir)

        shutil.copytree(source_dir, backup_dir)
        print(f"Backup utworzony: {source_dir} -> {backup_dir}")
        return True
    except Exception as e:
        print(f"Błąd podczas tworzenia backup: {e}")
        return False

# Przykład użycia
if __name__ == "__main__":
    # Sprawdź miejsce na dysku
    disk_info = check_disk_space()
    print(f"Dysk: {disk_info['used_gb']}GB / {disk_info['total_gb']}GB "
          f"{disk_info['usage_percent']}%")

    # Wykonaj polecenie systemowe
    result = execute_command("ls -la /tmp")
    if result['success']:
        print("Zawartość /tmp:")
        print(result['stdout'])

    # Utwórz przykładowy plik konfiguracyjny
    config_content = """
# Konfiguracja aplikacji
app_name = "DevOps App"
version = "1.0.0"
debug = true
port = 8080
"""
    write_config_file("/tmp/app_config.txt", config_content)

```

## Praca z API i HTTP

## Python

```
#!/usr/bin/env python3
"""
Praca z API i HTTP requests w DevOps
"""

import requests
import json
import time
from typing import Dict, Any, Optional

class APIClient:
    """Klasa do obsługi API requests"""

    def __init__(self, base_url: str, auth_token: Optional[str] = None):
        self.base_url = base_url.rstrip('/')
        self.session = requests.Session()

        if auth_token:
            self.session.headers.update({
                'Authorization': f'Bearer {auth_token}'
            })

        self.session.headers.update({
            'Content-Type': 'application/json',
            'User-Agent': 'DevOps-Python-Client/1.0'
        })

    def get(self, endpoint: str, params: Optional[Dict] = None) -> Dict[str, Any]:
        """Wykonaj GET request"""
        url = f"{self.base_url}/{endpoint.lstrip('/')}"

        try:
            response = self.session.get(url, params=params, timeout=30)
            response.raise_for_status()
            return {
                'success': True,
                'data': response.json() if response.content else None,
                'status_code': response.status_code
            }
        except requests.exceptions.RequestException as e:
            return {
                'success': False,
                'error': str(e),
                'status_code': getattr(e.response, 'status_code', None)
            }
```

```

def post(self, endpoint: str, data: Dict[str, Any]) -> Dict[str, Any]:
    """Wykonaj POST request"""
    url = f"{self.base_url}/{endpoint.lstrip('/')}"

    try:
        response = self.session.post(
            url,
            data=json.dumps(data),
            timeout=30
        )
        response.raise_for_status()
        return {
            'success': True,
            'data': response.json() if response.content else None,
            'status_code': response.status_code
        }
    except requests.exceptions.RequestException as e:
        return {
            'success': False,
            'error': str(e),
            'status_code': getattr(e.response, 'status_code', None)
        }

def health_check(self) -> bool:
    """Sprawdź dostępność API"""
    try:
        response = self.session.get(f"{self.base_url}/health",
        timeout=10)
        return response.status_code == 200
    except:
        return False

def monitor_service_health(services: Dict[str, str], interval: int = 60):
    """Monitoruj zdrowie serwisów"""
    while True:
        print(f"\n=== Health Check - {time.strftime('%Y-%m-%d %H:%M:%S')}")

        for service_name, service_url in services.items():
            client = APIClient(service_url)
            is_healthy = client.health_check()

            status = "✅ HEALTHY" if is_healthy else "❌ UNHEALTHY"
            print(f"{service_name}: {status}")

            if not is_healthy:
                # Wyślij alert (przykład)

```

```

        send_alert(f"Service {service_name} is unhealthy!")

        time.sleep(interval)

def send_alert(message: str):
    """Wyślij alert (przykład implementacji)"""
    # Slack webhook
    slack_webhook = "https://hooks.slack.com/services/YOUR/WEBHOOK/URL"

    payload = {
        "text": f"🚨 ALERT: {message}",
        "username": "DevOps Bot",
        "icon_emoji": ":warning:"
    }

    try:
        response = requests.post(slack_webhook, json=payload, timeout=10)
        if response.status_code == 200:
            print(f"Alert sent: {message}")
        else:
            print(f"Failed to send alert: {response.status_code}")
    except Exception as e:
        print(f"Error sending alert: {e}")

# Przykład użycia
if __name__ == "__main__":
    # Testowanie API
    api = APIClient("https://api.github.com")

    # Pobierz informacje o użytkowniku
    result = api.get("/users/octocat")
    if result['success']:
        user_data = result['data']
        print(f"User: {user_data['name']}")
        print(f"Public repos: {user_data['public_repos']}")

    # Monitorowanie serwisów
    services_to_monitor = {
        "GitHub API": "https://api.github.com",
        "Google": "https://www.google.com",
        "Local Service": "http://localhost:8080"
    }

    # Uruchom monitoring (odkomentuj aby uruchomić)
    # monitor_service_health(services_to_monitor, interval=30)

```

## Praca z YAML i JSON

## Python

```
#!/usr/bin/env python3
"""
Praca z plikami konfiguracyjnymi YAML i JSON
"""

import yaml
import json
import os
from typing import Dict, Any, Union

class ConfigManager:
    """Klasa do zarządzania plikami konfiguracyjnymi"""

    @staticmethod
    def load_yaml(file_path: str) -> Union[Dict[str, Any], None]:
        """Wczytaj plik YAML"""
        try:
            with open(file_path, 'r', encoding='utf-8') as file:
                return yaml.safe_load(file)
        except FileNotFoundError:
            print(f"Plik YAML {file_path} nie został znaleziony")
            return None
        except yaml.YAMLError as e:
            print(f"Błąd parsowania YAML: {e}")
            return None

    @staticmethod
    def save_yaml(data: Dict[str, Any], file_path: str) -> bool:
        """Zapisz dane do pliku YAML"""
        try:
            os.makedirs(os.path.dirname(file_path), exist_ok=True)
            with open(file_path, 'w', encoding='utf-8') as file:
                yaml.dump(data, file, default_flow_style=False,
                          allow_unicode=True, indent=2)
            return True
        except Exception as e:
            print(f"Błąd zapisywania YAML: {e}")
            return False

    @staticmethod
    def load_json(file_path: str) -> Union[Dict[str, Any], None]:
        """Wczytaj plik JSON"""
        try:
            with open(file_path, 'r', encoding='utf-8') as file:
                return json.load(file)
        except FileNotFoundError:
```



```

        print(f"Plik JSON {file_path} nie został znaleziony")
        return None
    except json.JSONDecodeError as e:
        print(f"Błąd parsowania JSON: {e}")
        return None

    @staticmethod
    def save_json(data: Dict[str, Any], file_path: str, indent: int = 2) ->
bool:
        """Zapisz dane do pliku JSON"""
        try:
            os.makedirs(os.path.dirname(file_path), exist_ok=True)
            with open(file_path, 'w', encoding='utf-8') as file:
                json.dump(data, file, indent=indent, ensure_ascii=False)
            return True
        except Exception as e:
            print(f"Błąd zapisywania JSON: {e}")
            return False

def create_kubernetes_deployment(app_name: str, image: str, replicas: int =
3) -> Dict[str, Any]:
    """Utwórz konfigurację Kubernetes Deployment"""
    deployment = {
        'apiVersion': 'apps/v1',
        'kind': 'Deployment',
        'metadata': {
            'name': f'{app_name}-deployment',
            'labels': {
                'app': app_name
            }
        },
        'spec': {
            'replicas': replicas,
            'selector': {
                'matchLabels': {
                    'app': app_name
                }
            },
            'template': {
                'metadata': {
                    'labels': {
                        'app': app_name
                    }
                },
                'spec': {
                    'containers': [{
                        'name': app_name,
                        'image': image,

```

```

        'ports': [{
            'containerPort': 8080
        }],
        'env': [{
            'name': 'APP_ENV',
            'value': 'production'
        }],
        'resources': {
            'requests': {
                'memory': '256Mi',
                'cpu': '250m'
            },
            'limits': {
                'memory': '512Mi',
                'cpu': '500m'
            }
        },
        'livenessProbe': {
            'httpGet': {
                'path': '/health',
                'port': 8080
            },
            'initialDelaySeconds': 30,
            'periodSeconds': 10
        },
        'readinessProbe': {
            'httpGet': {
                'path': '/ready',
                'port': 8080
            },
            'initialDelaySeconds': 5,
            'periodSeconds': 5
        }
    }
}
}
}
}
return deployment

```

```

def create_docker_compose(services: Dict[str, Dict[str, Any]]) -> Dict[str, Any]:
    """Utwórz konfigurację Docker Compose"""
    compose = {
        'version': '3.8',
        'services': services,
        'networks': {
            'app-network': {

```

```

        'driver': 'bridge'
    }
},
'volumes': {
    'app-data': {
        'driver': 'local'
    }
}
}
return compose

```

# Przykład użycia

```

if __name__ == "__main__":
    config_manager = ConfigManager()

    # Utwórz konfigurację aplikacji
    app_config = {
        'application': {
            'name': 'DevOps App',
            'version': '1.0.0',
            'environment': 'production'
        },
        'database': {
            'host': 'localhost',
            'port': 5432,
            'name': 'devops_db',
            'ssl': True
        },
        'redis': {
            'host': 'localhost',
            'port': 6379,
            'db': 0
        },
        'logging': {
            'level': 'INFO',
            'format': '%(asctime)s - %(name)s - %(levelname)s - %(message)s'
        }
    }

    # Zapisz konfigurację jako YAML
    config_manager.save_yaml(app_config, '/tmp/app-config.yaml')

    # Zapisz konfigurację jako JSON
    config_manager.save_json(app_config, '/tmp/app-config.json')

    # Utwórz Kubernetes Deployment
    k8s_deployment = create_kubernetes_deployment(
        app_name='my-app',

```

```

        image='my-app:1.0.0',
        replicas=5
    )
    config_manager.save_yaml(k8s_deployment, '/tmp/k8s-deployment.yaml')

# Utwórz Docker Compose
docker_services = {
    'web': {
        'image': 'nginx:alpine',
        'ports': ['80:80'],
        'networks': ['app-network']
    },
    'api': {
        'image': 'my-api:latest',
        'ports': ['8080:8080'],
        'environment': {
            'DATABASE_URL': 'postgresql://user:pass@db:5432/mydb'
        },
        'networks': ['app-network'],
        'depends_on': ['db']
    },
    'db': {
        'image': 'postgres:13',
        'environment': {
            'POSTGRES_DB': 'mydb',
            'POSTGRES_USER': 'user',
            'POSTGRES_PASSWORD': 'pass'
        },
        'volumes': ['app-data:/var/lib/postgresql/data'],
        'networks': ['app-network']
    }
}

docker_compose = create_docker_compose(docker_services)
config_manager.save_yaml(docker_compose, '/tmp/docker-compose.yaml')

print("Pliki konfiguracyjne zostały utworzone:")
print("- /tmp/app-config.yaml")
print("- /tmp/app-config.json")
print("- /tmp/k8s-deployment.yaml")
print("- /tmp/docker-compose.yaml")

```

## Logging i monitoring

Python

```
#!/usr/bin/env python3
"""
Logging i monitoring w Python DevOps
"""

import logging
import time
import psutil
import json
from datetime import datetime
from typing import Dict, Any
from pathlib import Path

class DevOpsLogger:
    """Klasa do zaawansowanego logowania DevOps"""

    def __init__(self, name: str, log_file: str = None, level: str = "INFO"):
        self.logger = logging.getLogger(name)
        self.logger.setLevel(getattr(logging, level.upper()))

        # Usuń istniejące handlersy
        self.logger.handlers.clear()

        # Format logów
        formatter = logging.Formatter(
            '%(asctime)s - %(name)s - %(levelname)s - %(message)s'
        )

        # Console handler
        console_handler = logging.StreamHandler()
        console_handler.setFormatter(formatter)
        self.logger.addHandler(console_handler)

        # File handler (jeśli podano)
        if log_file:
            Path(log_file).parent.mkdir(parents=True, exist_ok=True)
            file_handler = logging.FileHandler(log_file)
            file_handler.setFormatter(formatter)
            self.logger.addHandler(file_handler)

    def info(self, message: str, extra_data: Dict[str, Any] = None):
        """Log info message"""
        if extra_data:
            message = f"{message} | Data: {json.dumps(extra_data)}"
        self.logger.info(message)

    def error(self, message: str, exception: Exception = None):
```

```

        """Log error message"""
        if exception:
            message = f"{message} | Exception: {str(exception)}"
            self.logger.error(message)

    def warning(self, message: str, extra_data: Dict[str, Any] = None):
        """Log warning message"""
        if extra_data:
            message = f"{message} | Data: {json.dumps(extra_data)}"
            self.logger.warning(message)

class SystemMonitor:
    """Klasa do monitorowania systemu"""

    def __init__(self, logger: DevOpsLogger):
        self.logger = logger

    def get_system_metrics(self) -> Dict[str, Any]:
        """Pobierz metryki systemu"""
        try:
            # CPU
            cpu_percent = psutil.cpu_percent(interval=1)
            cpu_count = psutil.cpu_count()

            # Memory
            memory = psutil.virtual_memory()
            memory_percent = memory.percent
            memory_available_gb = memory.available / (1024**3)

            # Disk
            disk = psutil.disk_usage('/')
            disk_percent = (disk.used / disk.total) * 100
            disk_free_gb = disk.free / (1024**3)

            # Network
            network = psutil.net_io_counters()

            # Load average (Linux/macOS)
            try:
                load_avg = psutil.getloadavg()
            except AttributeError:
                load_avg = [0, 0, 0] # Windows doesn't have load average

            metrics = {
                'timestamp': datetime.now().isoformat(),
                'cpu': {
                    'percent': cpu_percent,
                    'count': cpu_count,

```

```

        'load_avg_1m': load_avg[0],
        'load_avg_5m': load_avg[1],
        'load_avg_15m': load_avg[2]
    },
    'memory': {
        'percent': memory_percent,
        'available_gb': round(memory_available_gb, 2),
        'total_gb': round(memory.total / (1024**3), 2)
    },
    'disk': {
        'percent': round(disk_percent, 2),
        'free_gb': round(disk_free_gb, 2),
        'total_gb': round(disk.total / (1024**3), 2)
    },
    'network': {
        'bytes_sent': network.bytes_sent,
        'bytes_recv': network.bytes_recv,
        'packets_sent': network.packets_sent,
        'packets_recv': network.packets_recv
    }
}

return metrics

except Exception as e:
    self.logger.error("Failed to get system metrics", e)
    return {}

def check_thresholds(self, metrics: Dict[str, Any]) -> Dict[str, str]:
    """Sprawdź progi alarmowe"""
    alerts = {}

    # CPU threshold
    if metrics.get('cpu', {}).get('percent', 0) > 80:
        alerts['cpu'] = f"High CPU usage: {metrics['cpu']['percent']}%"

    # Memory threshold
    if metrics.get('memory', {}).get('percent', 0) > 85:
        alerts['memory'] = f"High memory usage: {metrics['memory']
['percent']}%"

    # Disk threshold
    if metrics.get('disk', {}).get('percent', 0) > 90:
        alerts['disk'] = f"High disk usage: {metrics['disk']
['percent']}%"

    # Load average threshold (for systems that support it)
    load_avg_1m = metrics.get('cpu', {}).get('load_avg_1m', 0)

```

```

        cpu_count = metrics.get('cpu', {}).get('count', 1)
        if load_avg_1m > cpu_count * 2:
            alerts['load'] = f"High load average: {load_avg_1m}"

    return alerts

def monitor_continuously(self, interval: int = 60, alert_callback=None):
    """Monitoruj system w sposób ciągły"""
    self.logger.info(f"Starting continuous monitoring (interval:
{interval}s)")

    while True:
        try:
            metrics = self.get_system_metrics()
            if metrics:
                self.logger.info("System metrics collected", metrics)

                # Sprawdź progi alarmowe
                alerts = self.check_thresholds(metrics)
                if alerts:
                    for alert_type, alert_message in alerts.items():
                        self.logger.warning(f"ALERT [{alert_type}]:
{alert_message}")

                        if alert_callback:
                            alert_callback(alert_type, alert_message,
metrics)

                            time.sleep(interval)

        except KeyboardInterrupt:
            self.logger.info("Monitoring stopped by user")
            break
        except Exception as e:
            self.logger.error("Error during monitoring", e)
            time.sleep(interval)

def alert_handler(alert_type: str, message: str, metrics: Dict[str, Any]):
    """Handler dla alertów"""
    print(f"\n🚨 ALERT: {message}")

    # Tutaj można dodać integrację z systemami alertowania:
    # - Slack
    # - PagerDuty
    # - Email
    # - SMS

    # Przykład wysyłania do Slack (wymaga konfiguracji webhook)

```



```

# send_slack_alert(message, metrics)

# Przykład użycia
if __name__ == "__main__":
    # Inicjalizacja loggera
    logger = DevOpsLogger(
        name="DevOps-Monitor",
        log_file="/tmp/devops-monitor.log",
        level="INFO"
    )

    # Inicjalizacja monitora
    monitor = SystemMonitor(logger)

    # Jednorazowe pobranie metryk
    metrics = monitor.get_system_metrics()
    if metrics:
        print("Current system metrics:")
        print(json.dumps(metrics, indent=2))

        # Sprawdź alerty
        alerts = monitor.check_thresholds(metrics)
        if alerts:
            print("\nActive alerts:")
            for alert_type, message in alerts.items():
                print(f"- {alert_type}: {message}")
        else:
            print("\nNo alerts - system is healthy ✅")

    # Uruchom ciągły monitoring (odkomentuj aby uruchomić)
    # monitor.monitor_continuously(interval=30, alert_callback=alert_handler)

```

To kończy podstawy Python dla DevOps. W następnym rozdziale omówimy automatyzację z Python i praktyczne skrypty DevOps.

## 10. Automatyzacja z Python

### Wprowadzenie do automatyzacji DevOps

Automatyzacja jest sercem DevOps, a Python stanowi idealne narzędzie do tworzenia skryptów automatyzujących różnorodne zadania operacyjne. Od prostych skryptów backup po złożone systemy orkiestracji, Python umożliwia automatyzację praktycznie każdego aspektu infrastruktury IT.

Kluczowe obszary automatyzacji w DevOps obejmują:

**Zarządzanie infrastrukturą** - automatyczne provisioning, konfiguracja i zarządzanie serwerami, kontenerami i usługami chmurowymi.

**Deployment i release management** - automatyzacja procesów wdrażania aplikacji, rollback i zarządzania wersjami.

**Monitoring i alerting** - automatyczne zbieranie metryk, wykrywanie anomalii i wysyłanie powiadomień.

**Backup i disaster recovery** - automatyczne tworzenie kopii zapasowych, testowanie procedur odzyskiwania danych.

**Security i compliance** - automatyczne skanowanie podatności, audyty bezpieczeństwa i zapewnianie zgodności z regulacjami.

**Testing i quality assurance** - automatyczne uruchamianie testów, analiza jakości kodu i raportowanie.

## Automatyzacja zarządzania serwerami

### SSH i zdalne wykonywanie poleceń

Python

```
#!/usr/bin/env python3
"""
Automatyzacja zarządzania serwerami przez SSH
"""

import paramiko
import concurrent.futures
import time
from typing import List, Dict, Any, Tuple
import logging

class SSHManager:
    """Klasa do zarządzania połączeniami SSH"""

    def __init__(self, logger=None):
        self.logger = logger or logging.getLogger(__name__)
```

```

self.connections = {}

def connect(self, hostname: str, username: str, password: str = None,
            key_file: str = None, port: int = 22) -> bool:
    """Nawiąż połączenie SSH"""
    try:
        client = paramiko.SSHClient()
        client.set_missing_host_key_policy(paramiko.AutoAddPolicy())

        if key_file:
            client.connect(
                hostname=hostname,
                port=port,
                username=username,
                key_filename=key_file,
                timeout=10
            )
        else:
            client.connect(
                hostname=hostname,
                port=port,
                username=username,
                password=password,
                timeout=10
            )

        self.connections[hostname] = client
        self.logger.info(f"Connected to {hostname}")
        return True

    except Exception as e:
        self.logger.error(f"Failed to connect to {hostname}: {e}")
        return False

def execute_command(self, hostname: str, command: str) -> Dict[str, Any]:
    """Wykonaj polecenie na zdalnym serwerze"""
    if hostname not in self.connections:
        return {
            'success': False,
            'error': f'No connection to {hostname}'
        }

    try:
        client = self.connections[hostname]
        stdin, stdout, stderr = client.exec_command(command)

        # Czekaj na zakończenie polecenia
        exit_status = stdout.channel.recv_exit_status()

```

```

        result = {
            'success': exit_status == 0,
            'stdout': stdout.read().decode('utf-8'),
            'stderr': stderr.read().decode('utf-8'),
            'exit_status': exit_status,
            'hostname': hostname,
            'command': command
        }

        self.logger.info(f"Command executed on {hostname}: {command}")
        return result

    except Exception as e:
        self.logger.error(f"Failed to execute command on {hostname}:
{e}")
        return {
            'success': False,
            'error': str(e),
            'hostname': hostname,
            'command': command
        }

    def upload_file(self, hostname: str, local_path: str, remote_path: str) -
-> bool:
        """Prześlij plik na zdalny serwer"""
        if hostname not in self.connections:
            self.logger.error(f'No connection to {hostname}')
            return False

        try:
            client = self.connections[hostname]
            sftp = client.open_sftp()
            sftp.put(local_path, remote_path)
            sftp.close()

            self.logger.info(f"File uploaded to {hostname}: {local_path} ->
{remote_path}")
            return True

        except Exception as e:
            self.logger.error(f"Failed to upload file to {hostname}: {e}")
            return False

    def download_file(self, hostname: str, remote_path: str, local_path: str)
-> bool:
        """Pobierz plik ze zdalnego serwera"""
        if hostname not in self.connections:

```

```

        self.logger.error(f'No connection to {hostname}')
        return False

    try:
        client = self.connections[hostname]
        sftp = client.open_sftp()
        sftp.get(remote_path, local_path)
        sftp.close()

        self.logger.info(f"File downloaded from {hostname}: {remote_path}
-> {local_path}")
        return True

    except Exception as e:
        self.logger.error(f"Failed to download file from {hostname}:
{e}")
        return False

def close_connection(self, hostname: str):
    """Zamknij połączenie SSH"""
    if hostname in self.connections:
        self.connections[hostname].close()
        del self.connections[hostname]
        self.logger.info(f"Connection to {hostname} closed")

def close_all_connections(self):
    """Zamknij wszystkie połączenia SSH"""
    for hostname in list(self.connections.keys()):
        self.close_connection(hostname)

class ServerManager:
    """Klasa do zarządzania grupą serwerów"""

    def __init__(self, servers: List[Dict[str, str]]):
        self.servers = servers
        self.ssh_manager = SSHManager()
        self.logger = logging.getLogger(__name__)

    def connect_to_all_servers(self) -> Dict[str, bool]:
        """Nawiąż połączenia ze wszystkimi serwerami"""
        results = {}

        for server in self.servers:
            hostname = server['hostname']
            username = server['username']
            password = server.get('password')
            key_file = server.get('key_file')
            port = server.get('port', 22)

```

```

        success = self.ssh_manager.connect(
            hostname=hostname,
            username=username,
            password=password,
            key_file=key_file,
            port=port
        )
        results[hostname] = success

    return results

def execute_on_all_servers(self, command: str, parallel: bool = True) ->
List[Dict[str, Any]]:
    """Wykonaj polecenie na wszystkich serwerach"""
    hostnames = [server['hostname'] for server in self.servers]

    if parallel:
        return self._execute_parallel(hostnames, command)
    else:
        return self._execute_sequential(hostnames, command)

def _execute_parallel(self, hostnames: List[str], command: str) ->
List[Dict[str, Any]]:
    """Wykonaj polecenie równolegle na wszystkich serwerach"""
    results = []

    with concurrent.futures.ThreadPoolExecutor(max_workers=10) as
executor:
        future_to_hostname = {
            executor.submit(self.ssh_manager.execute_command, hostname,
command): hostname
            for hostname in hostnames
        }

        for future in
concurrent.futures.as_completed(future_to_hostname):
            result = future.result()
            results.append(result)

    return results

def _execute_sequential(self, hostnames: List[str], command: str) ->
List[Dict[str, Any]]:
    """Wykonaj polecenie sekwencyjnie na wszystkich serwerach"""
    results = []

    for hostname in hostnames:

```

```

        result = self.ssh_manager.execute_command(hostname, command)
        results.append(result)

    return results

def update_all_servers(self) -> List[Dict[str, Any]]:
    """Zaktualizuj wszystkie serwery"""
    commands = [
        "sudo apt update",
        "sudo apt upgrade -y",
        "sudo apt autoremove -y",
        "sudo apt autoclean"
    ]

    all_results = []

    for command in commands:
        self.logger.info(f"Executing: {command}")
        results = self.execute_on_all_servers(command, parallel=True)
        all_results.extend(results)

        # Sprawdź czy wszystkie polecenia się powiodły
        failed_servers = [r for r in results if not r['success']]
        if failed_servers:
            self.logger.error(f"Command failed on {len(failed_servers)}
servers")
            for result in failed_servers:
                self.logger.error(f"{result['hostname']}:
{result.get('error', result.get('stderr'))}")

    return all_results

def deploy_application(self, app_package: str, service_name: str) ->
List[Dict[str, Any]]:
    """Wdróż aplikację na wszystkich serwerach"""
    deployment_commands = [
        f"sudo systemctl stop {service_name}",
        f"sudo cp {app_package} /opt/",
        f"sudo tar -xzf /opt/{app_package} -C /opt/",
        f"sudo systemctl start {service_name}",
        f"sudo systemctl enable {service_name}",
        f"sleep 5",
        f"sudo systemctl status {service_name}"
    ]

    all_results = []

    for command in deployment_commands:

```

```

        self.logger.info(f"Deployment step: {command}")
        results = self.execute_on_all_servers(command, parallel=False)
        all_results.extend(results)

        # Sprawdź czy polecenie się powiodło na wszystkich serwerach
        failed_servers = [r for r in results if not r['success']]
        if failed_servers:
            self.logger.error(f"Deployment failed on
{len(failed_servers)} servers")
            # Rollback logic można dodać tutaj
            break

    return all_results

def collect_system_info(self) -> Dict[str, Dict[str, Any]]:
    """Zbierz informacje o systemie ze wszystkich serwerów"""
    commands = {
        'hostname': 'hostname',
        'uptime': 'uptime',
        'disk_usage': 'df -h',
        'memory_usage': 'free -h',
        'cpu_info': 'lscpu | grep "Model name"',
        'load_average': 'cat /proc/loadavg',
        'running_services': 'systemctl list-units --type=service --
state=running | head -20'
    }

    server_info = {}

    for info_type, command in commands.items():
        self.logger.info(f"Collecting {info_type}")
        results = self.execute_on_all_servers(command, parallel=True)

        for result in results:
            hostname = result['hostname']
            if hostname not in server_info:
                server_info[hostname] = {}

            if result['success']:
                server_info[hostname][info_type] =
result['stdout'].strip()
            else:
                server_info[hostname][info_type] = f"Error:
{result.get('error', result.get('stderr'))}"

    return server_info

# Przykład użycia

```



```
if __name__ == "__main__":
    # Konfiguracja logowania
    logging.basicConfig(
        level=logging.INFO,
        format='%(asctime)s - %(levelname)s - %(message)s'
    )

    # Definicja serwerów
    servers = [
        {
            'hostname': '192.168.1.10',
            'username': 'ubuntu',
            'key_file': '/home/user/.ssh/id_rsa'
        },
        {
            'hostname': '192.168.1.11',
            'username': 'ubuntu',
            'key_file': '/home/user/.ssh/id_rsa'
        },
        {
            'hostname': '192.168.1.12',
            'username': 'centos',
            'password': 'secure_password'
        }
    ]

    # Inicjalizacja managera
    server_manager = ServerManager(servers)

    # Nawiąż połączenia
    connection_results = server_manager.connect_to_all_servers()
    print("Connection results:", connection_results)

    # Zbierz informacje o systemie
    system_info = server_manager.collect_system_info()

    for hostname, info in system_info.items():
        print(f"\n=== {hostname} ===")
        for key, value in info.items():
            print(f"{key}: {value}")

    # Wykonaj polecenie na wszystkich serwerach
    results = server_manager.execute_on_all_servers("whoami", parallel=True)
    for result in results:
        print(f"{result['hostname']}: {result['stdout'].strip()}")
```

```
# Zamknij połączenia
server_manager.ssh_manager.close_all_connections()
```

## Automatyzacja Docker

Python

```
#!/usr/bin/env python3
"""
Automatyzacja Docker z Python
"""

import docker
import time
import json
from typing import List, Dict, Any, Optional
import logging

class DockerManager:
    """Klasa do zarządzania kontenerami Docker"""

    def __init__(self):
        try:
            self.client = docker.from_env()
            self.logger = logging.getLogger(__name__)
            self.logger.info("Docker client initialized")
        except Exception as e:
            self.logger.error(f"Failed to initialize Docker client: {e}")
            raise

    def build_image(self, dockerfile_path: str, image_name: str,
                    tag: str = "latest", build_args: Dict[str, str] = None) ->
bool:
    """Zbuduj obraz Docker"""
    try:
        self.logger.info(f"Building image {image_name}:{tag}")

        image, build_logs = self.client.images.build(
            path=dockerfile_path,
            tag=f"{image_name}:{tag}",
            buildargs=build_args or {},
            rm=True,
            forcerm=True
        )

        # Wyświetl logi budowania
```

```

        for log in build_logs:
            if 'stream' in log:
                self.logger.info(log['stream'].strip())

        self.logger.info(f"Image {image_name}:{tag} built successfully")
        return True

    except Exception as e:
        self.logger.error(f"Failed to build image {image_name}:{tag}:
{e}")
        return False

    def run_container(self, image_name: str, container_name: str = None,
ports: Dict[str, int] = None, environment: Dict[str,
str] = None,
volumes: Dict[str, Dict[str, str]] = None,
detach: bool = True) -> Optional[str]:
        """Uruchom kontener"""
        try:
            self.logger.info(f"Running container from image {image_name}")

            container = self.client.containers.run(
                image=image_name,
                name=container_name,
                ports=ports or {},
                environment=environment or {},
                volumes=volumes or {},
                detach=detach,
                remove=False
            )

            container_id = container.id
            self.logger.info(f"Container {container_name or container_id}
started")
            return container_id

        except Exception as e:
            self.logger.error(f"Failed to run container: {e}")
            return None

    def stop_container(self, container_name: str, timeout: int = 10) -> bool:
        """Zatrzymaj kontener"""
        try:
            container = self.client.containers.get(container_name)
            container.stop(timeout=timeout)
            self.logger.info(f"Container {container_name} stopped")
            return True

```

```

        except docker.errors.NotFound:
            self.logger.warning(f"Container {container_name} not found")
            return False
        except Exception as e:
            self.logger.error(f"Failed to stop container {container_name}:
{e}")
            return False

    def remove_container(self, container_name: str, force: bool = False) ->
bool:
        """Usuń kontener"""
        try:
            container = self.client.containers.get(container_name)
            container.remove(force=force)
            self.logger.info(f"Container {container_name} removed")
            return True

        except docker.errors.NotFound:
            self.logger.warning(f"Container {container_name} not found")
            return False
        except Exception as e:
            self.logger.error(f"Failed to remove container {container_name}:
{e}")
            return False

    def get_container_logs(self, container_name: str, tail: int = 100) ->
str:
        """Pobierz logi kontenera"""
        try:
            container = self.client.containers.get(container_name)
            logs = container.logs(tail=tail, timestamps=True)
            return logs.decode('utf-8')

        except docker.errors.NotFound:
            self.logger.warning(f"Container {container_name} not found")
            return ""
        except Exception as e:
            self.logger.error(f"Failed to get logs for {container_name}:
{e}")
            return ""

    def get_container_stats(self, container_name: str) -> Dict[str, Any]:
        """Pobierz statystyki kontenera"""
        try:
            container = self.client.containers.get(container_name)
            stats = container.stats(stream=False)

            # Oblicz użycie CPU

```

```

        cpu_delta = stats['cpu_stats']['cpu_usage']['total_usage'] - \
            stats['precpu_stats']['cpu_usage']['total_usage']
        system_delta = stats['cpu_stats']['system_cpu_usage'] - \
            stats['precpu_stats']['system_cpu_usage']
        cpu_percent = (cpu_delta / system_delta) * 100.0

        # Oblicz użycie pamięci
        memory_usage = stats['memory_stats']['usage']
        memory_limit = stats['memory_stats']['limit']
        memory_percent = (memory_usage / memory_limit) * 100.0

        return {
            'cpu_percent': round(cpu_percent, 2),
            'memory_usage_mb': round(memory_usage / (1024 * 1024), 2),
            'memory_limit_mb': round(memory_limit / (1024 * 1024), 2),
            'memory_percent': round(memory_percent, 2),
            'network_rx_bytes': stats['networks']['eth0']['rx_bytes'],
            'network_tx_bytes': stats['networks']['eth0']['tx_bytes']
        }

    except Exception as e:
        self.logger.error(f"Failed to get stats for {container_name}: {e}")
        return {}

    def list_containers(self, all_containers: bool = True) -> List[Dict[str, Any]]:
        """Lista kontenerów"""
        try:
            containers = self.client.containers.list(all=all_containers)
            container_list = []

            for container in containers:
                container_info = {
                    'id': container.id[:12],
                    'name': container.name,
                    'image': container.image.tags[0] if container.image.tags
else container.image.id,
                    'status': container.status,
                    'created': container.attrs['Created'],
                    'ports': container.ports
                }
                container_list.append(container_info)

            return container_list

        except Exception as e:
            self.logger.error(f"Failed to list containers: {e}")

```

```

        return []

def cleanup_unused_resources(self) -> Dict[str, Any]:
    """Wyczyść nieużywane zasoby Docker"""
    try:
        self.logger.info("Cleaning up unused Docker resources")

        # Usuń nieużywane kontenery
        containers_removed = self.client.containers.prune()

        # Usuń nieużywane obrazy
        images_removed = self.client.images.prune(filters={'dangling':
False})

        # Usuń nieużywane wolumeny
        volumes_removed = self.client.volumes.prune()

        # Usuń nieużywane sieci
        networks_removed = self.client.networks.prune()

        cleanup_result = {
            'containers_removed':
len(containers_removed.get('ContainersDeleted', [])),
            'images_removed': len(images_removed.get('ImagesDeleted',
[])),
            'volumes_removed': len(volumes_removed.get('VolumesDeleted',
[])),
            'networks_removed':
len(networks_removed.get('NetworksDeleted', [])),
            'space_reclaimed_mb': round(
                images_removed.get('SpaceReclaimed', 0) / (1024 * 1024),
2
            )
        }

        self.logger.info(f"Cleanup completed: {cleanup_result}")
        return cleanup_result

    except Exception as e:
        self.logger.error(f"Failed to cleanup Docker resources: {e}")
        return {}

class DockerComposeManager:
    """Klasa do zarządzania Docker Compose"""

    def __init__(self, compose_file: str = "docker-compose.yml"):
        self.compose_file = compose_file
        self.logger = logging.getLogger(__name__)

```

```

def up(self, detach: bool = True, build: bool = False) -> bool:
    """Uruchom usługi Docker Compose"""
    try:
        import subprocess

        command = ["docker-compose", "-f", self.compose_file, "up"]
        if detach:
            command.append("-d")
        if build:
            command.append("--build")

        result = subprocess.run(command, capture_output=True, text=True)

        if result.returncode == 0:
            self.logger.info("Docker Compose services started
successfully")
            return True
        else:
            self.logger.error(f"Failed to start services:
{result.stderr}")
            return False

    except Exception as e:
        self.logger.error(f"Failed to run docker-compose up: {e}")
        return False

def down(self, remove_volumes: bool = False) -> bool:
    """Zatrzymaj usługi Docker Compose"""
    try:
        import subprocess

        command = ["docker-compose", "-f", self.compose_file, "down"]
        if remove_volumes:
            command.append("-v")

        result = subprocess.run(command, capture_output=True, text=True)

        if result.returncode == 0:
            self.logger.info("Docker Compose services stopped
successfully")
            return True
        else:
            self.logger.error(f"Failed to stop services:
{result.stderr}")
            return False

    except Exception as e:

```

```

        self.logger.error(f"Failed to run docker-compose down: {e}")
        return False

def logs(self, service_name: str = None, tail: int = 100) -> str:
    """Pobierz logi usług"""
    try:
        import subprocess

        command = ["docker-compose", "-f", self.compose_file, "logs", "--tail", str(tail)]
        if service_name:
            command.append(service_name)

        result = subprocess.run(command, capture_output=True, text=True)
        return result.stdout

    except Exception as e:
        self.logger.error(f"Failed to get logs: {e}")
        return ""

# Przykład użycia
if __name__ == "__main__":
    # Konfiguracja logowania
    logging.basicConfig(
        level=logging.INFO,
        format='%(asctime)s - %(levelname)s - %(message)s'
    )

    # Inicjalizacja Docker managera
    docker_manager = DockerManager()

    # Lista kontenerów
    containers = docker_manager.list_containers()
    print("Running containers:")
    for container in containers:
        print(f"- {container['name']} ({container['image']}) - {container['status']}")

    # Przykład uruchomienia kontenera nginx
    container_id = docker_manager.run_container(
        image_name="nginx:alpine",
        container_name="test-nginx",
        ports={'80/tcp': 8080},
        detach=True
    )

    if container_id:
        print(f"Container started with ID: {container_id}")

```



```

# Poczekaj chwilę
time.sleep(5)

# Pobierz statystyki
stats = docker_manager.get_container_stats("test-nginx")
if stats:
    print(f"Container stats: {stats}")

# Pobierz logi
logs = docker_manager.get_container_logs("test-nginx")
print(f"Container logs:\n{logs}")

# Zatrzymaj i usuń kontener
docker_manager.stop_container("test-nginx")
docker_manager.remove_container("test-nginx")

# Wyczyść nieużywane zasoby
cleanup_result = docker_manager.cleanup_unused_resources()
print(f"Cleanup result: {cleanup_result}")

```

## Automatyzacja backup

Python

```

#!/usr/bin/env python3
"""
Automatyzacja backup i disaster recovery
"""

import os
import shutil
import tarfile
import gzip
import boto3
import mysql.connector
import psycopg2
import subprocess
import schedule
import time
from datetime import datetime, timedelta
from typing import List, Dict, Any, Optional
import logging

class BackupManager:
    """Klasa do zarządzania backup"""

```

```

def __init__(self, backup_config: Dict[str, Any]):
    self.config = backup_config
    self.logger = logging.getLogger(__name__)

    # Inicjalizacja AWS S3 (jeśli skonfigurowane)
    if 'aws' in backup_config:
        self.s3_client = boto3.client(
            's3',
            aws_access_key_id=backup_config['aws']['access_key'],
            aws_secret_access_key=backup_config['aws']['secret_key'],
            region_name=backup_config['aws'].get('region', 'us-east-1')
        )
    else:
        self.s3_client = None

    def create_directory_backup(self, source_dir: str, backup_name: str) ->
Optional[str]:
        """Utwórz backup katalogu"""
        try:
            timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
            backup_filename = f"{backup_name}_{timestamp}.tar.gz"
            backup_path = os.path.join(self.config['local_backup_dir'],
backup_filename)

            # Utwórz katalog backup jeśli nie istnieje
            os.makedirs(self.config['local_backup_dir'], exist_ok=True)

            self.logger.info(f"Creating backup of {source_dir}")

            with tarfile.open(backup_path, 'w:gz') as tar:
                tar.add(source_dir, arcname=os.path.basename(source_dir))

            # Sprawdź rozmiar backup
            backup_size = os.path.getsize(backup_path)
            self.logger.info(f"Backup created: {backup_path} ({backup_size /
(1024*1024):.2f} MB)")

            return backup_path

        except Exception as e:
            self.logger.error(f"Failed to create directory backup: {e}")
            return None

    def create_mysql_backup(self, db_config: Dict[str, str], backup_name:
str) -> Optional[str]:
        """Utwórz backup bazy danych MySQL"""
        try:

```

```

        timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
        backup_filename = f"{backup_name}_mysql_{timestamp}.sql.gz"
        backup_path = os.path.join(self.config['local_backup_dir'],
backup_filename)

        # Utwórz katalog backup jeśli nie istnieje
        os.makedirs(self.config['local_backup_dir'], exist_ok=True)

        self.logger.info(f"Creating MySQL backup for database
{db_config['database']}")

        # Polecenie mysqldump
        dump_command = [
            'mysqldump',
            f"--host={db_config['host']}",
            f"--port={db_config.get('port', 3306)}",
            f"--user={db_config['username']}",
            f"--password={db_config['password']}",
            '--single-transaction',
            '--routines',
            '--triggers',
            db_config['database']
        ]

        # Wykonaj mysqldump i skompresuj
        with gzip.open(backup_path, 'wt') as f:
            result = subprocess.run(
                dump_command,
                stdout=f,
                stderr=subprocess.PIPE,
                text=True
            )

            if result.returncode == 0:
                backup_size = os.path.getsize(backup_path)
                self.logger.info(f"MySQL backup created: {backup_path}
({backup_size / (1024*1024):.2f} MB)")
                return backup_path
            else:
                self.logger.error(f"MySQL backup failed: {result.stderr}")
                return None

        except Exception as e:
            self.logger.error(f"Failed to create MySQL backup: {e}")
            return None

    def create_postgresql_backup(self, db_config: Dict[str, str],
backup_name: str) -> Optional[str]:

```

```

"""Utwórz backup bazy danych PostgreSQL"""
try:
    timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
    backup_filename = f"{backup_name}_postgresql_{timestamp}.sql.gz"
    backup_path = os.path.join(self.config['local_backup_dir'],
backup_filename)

    # Utwórz katalog backup jeśli nie istnieje
    os.makedirs(self.config['local_backup_dir'], exist_ok=True)

    self.logger.info(f"Creating PostgreSQL backup for database
{db_config['database']}")

    # Ustaw zmienne środowiskowe dla pg_dump
    env = os.environ.copy()
    env['PGPASSWORD'] = db_config['password']

    # Polecenie pg_dump
    dump_command = [
        'pg_dump',
        f"--host={db_config['host']}",
        f"--port={db_config.get('port', 5432)}",
        f"--username={db_config['username']}",
        '--format=custom',
        '--no-password',
        '--verbose',
        db_config['database']
    ]

    # Wykonaj pg_dump i skompresuj
    with gzip.open(backup_path, 'wb') as f:
        result = subprocess.run(
            dump_command,
            stdout=f,
            stderr=subprocess.PIPE,
            env=env
        )

        if result.returncode == 0:
            backup_size = os.path.getsize(backup_path)
            self.logger.info(f"PostgreSQL backup created: {backup_path}
({backup_size / (1024*1024):.2f} MB)")
            return backup_path
        else:
            self.logger.error(f"PostgreSQL backup failed:
{result.stderr}")
            return None

```

```

except Exception as e:
    self.logger.error(f"Failed to create PostgreSQL backup: {e}")
    return None

def upload_to_s3(self, local_file: str, s3_key: str) -> bool:
    """Prześlij backup do S3"""
    if not self.s3_client:
        self.logger.warning("S3 client not configured")
        return False

    try:
        bucket_name = self.config['aws']['bucket']

        self.logger.info(f"Uploading {local_file} to S3:
s3://{bucket_name}/{s3_key}")

        self.s3_client.upload_file(
            local_file,
            bucket_name,
            s3_key,
            ExtraArgs={
                'StorageClass': 'STANDARD_IA', # Cheaper storage for
backups
                'ServerSideEncryption': 'AES256'
            }
        )

        self.logger.info(f"Upload to S3 completed: {s3_key}")
        return True

    except Exception as e:
        self.logger.error(f"Failed to upload to S3: {e}")
        return False

def cleanup_old_backups(self, retention_days: int = 30):
    """Usuń stare backup lokalne"""
    try:
        cutoff_date = datetime.now() - timedelta(days=retention_days)
        backup_dir = self.config['local_backup_dir']

        if not os.path.exists(backup_dir):
            return

        removed_count = 0
        total_size_removed = 0

        for filename in os.listdir(backup_dir):
            file_path = os.path.join(backup_dir, filename)

```

```

        if os.path.isfile(file_path):
            file_mtime =
datetime.fromtimestamp(os.path.getmtime(file_path))

            if file_mtime < cutoff_date:
                file_size = os.path.getsize(file_path)
                os.remove(file_path)
                removed_count += 1
                total_size_removed += file_size
                self.logger.info(f"Removed old backup: {filename}")

    if removed_count > 0:
        self.logger.info(f"Cleanup completed: {removed_count} files
removed, "
                        f"{total_size_removed / (1024*1024):.2f} MB
freed")
    else:
        self.logger.info("No old backups to remove")

except Exception as e:
    self.logger.error(f"Failed to cleanup old backups: {e}")

def cleanup_old_s3_backups(self, retention_days: int = 90):
    """Usuń stare backup z S3"""
    if not self.s3_client:
        return

    try:
        bucket_name = self.config['aws']['bucket']
        cutoff_date = datetime.now() - timedelta(days=retention_days)

        response = self.s3_client.list_objects_v2(Bucket=bucket_name)

        if 'Contents' not in response:
            return

        removed_count = 0

        for obj in response['Contents']:
            if obj['LastModified'].replace(tzinfo=None) < cutoff_date:
                self.s3_client.delete_object(Bucket=bucket_name,
Key=obj['Key'])
                removed_count += 1
                self.logger.info(f"Removed old S3 backup: {obj['Key']}")

        if removed_count > 0:
            self.logger.info(f"S3 cleanup completed: {removed_count}

```

```

files removed")
    else:
        self.logger.info("No old S3 backups to remove")

except Exception as e:
    self.logger.error(f"Failed to cleanup old S3 backups: {e}")

def run_full_backup(self):
    """Uruchom pełny backup"""
    self.logger.info("Starting full backup process")

    backup_results = []

    # Backup katalogów
    for dir_config in self.config.get('directories', []):
        backup_path = self.create_directory_backup(
            dir_config['path'],
            dir_config['name']
        )

        if backup_path:
            backup_results.append({
                'type': 'directory',
                'name': dir_config['name'],
                'local_path': backup_path,
                'success': True
            })

            # Upload do S3 jeśli skonfigurowane
            if self.s3_client:
                s3_key =
f"backups/directories/{os.path.basename(backup_path)}"
                s3_success = self.upload_to_s3(backup_path, s3_key)
                backup_results[-1]['s3_upload'] = s3_success

    # Backup baz danych MySQL
    for db_config in self.config.get('mysql_databases', []):
        backup_path = self.create_mysql_backup(db_config,
db_config['name'])

        if backup_path:
            backup_results.append({
                'type': 'mysql',
                'name': db_config['name'],
                'local_path': backup_path,
                'success': True
            })

```

```

        # Upload do S3 jeśli skonfigurowane
        if self.s3_client:
            s3_key = f"backups/mysql/{os.path.basename(backup_path)}"
            s3_success = self.upload_to_s3(backup_path, s3_key)
            backup_results[-1]['s3_upload'] = s3_success

    # Backup baz danych PostgreSQL
    for db_config in self.config.get('postgresql_databases', []):
        backup_path = self.create_postgresql_backup(db_config,
db_config['name'])

        if backup_path:
            backup_results.append({
                'type': 'postgresql',
                'name': db_config['name'],
                'local_path': backup_path,
                'success': True
            })

    # Upload do S3 jeśli skonfigurowane
    if self.s3_client:
        s3_key =
f"backups/postgresql/{os.path.basename(backup_path)}"
        s3_success = self.upload_to_s3(backup_path, s3_key)
        backup_results[-1]['s3_upload'] = s3_success

    # Cleanup starych backup
    self.cleanup_old_backups(self.config.get('local_retention_days', 7))
    if self.s3_client:
        self.cleanup_old_s3_backups(self.config.get('s3_retention_days',
90))

    # Raport z backup
    successful_backups = [r for r in backup_results if r['success']]
    self.logger.info(f"Backup completed:
{len(successful_backups)}/{len(backup_results)} successful")

    return backup_results

# Przykład konfiguracji i użycia
if __name__ == "__main__":
    # Konfiguracja logowania
    logging.basicConfig(
        level=logging.INFO,
        format='%(asctime)s - %(levelname)s - %(message)s',
        handlers=[
            logging.FileHandler('/var/log/backup.log'),
            logging.StreamHandler()

```



```

    ]
)

# Konfiguracja backup
backup_config = {
    'local_backup_dir': '/backup',
    'local_retention_days': 7,
    's3_retention_days': 90,
    'aws': {
        'access_key': 'YOUR_ACCESS_KEY',
        'secret_key': 'YOUR_SECRET_KEY',
        'bucket': 'your-backup-bucket',
        'region': 'us-east-1'
    },
    'directories': [
        {
            'name': 'web_content',
            'path': '/var/www/html'
        },
        {
            'name': 'app_config',
            'path': '/etc/myapp'
        }
    ],
    'mysql_databases': [
        {
            'name': 'production_db',
            'host': 'localhost',
            'username': 'backup_user',
            'password': 'backup_password',
            'database': 'production'
        }
    ],
    'postgresql_databases': [
        {
            'name': 'analytics_db',
            'host': 'localhost',
            'username': 'postgres',
            'password': 'postgres_password',
            'database': 'analytics'
        }
    ]
}

# Inicjalizacja backup managera
backup_manager = BackupManager(backup_config)

# Uruchom jednorazowy backup

```

```
results = backup_manager.run_full_backup()

# Zaplanuj automatyczne backup
schedule.every().day.at("02:00").do(backup_manager.run_full_backup)
schedule.every().sunday.at("01:00").do(
    lambda: backup_manager.cleanup_old_s3_backups(90)
)

# Uruchom scheduler (odkomentuj aby uruchomić)
# while True:
#     schedule.run_pending()
#     time.sleep(60)
```

To kończy rozdział o automatyzacji z Python. W następnym rozdziale omówimy testowanie aplikacji Python w kontekście DevOps.

## 11. Testowanie aplikacji Python

### Wprowadzenie do testowania w DevOps

Testowanie jest fundamentalnym elementem pipeline'u DevOps, zapewniającym wysoką jakość oprogramowania przy jednoczesnym utrzymaniu szybkiego tempa dostarczania. W kontekście DevOps testowanie nie ogranicza się tylko do weryfikacji funkcjonalności aplikacji, ale obejmuje także testy infrastruktury, bezpieczeństwa, wydajności i zgodności.

Python oferuje bogaty ekosystem narzędzi do testowania, które idealnie integrują się z praktykami DevOps:

**Automatyzacja testów** - wszystkie testy powinny być zautomatyzowane i uruchamiane w ramach pipeline'u CI/CD.

**Test-driven development (TDD)** - pisanie testów przed implementacją funkcjonalności pomaga w projektowaniu lepszej architektury.

**Continuous testing** - testy są uruchamiane przy każdej zmianie kodu, zapewniając szybkie feedback.

**Shift-left testing** - przeniesienie testowania wcześniej w cyklu rozwoju, co pozwala na wcześniejsze wykrycie problemów.

**Test pyramid** - strategia testowania obejmująca różne poziomy testów: jednostkowe, integracyjne i end-to-end.

## Konfiguracja środowiska testowego

### Instalacja narzędzi testowych

Bash

```
# Aktywuj środowisko wirtualne
source venv/bin/activate

# Podstawowe narzędzia testowe
pip install pytest                # Framework testowy
pip install pytest-cov            # Coverage reporting
pip install pytest-xdist          # Parallel test execution
pip install pytest-mock           # Mocking utilities
pip install pytest-html           # HTML test reports
pip install pytest-bdd            # Behavior-driven development
pip install pytest-benchmark      # Performance testing

# Narzędzia do jakości kodu
pip install black                 # Code formatting
pip install flake8               # Linting
pip install isort                # Import sorting
pip install mypy                 # Type checking
pip install bandit               # Security testing
pip install safety               # Dependency vulnerability scanning

# Narzędzia do testowania API
pip install requests             # HTTP client
pip install responses            # HTTP mocking
pip install httpx                # Async HTTP client

# Narzędzia do testowania baz danych
pip install pytest-postgresql    # PostgreSQL testing
pip install pytest-mysql         # MySQL testing
pip install fakeredis            # Redis mocking

# Zapisz zależności
pip freeze > requirements-test.txt
```

### Struktura projektu testowego

## Plain Text

```
project/
├── src/
│   ├── __init__.py
│   ├── app.py
│   ├── models/
│   │   ├── __init__.py
│   │   └── user.py
│   ├── services/
│   │   ├── __init__.py
│   │   └── user_service.py
│   └── utils/
│       ├── __init__.py
│       └── helpers.py
├── tests/
│   ├── __init__.py
│   ├── conftest.py
│   ├── unit/
│   │   ├── __init__.py
│   │   ├── test_models.py
│   │   ├── test_services.py
│   │   └── test_utils.py
│   ├── integration/
│   │   ├── __init__.py
│   │   ├── test_api.py
│   │   └── test_database.py
│   ├── e2e/
│   │   ├── __init__.py
│   │   └── test_user_journey.py
│   └── fixtures/
│       ├── __init__.py
│       └── sample_data.py
├── pytest.ini
├── requirements.txt
├── requirements-test.txt
└── tox.ini
```

## Konfiguracja pytest

### Plain Text

```
# pytest.ini
[tool:pytest]
testpaths = tests
python_files = test_*.py
```

```
python_classes = Test*
python_functions = test_*
addopts =
    --strict-markers
    --strict-config
    --verbose
    --tb=short
    --cov=src
    --cov-report=term-missing
    --cov-report=html:htmlcov
    --cov-report=xml:coverage.xml
    --cov-fail-under=80
    --junitxml=test-results.xml
markers =
    unit: Unit tests
    integration: Integration tests
    e2e: End-to-end tests
    slow: Slow running tests
    api: API tests
    database: Database tests
    security: Security tests
filterwarnings =
    ignore::UserWarning
    ignore::DeprecationWarning
```

## Testy jednostkowe

### Podstawowe testy jednostkowe

Python

```
# src/models/user.py
from dataclasses import dataclass
from typing import Optional
import re

@dataclass
class User:
    username: str
    email: str
    age: Optional[int] = None
    is_active: bool = True

    def __post_init__(self):
        if not self.is_valid_email(self.email):
```

```

        raise ValueError("Invalid email format")
    if not self.is_valid_username(self.username):
        raise ValueError("Invalid username format")
    if self.age is not None and (self.age < 0 or self.age > 150):
        raise ValueError("Invalid age")

    @staticmethod
    def is_valid_email(email: str) -> bool:
        pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
        return re.match(pattern, email) is not None

    @staticmethod
    def is_valid_username(username: str) -> bool:
        return len(username) >= 3 and username.isalnum()

    def deactivate(self):
        self.is_active = False

    def activate(self):
        self.is_active = True

    def update_email(self, new_email: str):
        if not self.is_valid_email(new_email):
            raise ValueError("Invalid email format")
        self.email = new_email

```

## Python

```

# tests/unit/test_models.py
import pytest
from src.models.user import User

class TestUser:
    """Test suite for User model"""

    def test_user_creation_valid_data(self):
        """Test creating user with valid data"""
        user = User(
            username="testuser",
            email="test@example.com",
            age=25
        )

        assert user.username == "testuser"
        assert user.email == "test@example.com"
        assert user.age == 25
        assert user.is_active is True

```

```

def test_user_creation_minimal_data(self):
    """Test creating user with minimal required data"""
    user = User(
        username="testuser",
        email="test@example.com"
    )

    assert user.username == "testuser"
    assert user.email == "test@example.com"
    assert user.age is None
    assert user.is_active is True

@pytest.mark.parametrize("invalid_email", [
    "invalid-email",
    "@example.com",
    "test@",
    "test.example.com",
    "",
    "test@.com"
])
def test_user_creation_invalid_email(self, invalid_email):
    """Test user creation with invalid email formats"""
    with pytest.raises(ValueError, match="Invalid email format"):
        User(username="testuser", email=invalid_email)

@pytest.mark.parametrize("invalid_username", [
    "ab", # too short
    "", # empty
    "test user", # contains space
    "test@user", # contains special character
])
def test_user_creation_invalid_username(self, invalid_username):
    """Test user creation with invalid username formats"""
    with pytest.raises(ValueError, match="Invalid username format"):
        User(username=invalid_username, email="test@example.com")

@pytest.mark.parametrize("invalid_age", [-1, 151, 200])
def test_user_creation_invalid_age(self, invalid_age):
    """Test user creation with invalid age values"""
    with pytest.raises(ValueError, match="Invalid age"):
        User(username="testuser", email="test@example.com",
age=invalid_age)

def test_user_deactivate(self):
    """Test user deactivation"""
    user = User(username="testuser", email="test@example.com")
    assert user.is_active is True

```

```

        user.deactivate()
        assert user.is_active is False

def test_user_activate(self):
    """Test user activation"""
    user = User(username="testuser", email="test@example.com")
    user.deactivate()
    assert user.is_active is False

    user.activate()
    assert user.is_active is True

def test_update_email_valid(self):
    """Test updating email with valid format"""
    user = User(username="testuser", email="test@example.com")
    new_email = "newemail@example.com"

    user.update_email(new_email)
    assert user.email == new_email

def test_update_email_invalid(self):
    """Test updating email with invalid format"""
    user = User(username="testuser", email="test@example.com")

    with pytest.raises(ValueError, match="Invalid email format"):
        user.update_email("invalid-email")

@pytest.mark.parametrize("email,expected", [
    ("test@example.com", True),
    ("user.name@domain.co.uk", True),
    ("invalid-email", False),
    ("@example.com", False),
    ("test@", False),
])
def test_is_valid_email(self, email, expected):
    """Test email validation method"""
    assert User.is_valid_email(email) == expected

@pytest.mark.parametrize("username,expected", [
    ("testuser", True),
    ("user123", True),
    ("ab", False), # too short
    ("", False), # empty
    ("test user", False), # contains space
])
def test_is_valid_username(self, username, expected):

```



```
"""Test username validation method"""
assert User.is_valid_username(username) == expected
```

## Testy z mockami

Python

```
# src/services/user_service.py
from typing import List, Optional
import requests
from src.models.user import User

class UserService:
    """Service for managing users"""

    def __init__(self, api_base_url: str, api_key: str):
        self.api_base_url = api_base_url.rstrip('/')
        self.api_key = api_key
        self.session = requests.Session()
        self.session.headers.update({
            'Authorization': f'Bearer {api_key}',
            'Content-Type': 'application/json'
        })

    def create_user(self, user: User) -> dict:
        """Create user via API"""
        user_data = {
            'username': user.username,
            'email': user.email,
            'age': user.age,
            'is_active': user.is_active
        }

        response = self.session.post(
            f'{self.api_base_url}/users',
            json=user_data,
            timeout=30
        )
        response.raise_for_status()
        return response.json()

    def get_user(self, user_id: int) -> Optional[User]:
        """Get user by ID"""
        response = self.session.get(
            f'{self.api_base_url}/users/{user_id}',
            timeout=30
```

```

    )

    if response.status_code == 404:
        return None

    response.raise_for_status()
    data = response.json()

    return User(
        username=data['username'],
        email=data['email'],
        age=data.get('age'),
        is_active=data.get('is_active', True)
    )

def list_users(self, active_only: bool = True) -> List[User]:
    """List all users"""
    params = {'active_only': active_only} if active_only else {}

    response = self.session.get(
        f'{self.api_base_url}/users',
        params=params,
        timeout=30
    )
    response.raise_for_status()

    users = []
    for user_data in response.json():
        user = User(
            username=user_data['username'],
            email=user_data['email'],
            age=user_data.get('age'),
            is_active=user_data.get('is_active', True)
        )
        users.append(user)

    return users

def delete_user(self, user_id: int) -> bool:
    """Delete user by ID"""
    response = self.session.delete(
        f'{self.api_base_url}/users/{user_id}',
        timeout=30
    )

    if response.status_code == 404:
        return False

```

```
response.raise_for_status()
return True
```

## Python

```
# tests/unit/test_services.py
import pytest
import requests
from unittest.mock import Mock, patch
from src.services.user_service import UserService
from src.models.user import User

class TestUserService:
    """Test suite for UserService"""

    @pytest.fixture
    def user_service(self):
        """Create UserService instance for testing"""
        return UserService(
            api_base_url="https://api.example.com",
            api_key="test-api-key"
        )

    @pytest.fixture
    def sample_user(self):
        """Create sample user for testing"""
        return User(
            username="testuser",
            email="test@example.com",
            age=25
        )

    @patch('src.services.user_service.requests.Session.post')
    def test_create_user_success(self, mock_post, user_service, sample_user):
        """Test successful user creation"""
        # Mock response
        mock_response = Mock()
        mock_response.json.return_value = {
            'id': 123,
            'username': 'testuser',
            'email': 'test@example.com',
            'age': 25,
            'is_active': True
        }
        mock_response.raise_for_status.return_value = None
        mock_post.return_value = mock_response
```

```

# Execute
result = user_service.create_user(sample_user)

# Verify
assert result['id'] == 123
assert result['username'] == 'testuser'

# Verify API call
mock_post.assert_called_once_with(
    'https://api.example.com/users',
    json={
        'username': 'testuser',
        'email': 'test@example.com',
        'age': 25,
        'is_active': True
    },
    timeout=30
)

@patch('src.services.user_service.requests.Session.post')
def test_create_user_api_error(self, mock_post, user_service,
sample_user):
    """Test user creation with API error"""
    # Mock error response
    mock_response = Mock()
    mock_response.raise_for_status.side_effect = requests.HTTPError("API
Error")
    mock_post.return_value = mock_response

    # Execute and verify exception
    with pytest.raises(requests.HTTPError):
        user_service.create_user(sample_user)

@patch('src.services.user_service.requests.Session.get')
def test_get_user_success(self, mock_get, user_service):
    """Test successful user retrieval"""
    # Mock response
    mock_response = Mock()
    mock_response.status_code = 200
    mock_response.json.return_value = {
        'username': 'testuser',
        'email': 'test@example.com',
        'age': 25,
        'is_active': True
    }
    mock_response.raise_for_status.return_value = None
    mock_get.return_value = mock_response

```

```

# Execute
user = user_service.get_user(123)

# Verify
assert user is not None
assert user.username == 'testuser'
assert user.email == 'test@example.com'
assert user.age == 25
assert user.is_active is True

# Verify API call
mock_get.assert_called_once_with(
    'https://api.example.com/users/123',
    timeout=30
)

@patch('src.services.user_service.requests.Session.get')
def test_get_user_not_found(self, mock_get, user_service):
    """Test user retrieval when user not found"""
    # Mock 404 response
    mock_response = Mock()
    mock_response.status_code = 404
    mock_get.return_value = mock_response

    # Execute
    user = user_service.get_user(999)

    # Verify
    assert user is None

@patch('src.services.user_service.requests.Session.get')
def test_list_users_success(self, mock_get, user_service):
    """Test successful user listing"""
    # Mock response
    mock_response = Mock()
    mock_response.json.return_value = [
        {
            'username': 'user1',
            'email': 'user1@example.com',
            'age': 25,
            'is_active': True
        },
        {
            'username': 'user2',
            'email': 'user2@example.com',
            'age': 30,
            'is_active': False
        }
    ]

```

```

    ]
    mock_response.raise_for_status.return_value = None
    mock_get.return_value = mock_response

    # Execute
    users = user_service.list_users(active_only=False)

    # Verify
    assert len(users) == 2
    assert users[0].username == 'user1'
    assert users[1].username == 'user2'

    # Verify API call
    mock_get.assert_called_once_with(
        'https://api.example.com/users',
        params={},
        timeout=30
    )

@patch('src.services.user_service.requests.Session.delete')
def test_delete_user_success(self, mock_delete, user_service):
    """Test successful user deletion"""
    # Mock response
    mock_response = Mock()
    mock_response.status_code = 200
    mock_response.raise_for_status.return_value = None
    mock_delete.return_value = mock_response

    # Execute
    result = user_service.delete_user(123)

    # Verify
    assert result is True

    # Verify API call
    mock_delete.assert_called_once_with(
        'https://api.example.com/users/123',
        timeout=30
    )

@patch('src.services.user_service.requests.Session.delete')
def test_delete_user_not_found(self, mock_delete, user_service):
    """Test user deletion when user not found"""
    # Mock 404 response
    mock_response = Mock()
    mock_response.status_code = 404
    mock_delete.return_value = mock_response

```

```
# Execute
result = user_service.delete_user(999)

# Verify
assert result is False
```

## Testy integracyjne

### Testy API

Python

```
# tests/integration/test_api.py
import pytest
import requests
from fastapi.testclient import TestClient
from src.app import app # Assuming FastAPI app

class TestUserAPI:
    """Integration tests for User API"""

    @pytest.fixture
    def client(self):
        """Create test client"""
        return TestClient(app)

    @pytest.fixture
    def sample_user_data(self):
        """Sample user data for testing"""
        return {
            'username': 'testuser',
            'email': 'test@example.com',
            'age': 25
        }

    def test_create_user_endpoint(self, client, sample_user_data):
        """Test user creation endpoint"""
        response = client.post('/users', json=sample_user_data)

        assert response.status_code == 201
        data = response.json()
        assert data['username'] == sample_user_data['username']
        assert data['email'] == sample_user_data['email']
        assert 'id' in data
```

```

def test_create_user_invalid_data(self, client):
    """Test user creation with invalid data"""
    invalid_data = {
        'username': 'ab', # too short
        'email': 'invalid-email',
        'age': -1
    }

    response = client.post('/users', json=invalid_data)
    assert response.status_code == 422

def test_get_user_endpoint(self, client, sample_user_data):
    """Test user retrieval endpoint"""
    # First create a user
    create_response = client.post('/users', json=sample_user_data)
    user_id = create_response.json()['id']

    # Then retrieve it
    response = client.get(f'/users/{user_id}')

    assert response.status_code == 200
    data = response.json()
    assert data['username'] == sample_user_data['username']
    assert data['email'] == sample_user_data['email']

def test_get_user_not_found(self, client):
    """Test user retrieval for non-existent user"""
    response = client.get('/users/9999999')
    assert response.status_code == 404

def test_list_users_endpoint(self, client, sample_user_data):
    """Test user listing endpoint"""
    # Create a few users
    for i in range(3):
        user_data = sample_user_data.copy()
        user_data['username'] = f'testuser{i}'
        user_data['email'] = f'test{i}@example.com'
        client.post('/users', json=user_data)

    # List users
    response = client.get('/users')

    assert response.status_code == 200
    data = response.json()
    assert len(data) >= 3

def test_update_user_endpoint(self, client, sample_user_data):
    """Test user update endpoint"""

```



```

# Create user
create_response = client.post('/users', json=sample_user_data)
user_id = create_response.json()['id']

# Update user
update_data = {'email': 'updated@example.com'}
response = client.patch(f'/users/{user_id}', json=update_data)

assert response.status_code == 200
data = response.json()
assert data['email'] == 'updated@example.com'

def test_delete_user_endpoint(self, client, sample_user_data):
    """Test user deletion endpoint"""
    # Create user
    create_response = client.post('/users', json=sample_user_data)
    user_id = create_response.json()['id']

    # Delete user
    response = client.delete(f'/users/{user_id}')
    assert response.status_code == 204

    # Verify user is deleted
    get_response = client.get(f'/users/{user_id}')
    assert get_response.status_code == 404

def test_user_lifecycle(self, client, sample_user_data):
    """Test complete user lifecycle"""
    # Create
    create_response = client.post('/users', json=sample_user_data)
    assert create_response.status_code == 201
    user_id = create_response.json()['id']

    # Read
    get_response = client.get(f'/users/{user_id}')
    assert get_response.status_code == 200

    # Update
    update_data = {'age': 30}
    update_response = client.patch(f'/users/{user_id}', json=update_data)
    assert update_response.status_code == 200
    assert update_response.json()['age'] == 30

    # Delete
    delete_response = client.delete(f'/users/{user_id}')
    assert delete_response.status_code == 204

    # Verify deletion

```

```
final_get_response = client.get(f'/users/{user_id}')
assert final_get_response.status_code == 404
```

## Testy bazy danych

Python

```
# tests/integration/test_database.py
import pytest
import psycopg2
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from src.database import Base, User as DBUser
from src.repositories.user_repository import UserRepository

class TestUserRepository:
    """Integration tests for UserRepository with real database"""

    @pytest.fixture(scope="class")
    def db_engine(self):
        """Create test database engine"""
        # Use test database
        engine = create_engine(
            "postgresql://test_user:test_pass@localhost:5432/test_db",
            echo=False
        )

        # Create tables
        Base.metadata.create_all(engine)

        yield engine

        # Cleanup
        Base.metadata.drop_all(engine)

    @pytest.fixture
    def db_session(self, db_engine):
        """Create database session for each test"""
        Session = sessionmaker(bind=db_engine)
        session = Session()

        yield session

        # Rollback any changes
        session.rollback()
        session.close()
```

```

@pytest.fixture
def user_repository(self, db_session):
    """Create UserRepository instance"""
    return UserRepository(db_session)

@pytest.fixture
def sample_user_data(self):
    """Sample user data"""
    return {
        'username': 'testuser',
        'email': 'test@example.com',
        'age': 25,
        'is_active': True
    }

def test_create_user(self, user_repository, sample_user_data):
    """Test user creation in database"""
    user = user_repository.create_user(**sample_user_data)

    assert user.id is not None
    assert user.username == sample_user_data['username']
    assert user.email == sample_user_data['email']
    assert user.age == sample_user_data['age']
    assert user.is_active == sample_user_data['is_active']

def test_get_user_by_id(self, user_repository, sample_user_data):
    """Test user retrieval by ID"""
    # Create user
    created_user = user_repository.create_user(**sample_user_data)

    # Retrieve user
    retrieved_user = user_repository.get_user_by_id(created_user.id)

    assert retrieved_user is not None
    assert retrieved_user.id == created_user.id
    assert retrieved_user.username == sample_user_data['username']

def test_get_user_by_username(self, user_repository, sample_user_data):
    """Test user retrieval by username"""
    # Create user
    user_repository.create_user(**sample_user_data)

    # Retrieve user
    retrieved_user =
user_repository.get_user_by_username(sample_user_data['username'])

    assert retrieved_user is not None

```

```

        assert retrieved_user.username == sample_user_data['username']

def test_get_user_not_found(self, user_repository):
    """Test user retrieval when user doesn't exist"""
    user = user_repository.get_user_by_id(999999)
    assert user is None

    user = user_repository.get_user_by_username('nonexistent')
    assert user is None

def test_update_user(self, user_repository, sample_user_data):
    """Test user update"""
    # Create user
    user = user_repository.create_user(**sample_user_data)

    # Update user
    updated_data = {'email': 'updated@example.com', 'age': 30}
    updated_user = user_repository.update_user(user.id, **updated_data)

    assert updated_user.email == 'updated@example.com'
    assert updated_user.age == 30
    assert updated_user.username == sample_user_data['username'] #
unchanged

def test_delete_user(self, user_repository, sample_user_data):
    """Test user deletion"""
    # Create user
    user = user_repository.create_user(**sample_user_data)
    user_id = user.id

    # Delete user
    result = user_repository.delete_user(user_id)
    assert result is True

    # Verify deletion
    deleted_user = user_repository.get_user_by_id(user_id)
    assert deleted_user is None

def test_list_users(self, user_repository, sample_user_data):
    """Test user listing"""
    # Create multiple users
    users_data = []
    for i in range(3):
        user_data = sample_user_data.copy()
        user_data['username'] = f'testuser{i}'
        user_data['email'] = f'test{i}@example.com'
        users_data.append(user_data)
        user_repository.create_user(**user_data)

```

```

# List all users
users = user_repository.list_users()
assert len(users) >= 3

# List active users only
active_users = user_repository.list_users(active_only=True)
assert all(user.is_active for user in active_users)

def test_user_unique_constraints(self, user_repository,
sample_user_data):
    """Test database unique constraints"""
    # Create first user
    user_repository.create_user(**sample_user_data)

    # Try to create user with same username
    with pytest.raises(Exception): # Should raise integrity error
        duplicate_data = sample_user_data.copy()
        duplicate_data['email'] = 'different@example.com'
        user_repository.create_user(**duplicate_data)

    # Try to create user with same email
    with pytest.raises(Exception): # Should raise integrity error
        duplicate_data = sample_user_data.copy()
        duplicate_data['username'] = 'differentuser'
        user_repository.create_user(**duplicate_data)

def test_transaction_rollback(self, user_repository, sample_user_data):
    """Test transaction rollback on error"""
    try:
        with user_repository.session.begin():
            # Create valid user
            user_repository.create_user(**sample_user_data)

            # Try to create invalid user (should fail)
            invalid_data = sample_user_data.copy()
            invalid_data['email'] = None # This should fail
            user_repository.create_user(**invalid_data)
    except Exception:
        pass # Expected to fail

    # Verify that the first user was not created due to rollback
    user =
user_repository.get_user_by_username(sample_user_data['username'])
assert user is None

```

# Testy wydajności

## Benchmarking z pytest-benchmark

Python

```
# tests/performance/test_benchmarks.py
import pytest
from src.services.user_service import UserService
from src.models.user import User
import time

class TestPerformance:
    """Performance tests for critical functions"""

    @pytest.fixture
    def user_service(self):
        return UserService("https://api.example.com", "test-key")

    @pytest.fixture
    def sample_users(self):
        """Generate sample users for testing"""
        users = []
        for i in range(1000):
            user = User(
                username=f"user{i}",
                email=f"user{i}@example.com",
                age=20 + (i % 50)
            )
            users.append(user)
        return users

    def test_user_creation_performance(self, benchmark):
        """Benchmark user creation"""
        def create_user():
            return User(
                username="testuser",
                email="test@example.com",
                age=25
            )

        result = benchmark(create_user)
        assert result.username == "testuser"

    def test_user_validation_performance(self, benchmark):
        """Benchmark user validation"""
```

```

def validate_email():
    return User.is_valid_email("test@example.com")

result = benchmark(validate_email)
assert result is True

def test_bulk_user_processing(self, benchmark, sample_users):
    """Benchmark bulk user processing"""
    def process_users(users):
        active_users = []
        for user in users:
            if user.is_active:
                active_users.append(user)
        return active_users

    result = benchmark(process_users, sample_users)
    assert len(result) == len(sample_users)

@pytest.mark.slow
def test_concurrent_user_operations(self, benchmark):
    """Benchmark concurrent operations"""
    import concurrent.futures

    def concurrent_user_creation():
        users = []
        with concurrent.futures.ThreadPoolExecutor(max_workers=10) as
executor:
            futures = []
            for i in range(100):
                future = executor.submit(
                    User,
                    username=f"user{i}",
                    email=f"user{i}@example.com",
                    age=25
                )
                futures.append(future)

            for future in concurrent.futures.as_completed(futures):
                users.append(future.result())

        return users

    result = benchmark(concurrent_user_creation)
    assert len(result) == 100

def test_memory_usage(self, sample_users):
    """Test memory usage with large datasets"""
    import psutil

```

```

import os

process = psutil.Process(os.getpid())
initial_memory = process.memory_info().rss

# Process large dataset
processed_users = []
for user in sample_users:
    if user.age > 30:
        processed_users.append(user)

final_memory = process.memory_info().rss
memory_increase = final_memory - initial_memory

# Memory increase should be reasonable (less than 50MB for 1000
users)
assert memory_increase < 50 * 1024 * 1024
assert len(processed_users) > 0

```

## Load testing z locust

Python

```

# tests/load/locustfile.py
from locust import HttpUser, task, between
import random

class UserAPILoadTest(HttpUser):
    """Load test for User API"""

    wait_time = between(1, 3) # Wait 1-3 seconds between requests

    def on_start(self):
        """Setup before starting tests"""
        self.user_ids = []

    @task(3)
    def create_user(self):
        """Create user (weight: 3)"""
        user_id = random.randint(1000, 9999)
        user_data = {
            'username': f'loadtest_user_{user_id}',
            'email': f'loadtest_{user_id}@example.com',
            'age': random.randint(18, 65)
        }

```



```

        with self.client.post('/users', json=user_data, catch_response=True)
as response:
    if response.status_code == 201:
        user_id = response.json().get('id')
        if user_id:
            self.user_ids.append(user_id)
            response.success()
        else:
            response.failure(f"Failed to create user:
{response.status_code}")

@task(5)
def get_user(self):
    """Get user (weight: 5)"""
    if self.user_ids:
        user_id = random.choice(self.user_ids)
        with self.client.get(f'/users/{user_id}', catch_response=True) as
response:
            if response.status_code == 200:
                response.success()
            elif response.status_code == 404:
                # User might have been deleted, remove from list
                if user_id in self.user_ids:
                    self.user_ids.remove(user_id)
                response.success()
            else:
                response.failure(f"Failed to get user:
{response.status_code}")

@task(2)
def list_users(self):
    """List users (weight: 2)"""
    with self.client.get('/users', catch_response=True) as response:
        if response.status_code == 200:
            response.success()
        else:
            response.failure(f"Failed to list users:
{response.status_code}")

@task(1)
def update_user(self):
    """Update user (weight: 1)"""
    if self.user_ids:
        user_id = random.choice(self.user_ids)
        update_data = {
            'age': random.randint(18, 65)
        }

```

```

        with self.client.patch(f'/users/{user_id}', json=update_data,
catch_response=True) as response:
            if response.status_code == 200:
                response.success()
            elif response.status_code == 404:
                # User might have been deleted
                if user_id in self.user_ids:
                    self.user_ids.remove(user_id)
                response.success()
            else:
                response.failure(f"Failed to update user:
{response.status_code}")

@task(1)
def delete_user(self):
    """Delete user (weight: 1)"""
    if self.user_ids:
        user_id = self.user_ids.pop()
        with self.client.delete(f'/users/{user_id}', catch_response=True)
as response:
            if response.status_code in [204, 404]:
                response.success()
            else:
                response.failure(f"Failed to delete user:
{response.status_code}")

# Uruchomienie load testów:
# locust -f tests/load/locustfile.py --host=http://localhost:8000

```

To kończy rozdział o testowaniu aplikacji Python. W następnym rozdziale omówimy skrypty DevOps w Python i praktyczne przykłady automatyzacji.

## 12. Skrypty DevOps w Python

### Wprowadzenie do skryptowania DevOps

Skrypty Python są nieodłącznym elementem nowoczesnych środowisk DevOps, umożliwiając automatyzację złożonych zadań operacyjnych, integrację różnych narzędzi i systemów oraz tworzenie niestandardowych rozwiązań dostosowanych do specyficznych potrzeb organizacji.

Kluczowe zastosowania skryptów Python w DevOps obejmują:

**Automatyzacja deployment** - skrypty do wdrażania aplikacji, zarządzania konfiguracją i orkiestracji procesów release.

**Monitoring i alerting** - zbieranie metryk, analiza logów, wykrywanie anomalii i wysyłanie powiadomień.

**Zarządzanie infrastrukturą** - provisioning zasobów, konfiguracja serwerów, zarządzanie kontenerami.

**Integracja systemów** - łączenie różnych narzędzi DevOps, synchronizacja danych, automatyzacja workflow.

**Raportowanie i analytics** - generowanie raportów, analiza danych operacyjnych, dashboardy.

**Security i compliance** - skanowanie podatności, audyty bezpieczeństwa, zarządzanie certyfikatami.

## Skrypty deployment i release management

### Skrypt deployment aplikacji

Python

```
#!/usr/bin/env python3
"""
Skrypt do automatycznego deployment aplikacji
"""

import os
import sys
import json
import time
import subprocess
import argparse
import logging
from pathlib import Path
from typing import Dict, List, Any, Optional
import yaml
import requests

class DeploymentManager:
```

```

"""Klasa do zarządzania deployment aplikacji"""

def __init__(self, config_file: str):
    self.config = self._load_config(config_file)
    self.logger = self._setup_logging()
    self.deployment_id = int(time.time())

def _load_config(self, config_file: str) -> Dict[str, Any]:
    """Wczytaj konfigurację deployment"""
    try:
        with open(config_file, 'r') as f:
            if config_file.endswith('.yaml') or
config_file.endswith('.yml'):
                return yaml.safe_load(f)
            else:
                return json.load(f)
    except Exception as e:
        print(f"Error loading config: {e}")
        sys.exit(1)

def _setup_logging(self) -> logging.Logger:
    """Konfiguracja logowania"""
    logger = logging.getLogger('deployment')
    logger.setLevel(logging.INFO)

    # Console handler
    console_handler = logging.StreamHandler()
    console_formatter = logging.Formatter(
        '%(asctime)s - %(levelname)s - %(message)s'
    )
    console_handler.setFormatter(console_formatter)
    logger.addHandler(console_handler)

    # File handler
    log_dir = Path(self.config.get('log_dir', '/var/log/deployment'))
    log_dir.mkdir(parents=True, exist_ok=True)

    file_handler = logging.FileHandler(
        log_dir / f'deployment_{self.deployment_id}.log'
    )
    file_formatter = logging.Formatter(
        '%(asctime)s - %(name)s - %(levelname)s - %(message)s'
    )
    file_handler.setFormatter(file_formatter)
    logger.addHandler(file_handler)

    return logger

```

```

def _execute_command(self, command: str, cwd: str = None) -> Dict[str,
Any]:
    """Wykonaj polecenie systemowe"""
    self.logger.info(f"Executing: {command}")

    try:
        result = subprocess.run(
            command,
            shell=True,
            cwd=cwd,
            capture_output=True,
            text=True,
            timeout=300 # 5 minutes timeout
        )

        if result.returncode == 0:
            self.logger.info(f"Command succeeded: {command}")
            return {
                'success': True,
                'stdout': result.stdout,
                'stderr': result.stderr,
                'returncode': result.returncode
            }
        else:
            self.logger.error(f"Command failed: {command}")
            self.logger.error(f"Error output: {result.stderr}")
            return {
                'success': False,
                'stdout': result.stdout,
                'stderr': result.stderr,
                'returncode': result.returncode
            }
    except subprocess.TimeoutExpired:
        self.logger.error(f"Command timed out: {command}")
        return {
            'success': False,
            'error': 'Command timed out'
        }
    except Exception as e:
        self.logger.error(f"Command execution error: {e}")
        return {
            'success': False,
            'error': str(e)
        }

def _send_notification(self, message: str, status: str = "info"):
    """Wyślij powiadomienie"""
    notifications = self.config.get('notifications', {})

```

```

# Slack notification
if 'slack' in notifications:
    slack_config = notifications['slack']
    webhook_url = slack_config['webhook_url']

    color_map = {
        'info': 'good',
        'warning': 'warning',
        'error': 'danger'
    }

    payload = {
        'text': f"Deployment {self.deployment_id}",
        'attachments': [{
            'color': color_map.get(status, 'good'),
            'text': message,
            'ts': int(time.time())
        }]
    }

    try:
        response = requests.post(webhook_url, json=payload,
timeout=10)
        if response.status_code == 200:
            self.logger.info("Slack notification sent")
        else:
            self.logger.warning(f"Failed to send Slack notification:
{response.status_code}")
    except Exception as e:
        self.logger.warning(f"Error sending Slack notification: {e}")

# Email notification (przykład)
if 'email' in notifications:
    # Implementacja wysyłania email
    pass

def pre_deployment_checks(self) -> bool:
    """Sprawdzenia przed deployment"""
    self.logger.info("Running pre-deployment checks")

    checks = self.config.get('pre_deployment_checks', [])

    for check in checks:
        check_name = check['name']
        check_command = check['command']

        self.logger.info(f"Running check: {check_name}")

```

```

        result = self._execute_command(check_command)

        if not result['success']:
            self.logger.error(f"Pre-deployment check failed:
{check_name}")
            return False

        self.logger.info("All pre-deployment checks passed")
        return True

def backup_current_version(self) -> bool:
    """Utwórz backup aktualnej wersji"""
    backup_config = self.config.get('backup', {})

    if not backup_config.get('enabled', False):
        self.logger.info("Backup disabled, skipping")
        return True

    self.logger.info("Creating backup of current version")

    backup_dir = backup_config['directory']
    source_dir = backup_config['source']

    # Utwórz katalog backup
    backup_path = Path(backup_dir) / f"backup_{self.deployment_id}"
    backup_path.mkdir(parents=True, exist_ok=True)

    # Wykonaj backup
    backup_command = f"cp -r {source_dir}/* {backup_path}/"
    result = self._execute_command(backup_command)

    if result['success']:
        self.logger.info(f"Backup created: {backup_path}")
        return True
    else:
        self.logger.error("Backup creation failed")
        return False

def stop_services(self) -> bool:
    """Zatrzymaj usługi"""
    services = self.config.get('services', [])

    self.logger.info("Stopping services")

    for service in services:
        service_name = service['name']
        stop_command = service.get('stop_command', f"sudo systemctl stop
{service_name}")

```

```

        self.logger.info(f"Stopping service: {service_name}")
        result = self._execute_command(stop_command)

        if not result['success']:
            self.logger.error(f"Failed to stop service: {service_name}")
            return False

        # Sprawdź czy usługa została zatrzymana
        check_command = service.get('status_command', f"sudo systemctl
is-active {service_name}")
        time.sleep(2) # Poczekać chwilę

        status_result = self._execute_command(check_command)
        if 'inactive' not in status_result.get('stdout', '').lower():
            self.logger.warning(f"Service {service_name} may not be fully
stopped")

        self.logger.info("All services stopped")
        return True

def deploy_application(self) -> bool:
    """Wdróż aplikację"""
    deployment_config = self.config['deployment']

    self.logger.info("Deploying application")

    # Pobierz artefakt
    if 'artifact' in deployment_config:
        artifact_config = deployment_config['artifact']
        artifact_url = artifact_config['url']
        artifact_path = artifact_config['local_path']

        self.logger.info(f"Downloading artifact: {artifact_url}")
        download_command = f"wget -O {artifact_path} {artifact_url}"
        result = self._execute_command(download_command)

        if not result['success']:
            self.logger.error("Failed to download artifact")
            return False

    # Wykonaj kroki deployment
    steps = deployment_config.get('steps', [])

    for step in steps:
        step_name = step['name']
        step_command = step['command']
        step_cwd = step.get('working_directory')

```



```

        self.logger.info(f"Executing deployment step: {step_name}")
        result = self._execute_command(step_command, cwd=step_cwd)

        if not result['success']:
            self.logger.error(f"Deployment step failed: {step_name}")
            return False

    self.logger.info("Application deployment completed")
    return True

def start_services(self) -> bool:
    """Uruchom usługi"""
    services = self.config.get('services', [])

    self.logger.info("Starting services")

    for service in services:
        service_name = service['name']
        start_command = service.get('start_command', f"sudo systemctl
start {service_name}")

        self.logger.info(f"Starting service: {service_name}")
        result = self._execute_command(start_command)

        if not result['success']:
            self.logger.error(f"Failed to start service: {service_name}")
            return False

        # Sprawdź czy usługa została uruchomiona
        check_command = service.get('status_command', f"sudo systemctl
is-active {service_name}")
        time.sleep(5) # Poczekaaj na uruchomienie

        status_result = self._execute_command(check_command)
        if 'active' not in status_result.get('stdout', '').lower():
            self.logger.error(f"Service {service_name} failed to start
properly")
            return False

    self.logger.info("All services started")
    return True

def post_deployment_tests(self) -> bool:
    """Testy po deployment"""
    tests = self.config.get('post_deployment_tests', [])

    if not tests:

```

```

        self.logger.info("No post-deployment tests configured")
        return True

    self.logger.info("Running post-deployment tests")

    for test in tests:
        test_name = test['name']
        test_command = test['command']

        self.logger.info(f"Running test: {test_name}")
        result = self._execute_command(test_command)

        if not result['success']:
            self.logger.error(f"Post-deployment test failed:
{test_name}")
            return False

    self.logger.info("All post-deployment tests passed")
    return True

def health_check(self) -> bool:
    """Sprawdzenie zdrowia aplikacji"""
    health_config = self.config.get('health_check', {})

    if not health_config:
        self.logger.info("No health check configured")
        return True

    url = health_config['url']
    timeout = health_config.get('timeout', 30)
    retries = health_config.get('retries', 5)
    retry_delay = health_config.get('retry_delay', 10)

    self.logger.info(f"Performing health check: {url}")

    for attempt in range(retries):
        try:
            response = requests.get(url, timeout=timeout)
            if response.status_code == 200:
                self.logger.info("Health check passed")
                return True
            else:
                self.logger.warning(f"Health check failed with status
{response.status_code}")
        except Exception as e:
            self.logger.warning(f"Health check attempt {attempt + 1}
failed: {e}")

```

```

        if attempt < retries - 1:
            self.logger.info(f"Retrying health check in {retry_delay}
seconds")
            time.sleep(retry_delay)

        self.logger.error("Health check failed after all retries")
        return False

def rollback(self) -> bool:
    """Rollback do poprzedniej wersji"""
    self.logger.info("Starting rollback procedure")

    rollback_config = self.config.get('rollback', {})

    if not rollback_config:
        self.logger.error("No rollback configuration found")
        return False

    # Zatrzymaj usługi
    if not self.stop_services():
        self.logger.error("Failed to stop services during rollback")
        return False

    # Przywróć backup
    backup_dir = self.config['backup']['directory']
    target_dir = rollback_config['target_directory']

    # Znajdź najnowszy backup
    backup_path = Path(backup_dir)
    backups = sorted(backup_path.glob('backup_*'), key=lambda x:
x.stat().st_mtime, reverse=True)

    if not backups:
        self.logger.error("No backup found for rollback")
        return False

    latest_backup = backups[0]
    self.logger.info(f"Rolling back to: {latest_backup}")

    rollback_command = f"rm -rf {target_dir}/* && cp -r {latest_backup}/*
{target_dir}/"
    result = self._execute_command(rollback_command)

    if not result['success']:
        self.logger.error("Failed to restore backup during rollback")
        return False

    # Uruchom usługi

```

```

        if not self.start_services():
            self.logger.error("Failed to start services during rollback")
            return False

        self.logger.info("Rollback completed successfully")
        return True

def deploy(self) -> bool:
    """Główna metoda deployment"""
    self.logger.info(f"Starting deployment {self.deployment_id}")
    self._send_notification(f"Deployment {self.deployment_id} started",
"info")

    try:
        # Pre-deployment checks
        if not self.pre_deployment_checks():
            raise Exception("Pre-deployment checks failed")

        # Backup
        if not self.backup_current_version():
            raise Exception("Backup creation failed")

        # Stop services
        if not self.stop_services():
            raise Exception("Failed to stop services")

        # Deploy application
        if not self.deploy_application():
            raise Exception("Application deployment failed")

        # Start services
        if not self.start_services():
            raise Exception("Failed to start services")

        # Post-deployment tests
        if not self.post_deployment_tests():
            raise Exception("Post-deployment tests failed")

        # Health check
        if not self.health_check():
            raise Exception("Health check failed")

        self.logger.info(f"Deployment {self.deployment_id} completed
successfully")
        self._send_notification(f"Deployment {self.deployment_id}
completed successfully", "info")
        return True

```

```

        except Exception as e:
            self.logger.error(f"Deployment failed: {e}")
            self._send_notification(f"Deployment {self.deployment_id} failed: {e}", "error")

            # Attempt rollback
            if self.config.get('auto_rollback', False):
                self.logger.info("Attempting automatic rollback")
                if self.rollback():
                    self._send_notification(f"Automatic rollback completed for deployment {self.deployment_id}", "warning")
                else:
                    self._send_notification(f"Automatic rollback failed for deployment {self.deployment_id}", "error")

            return False

def main():
    parser = argparse.ArgumentParser(description='Application Deployment Script')
    parser.add_argument('--config', required=True, help='Deployment configuration file')
    parser.add_argument('--rollback', action='store_true', help='Perform rollback instead of deployment')
    parser.add_argument('--dry-run', action='store_true', help='Perform dry run without actual deployment')

    args = parser.parse_args()

    # Sprawdź czy plik konfiguracyjny istnieje
    if not os.path.exists(args.config):
        print(f"Configuration file not found: {args.config}")
        sys.exit(1)

    # Inicjalizuj deployment manager
    deployment_manager = DeploymentManager(args.config)

    if args.rollback:
        success = deployment_manager.rollback()
    else:
        if args.dry_run:
            print("Dry run mode - no actual deployment will be performed")
            # Implementacja dry run
            success = True
        else:
            success = deployment_manager.deploy()

    sys.exit(0 if success else 1)

```

```
if __name__ == "__main__":
    main()
```

## Przykład konfiguracji deployment

YAML

```
# deployment-config.yaml
log_dir: "/var/log/deployment"

notifications:
  slack:
    webhook_url: "https://hooks.slack.com/services/YOUR/WEBHOOK/URL"

backup:
  enabled: true
  directory: "/backup/deployments"
  source: "/opt/myapp"

services:
  - name: "myapp"
    stop_command: "sudo systemctl stop myapp"
    start_command: "sudo systemctl start myapp"
    status_command: "sudo systemctl is-active myapp"
  - name: "nginx"
    stop_command: "sudo systemctl reload nginx"
    start_command: "sudo systemctl start nginx"

pre_deployment_checks:
  - name: "Check disk space"
    command: "df -h / | awk 'NR==2 {if($5+0 > 90) exit 1}'"
  - name: "Check memory"
    command: "free | awk 'NR==2{printf \"%.2f%\\\", $3*100/$2}' | awk '{if($1+0 > 90) exit 1}'"

deployment:
  artifact:
    url: "https://releases.example.com/myapp/v1.2.3/myapp.tar.gz"
    local_path: "/tmp/myapp.tar.gz"

  steps:
    - name: "Extract application"
      command: "tar -xzf /tmp/myapp.tar.gz -C /opt/"
      working_directory: "/tmp"
```

```
- name: "Install dependencies"
  command: "pip install -r requirements.txt"
  working_directory: "/opt/myapp"

- name: "Update configuration"
  command: "cp config/production.conf /etc/myapp/app.conf"
  working_directory: "/opt/myapp"

- name: "Run database migrations"
  command: "python manage.py migrate"
  working_directory: "/opt/myapp"

post_deployment_tests:
- name: "Check application startup"
  command: "curl -f http://localhost:8080/health"

- name: "Run smoke tests"
  command: "python -m pytest tests/smoke/"

health_check:
  url: "http://localhost:8080/health"
  timeout: 30
  retries: 5
  retry_delay: 10

rollback:
  target_directory: "/opt/myapp"

auto_rollback: true
```

## Skrypty monitoringu i alertingu

### System monitoringu zasobów

Python

```
#!/usr/bin/env python3
"""
Zaawansowany system monitoringu zasobów systemowych
"""

import psutil
import time
import json
import smtplib
```

```

import requests
from datetime import datetime, timedelta
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from typing import Dict, List, Any, Optional
import logging
import sqlite3
from dataclasses import dataclass, asdict
import threading
import queue

@dataclass
class MetricData:
    """Klasa reprezentująca dane metryki"""
    timestamp: datetime
    metric_name: str
    value: float
    unit: str
    hostname: str
    tags: Dict[str, str] = None

class MetricsCollector:
    """Klasa do zbierania metryk systemowych"""

    def __init__(self, hostname: str = None):
        self.hostname = hostname or psutil.os.uname().nodename
        self.logger = logging.getLogger(__name__)

    def collect_cpu_metrics(self) -> List[MetricData]:
        """Zbierz metryki CPU"""
        metrics = []

        # Ogólne użycie CPU
        cpu_percent = psutil.cpu_percent(interval=1)
        metrics.append(MetricData(
            timestamp=datetime.now(),
            metric_name="cpu.usage.percent",
            value=cpu_percent,
            unit="percent",
            hostname=self.hostname
        ))

        # Użycie CPU per core
        cpu_per_core = psutil.cpu_percent(interval=1, percpu=True)
        for i, usage in enumerate(cpu_per_core):
            metrics.append(MetricData(
                timestamp=datetime.now(),
                metric_name="cpu.core.usage.percent",

```



```

        value=usage,
        unit="percent",
        hostname=self.hostname,
        tags={"core": str(i)}
    ))

# Load average (Linux/macOS)
try:
    load_avg = psutil.getloadavg()
    for i, period in enumerate(['1m', '5m', '15m']):
        metrics.append(MetricData(
            timestamp=datetime.now(),
            metric_name=f"system.load.{period}",
            value=load_avg[i],
            unit="load",
            hostname=self.hostname
        ))
except AttributeError:
    pass # Windows doesn't have load average

return metrics

def collect_memory_metrics(self) -> List[MetricData]:
    """Zbierz metryki pamięci"""
    metrics = []

    # Virtual memory
    memory = psutil.virtual_memory()

    metrics.extend([
        MetricData(
            timestamp=datetime.now(),
            metric_name="memory.usage.percent",
            value=memory.percent,
            unit="percent",
            hostname=self.hostname
        ),
        MetricData(
            timestamp=datetime.now(),
            metric_name="memory.used.bytes",
            value=memory.used,
            unit="bytes",
            hostname=self.hostname
        ),
        MetricData(
            timestamp=datetime.now(),
            metric_name="memory.available.bytes",
            value=memory.available,

```

```

        unit="bytes",
        hostname=self.hostname
    ),
    MetricData(
        timestamp=datetime.now(),
        metric_name="memory.total.bytes",
        value=memory.total,
        unit="bytes",
        hostname=self.hostname
    )
])

# Swap memory
swap = psutil.swap_memory()
metrics.extend([
    MetricData(
        timestamp=datetime.now(),
        metric_name="swap.usage.percent",
        value=swap.percent,
        unit="percent",
        hostname=self.hostname
    ),
    MetricData(
        timestamp=datetime.now(),
        metric_name="swap.used.bytes",
        value=swap.used,
        unit="bytes",
        hostname=self.hostname
    )
])

return metrics

def collect_disk_metrics(self) -> List[MetricData]:
    """Zbierz metryki dysku"""
    metrics = []

    # Disk usage per partition
    for partition in psutil.disk_partitions():
        try:
            usage = psutil.disk_usage(partition.mountpoint)

            metrics.extend([
                MetricData(
                    timestamp=datetime.now(),
                    metric_name="disk.usage.percent",
                    value=(usage.used / usage.total) * 100,
                    unit="percent",

```

```

        hostname=self.hostname,
        tags={"device": partition.device, "mountpoint":
partition.mountpoint}
    ),
    MetricData(
        timestamp=datetime.now(),
        metric_name="disk.used.bytes",
        value=usage.used,
        unit="bytes",
        hostname=self.hostname,
        tags={"device": partition.device, "mountpoint":
partition.mountpoint}
    ),
    MetricData(
        timestamp=datetime.now(),
        metric_name="disk.free.bytes",
        value=usage.free,
        unit="bytes",
        hostname=self.hostname,
        tags={"device": partition.device, "mountpoint":
partition.mountpoint}
    )
    ])
except PermissionError:
    continue

# Disk I/O
disk_io = psutil.disk_io_counters()
if disk_io:
    metrics.extend([
        MetricData(
            timestamp=datetime.now(),
            metric_name="disk.read.bytes",
            value=disk_io.read_bytes,
            unit="bytes",
            hostname=self.hostname
        ),
        MetricData(
            timestamp=datetime.now(),
            metric_name="disk.write.bytes",
            value=disk_io.write_bytes,
            unit="bytes",
            hostname=self.hostname
        ),
        MetricData(
            timestamp=datetime.now(),
            metric_name="disk.read.count",
            value=disk_io.read_count,

```

```

        unit="count",
        hostname=self.hostname
    ),
    MetricData(
        timestamp=datetime.now(),
        metric_name="disk.write.count",
        value=disk_io.write_count,
        unit="count",
        hostname=self.hostname
    )
])

return metrics

def collect_network_metrics(self) -> List[MetricData]:
    """Zbierz metryki sieci"""
    metrics = []

    # Network I/O
    network_io = psutil.net_io_counters()

    metrics.extend([
        MetricData(
            timestamp=datetime.now(),
            metric_name="network.bytes.sent",
            value=network_io.bytes_sent,
            unit="bytes",
            hostname=self.hostname
        ),
        MetricData(
            timestamp=datetime.now(),
            metric_name="network.bytes.recv",
            value=network_io.bytes_recv,
            unit="bytes",
            hostname=self.hostname
        ),
        MetricData(
            timestamp=datetime.now(),
            metric_name="network.packets.sent",
            value=network_io.packets_sent,
            unit="count",
            hostname=self.hostname
        ),
        MetricData(
            timestamp=datetime.now(),
            metric_name="network.packets.recv",
            value=network_io.packets_recv,
            unit="count",

```

```

        hostname=self.hostname
    )
])

# Network connections
connections = psutil.net_connections()
connection_states = {}

for conn in connections:
    state = conn.status
    connection_states[state] = connection_states.get(state, 0) + 1

for state, count in connection_states.items():
    metrics.append(MetricData(
        timestamp=datetime.now(),
        metric_name="network.connections.count",
        value=count,
        unit="count",
        hostname=self.hostname,
        tags={"state": state}
    ))

return metrics

def collect_process_metrics(self) -> List[MetricData]:
    """Zbierz metryki procesów"""
    metrics = []

    # Liczba procesów
    process_count = len(psutil.pids())
    metrics.append(MetricData(
        timestamp=datetime.now(),
        metric_name="processes.count",
        value=process_count,
        unit="count",
        hostname=self.hostname
    ))

    # Top processes by CPU
    processes = []
    for proc in psutil.process_iter(['pid', 'name', 'cpu_percent',
'memory_percent']):
        try:
            processes.append(proc.info)
        except (psutil.NoSuchProcess, psutil.AccessDenied):
            pass

    # Sort by CPU usage

```

```

        top_cpu_processes = sorted(processes, key=lambda x: x['cpu_percent']
or 0, reverse=True)[:5]

        for i, proc in enumerate(top_cpu_processes):
            metrics.append(MetricData(
                timestamp=datetime.now(),
                metric_name="process.cpu.percent",
                value=proc['cpu_percent'] or 0,
                unit="percent",
                hostname=self.hostname,
                tags={"process": proc['name'], "pid": str(proc['pid']),
"rank": str(i+1)}
            ))

        return metrics

def collect_all_metrics(self) -> List[MetricData]:
    """Zbierz wszystkie metryki"""
    all_metrics = []

    try:
        all_metrics.extend(self.collect_cpu_metrics())
    except Exception as e:
        self.logger.error(f"Error collecting CPU metrics: {e}")

    try:
        all_metrics.extend(self.collect_memory_metrics())
    except Exception as e:
        self.logger.error(f"Error collecting memory metrics: {e}")

    try:
        all_metrics.extend(self.collect_disk_metrics())
    except Exception as e:
        self.logger.error(f"Error collecting disk metrics: {e}")

    try:
        all_metrics.extend(self.collect_network_metrics())
    except Exception as e:
        self.logger.error(f"Error collecting network metrics: {e}")

    try:
        all_metrics.extend(self.collect_process_metrics())
    except Exception as e:
        self.logger.error(f"Error collecting process metrics: {e}")

    return all_metrics

class AlertManager:

```

```

"""Klasa do zarządzania alertami"""

def __init__(self, config: Dict[str, Any]):
    self.config = config
    self.logger = logging.getLogger(__name__)
    self.alert_history = {}

    def check_thresholds(self, metrics: List[MetricData]) -> List[Dict[str, Any]]:
        """Sprawdź progi alarmowe"""
        alerts = []
        thresholds = self.config.get('thresholds', {})

        for metric in metrics:
            metric_key = metric.metric_name
            if metric_key in thresholds:
                threshold_config = thresholds[metric_key]

                # Sprawdź próg krytyczny
                if 'critical' in threshold_config and metric.value >=
threshold_config['critical']:
                    alert = {
                        'level': 'critical',
                        'metric': metric_key,
                        'value': metric.value,
                        'threshold': threshold_config['critical'],
                        'hostname': metric.hostname,
                        'timestamp': metric.timestamp,
                        'message': f"CRITICAL: {metric_key} is {metric.value}
{metric.unit} (threshold: {threshold_config['critical']}{metric.unit})"
                    }
                    alerts.append(alert)

                # Sprawdź próg ostrzegawczy
                elif 'warning' in threshold_config and metric.value >=
threshold_config['warning']:
                    alert = {
                        'level': 'warning',
                        'metric': metric_key,
                        'value': metric.value,
                        'threshold': threshold_config['warning'],
                        'hostname': metric.hostname,
                        'timestamp': metric.timestamp,
                        'message': f"WARNING: {metric_key} is {metric.value}
{metric.unit} (threshold: {threshold_config['warning']}{metric.unit})"
                    }
                    alerts.append(alert)

```

```

        return alerts

def should_send_alert(self, alert: Dict[str, Any]) -> bool:
    """Sprawdź czy alert powinien być wysłany (throttling)"""
    alert_key = f"{alert['hostname']}:{alert['metric']}:{alert['level']}"
    now = datetime.now()

    # Sprawdź czy alert był już wysłany w ostatnim czasie
    if alert_key in self.alert_history:
        last_sent = self.alert_history[alert_key]
        throttle_minutes = self.config.get('alert_throttle_minutes', 15)

        if now - last_sent < timedelta(minutes=throttle_minutes):
            return False

    self.alert_history[alert_key] = now
    return True

def send_email_alert(self, alert: Dict[str, Any]):
    """Wyślij alert email"""
    email_config = self.config.get('email', {})

    if not email_config.get('enabled', False):
        return

    try:
        msg = Multipart()
        msg['From'] = email_config['from']
        msg['To'] = ', '.join(email_config['to'])
        msg['Subject'] = f"[{alert['level'].upper()}] System Alert - {alert['hostname']}"

        body = f"""
System Alert Details:

Hostname: {alert['hostname']}
Metric: {alert['metric']}
Current Value: {alert['value']}
Threshold: {alert['threshold']}
Level: {alert['level'].upper()}
Timestamp: {alert['timestamp']}

Message: {alert['message']}

Please investigate this issue immediately.
"""

        msg.attach(MimeText(body, 'plain'))

```



```

        server = smtplib.SMTP(email_config['smtp_server'],
email_config['smtp_port'])
        if email_config.get('use_tls', True):
            server.starttls()

        if 'username' in email_config:
            server.login(email_config['username'],
email_config['password'])

        server.send_message(msg)
        server.quit()

        self.logger.info(f"Email alert sent for {alert['metric']}")

    except Exception as e:
        self.logger.error(f"Failed to send email alert: {e}")

def send_slack_alert(self, alert: Dict[str, Any]):
    """Wyślij alert Slack"""
    slack_config = self.config.get('slack', {})

    if not slack_config.get('enabled', False):
        return

    try:
        color_map = {
            'warning': 'warning',
            'critical': 'danger'
        }

        payload = {
            'text': f"System Alert - {alert['hostname']}",
            'attachments': [{
                'color': color_map.get(alert['level'], 'warning'),
                'fields': [
                    {'title': 'Hostname', 'value': alert['hostname'],
'short': True},
                    {'title': 'Metric', 'value': alert['metric'],
'short': True},
                    {'title': 'Value', 'value': str(alert['value']),
'short': True},
                    {'title': 'Threshold', 'value':
str(alert['threshold']), 'short': True},
                    {'title': 'Level', 'value': alert['level'].upper(),
'short': True},
                    {'title': 'Time', 'value':
alert['timestamp'].strftime('%Y-%m-%d %H:%M:%S'), 'short': True}

```

```

        ],
        'text': alert['message'],
        'ts': int(alert['timestamp'].timestamp())
    }]
}

response = requests.post(
    slack_config['webhook_url'],
    json=payload,
    timeout=10
)

if response.status_code == 200:
    self.logger.info(f"Slack alert sent for {alert['metric']}")
else:
    self.logger.error(f"Failed to send Slack alert:
{response.status_code}")

except Exception as e:
    self.logger.error(f"Failed to send Slack alert: {e}")

def process_alerts(self, alerts: List[Dict[str, Any]]):
    """Przetwórz alerty"""
    for alert in alerts:
        if self.should_send_alert(alert):
            self.logger.warning(f"Sending alert: {alert['message']}")

            # Wyślij email
            self.send_email_alert(alert)

            # Wyślij Slack
            self.send_slack_alert(alert)

class MetricsStorage:
    """Klasa do przechowywania metryk"""

    def __init__(self, db_path: str):
        self.db_path = db_path
        self.logger = logging.getLogger(__name__)
        self._init_database()

    def _init_database(self):
        """Inicjalizuj bazę danych"""
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()

        cursor.execute('''
            CREATE TABLE IF NOT EXISTS metrics (

```

```

        id INTEGER PRIMARY KEY AUTOINCREMENT,
        timestamp TEXT NOT NULL,
        metric_name TEXT NOT NULL,
        value REAL NOT NULL,
        unit TEXT NOT NULL,
        hostname TEXT NOT NULL,
        tags TEXT
    )
'''

# Indeksy dla wydajności
cursor.execute('CREATE INDEX IF NOT EXISTS idx_timestamp ON
metrics(timestamp)')
cursor.execute('CREATE INDEX IF NOT EXISTS idx_metric_name ON
metrics(metric_name)')
cursor.execute('CREATE INDEX IF NOT EXISTS idx_hostname ON
metrics(hostname)')

conn.commit()
conn.close()

def store_metrics(self, metrics: List[MetricData]):
    """Zapisz metryki do bazy danych"""
    try:
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()

        for metric in metrics:
            cursor.execute('''
                INSERT INTO metrics (timestamp, metric_name, value, unit,
hostname, tags)
                VALUES (?, ?, ?, ?, ?, ?)
            ''', (
                metric.timestamp.isoformat(),
                metric.metric_name,
                metric.value,
                metric.unit,
                metric.hostname,
                json.dumps(metric.tags) if metric.tags else None
            ))

        conn.commit()
        conn.close()

        self.logger.debug(f"Stored {len(metrics)} metrics")

    except Exception as e:
        self.logger.error(f"Failed to store metrics: {e}")

```

```

def cleanup_old_metrics(self, retention_days: int = 30):
    """Usuń stare metryki"""
    try:
        cutoff_date = datetime.now() - timedelta(days=retention_days)

        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()

        cursor.execute(
            'DELETE FROM metrics WHERE timestamp < ?',
            (cutoff_date.isoformat(),)
        )

        deleted_count = cursor.rowcount
        conn.commit()
        conn.close()

        if deleted_count > 0:
            self.logger.info(f"Cleaned up {deleted_count} old metrics")

    except Exception as e:
        self.logger.error(f"Failed to cleanup old metrics: {e}")

class MonitoringSystem:
    """Główna klasa systemu monitoringu"""

    def __init__(self, config_file: str):
        self.config = self._load_config(config_file)
        self.logger = self._setup_logging()

        self.metrics_collector = MetricsCollector()
        self.alert_manager = AlertManager(self.config)
        self.metrics_storage =
MetricsStorage(self.config.get('database_path',
'/var/lib/monitoring/metrics.db'))

        self.running = False
        self.metrics_queue = queue.Queue()

    def _load_config(self, config_file: str) -> Dict[str, Any]:
        """Wczytaj konfigurację"""
        with open(config_file, 'r') as f:
            if config_file.endswith('.yaml') or config_file.endswith('.yml'):
                import yaml
                return yaml.safe_load(f)
            else:
                return json.load(f)

```

```

def _setup_logging(self) -> logging.Logger:
    """Konfiguracja logowania"""
    logger = logging.getLogger('monitoring')
    logger.setLevel(logging.INFO)

    formatter = logging.Formatter(
        '%(asctime)s - %(name)s - %(levelname)s - %(message)s'
    )

    # Console handler
    console_handler = logging.StreamHandler()
    console_handler.setFormatter(formatter)
    logger.addHandler(console_handler)

    # File handler
    if 'log_file' in self.config:
        file_handler = logging.FileHandler(self.config['log_file'])
        file_handler.setFormatter(formatter)
        logger.addHandler(file_handler)

    return logger

def _metrics_worker(self):
    """Worker thread do przetwarzania metryk"""
    while self.running:
        try:
            metrics = self.metrics_queue.get(timeout=1)

            # Zapisz metryki
            self.metrics_storage.store_metrics(metrics)

            # Sprawdź alerty
            alerts = self.alert_manager.check_thresholds(metrics)
            if alerts:
                self.alert_manager.process_alerts(alerts)

            self.metrics_queue.task_done()

        except queue.Empty:
            continue
        except Exception as e:
            self.logger.error(f"Error in metrics worker: {e}")

def start(self):
    """Uruchom system monitoringu"""
    self.logger.info("Starting monitoring system")
    self.running = True

```

```

        # Uruchom worker thread
        worker_thread = threading.Thread(target=self._metrics_worker,
daemon=True)
        worker_thread.start()

        collection_interval = self.config.get('collection_interval', 60)

        try:
            while self.running:
                # Zbierz metryki
                metrics = self.metrics_collector.collect_all_metrics()

                # Dodaj do kolejki
                self.metrics_queue.put(metrics)

                self.logger.debug(f"Collected {len(metrics)} metrics")

                # Poczekaj do następnego cyklu
                time.sleep(collection_interval)

        except KeyboardInterrupt:
            self.logger.info("Monitoring system stopped by user")
        except Exception as e:
            self.logger.error(f"Error in monitoring system: {e}")
        finally:
            self.stop()

    def stop(self):
        """Zatrzymaj system monitoringu"""
        self.logger.info("Stopping monitoring system")
        self.running = False

        # Poczekaj na zakończenie przetwarzania
        self.metrics_queue.join()

        # Wyczyść stare metryki
        retention_days = self.config.get('retention_days', 30)
        self.metrics_storage.cleanup_old_metrics(retention_days)

# Przykład konfiguracji
monitoring_config = {
    "collection_interval": 60,
    "retention_days": 30,
    "database_path": "/var/lib/monitoring/metrics.db",
    "log_file": "/var/log/monitoring.log",

    "thresholds": {

```

```

        "cpu.usage.percent": {
            "warning": 80,
            "critical": 95
        },
        "memory.usage.percent": {
            "warning": 85,
            "critical": 95
        },
        "disk.usage.percent": {
            "warning": 85,
            "critical": 95
        },
        "system.load.1m": {
            "warning": 4,
            "critical": 8
        }
    },

    "alert_throttle_minutes": 15,

    "email": {
        "enabled": True,
        "smtp_server": "smtp.gmail.com",
        "smtp_port": 587,
        "use_tls": True,
        "username": "alerts@company.com",
        "password": "app_password",
        "from": "alerts@company.com",
        "to": ["devops@company.com", "admin@company.com"]
    },

    "slack": {
        "enabled": True,
        "webhook_url": "https://hooks.slack.com/services/YOUR/WEBHOOK/URL"
    }
}

if __name__ == "__main__":
    import argparse

    parser = argparse.ArgumentParser(description='System Monitoring')
    parser.add_argument('--config', required=True, help='Configuration file')
    args = parser.parse_args()

    monitoring_system = MonitoringSystem(args.config)
    monitoring_system.start()

```

To kończy rozdział o skryptach DevOps w Python. Te przykłady pokazują, jak Python może być używany do automatyzacji złożonych zadań operacyjnych, od deployment aplikacji po zaawansowane systemy monitoringu. W następnym rozdziale przejdziemy do Terraform i Infrastructure as Code.

## 13. Wprowadzenie do Infrastructure as Code

### Czym jest Infrastructure as Code

Infrastructure as Code (IaC) to fundamentalna praktyka DevOps, która polega na zarządzaniu i provisioningu infrastruktury IT za pomocą kodu maszynowego, zamiast ręcznych procesów konfiguracyjnych. Ta metodologia rewolucjonizuje sposób, w jaki organizacje podchodzą do zarządzania infrastrukturą, wprowadzając te same zasady i praktyki, które sprawdziły się w rozwoju oprogramowania.

Tradycyjne podejście do zarządzania infrastrukturą opierało się na ręcznej konfiguracji serwerów, sieci i innych komponentów infrastruktury. Administratorzy systemów logowali się do serwerów, instalowali oprogramowanie, konfigurowali usługi i wprowadzali zmiany poprzez interfejsy graficzne lub polecenia wykonywane bezpośrednio w systemie. To podejście, choć intuicyjne, niosło ze sobą liczne problemy: brak spójności między środowiskami, trudności w odtwarzaniu konfiguracji, podatność na błędy ludzkie oraz brak możliwości śledzenia zmian.

Infrastructure as Code fundamentalnie zmienia ten paradygmat. Zamiast ręcznej konfiguracji, cała infrastruktura jest definiowana w plikach konfiguracyjnych, które można wersjonować, testować, recenzować i automatycznie wdrażać. Te pliki stają się "źródłem prawdy" dla infrastruktury, dokładnie opisując pożądany stan wszystkich komponentów systemu.

Kluczowe charakterystyki Infrastructure as Code obejmują deklaratywność, gdzie opisujemy pożądany stan infrastruktury, a nie kroki potrzebne do jego osiągnięcia. System IaC automatycznie określa, jakie działania są potrzebne, aby osiągnąć ten stan. Idempotentność zapewnia, że wielokrotne wykonanie tego samego kodu IaC zawsze prowadzi do tego samego rezultatu, niezależnie od początkowego stanu infrastruktury. Wersjonowanie



umożliwia śledzenie wszystkich zmian w infrastrukturze, podobnie jak w przypadku kodu aplikacji, co pozwala na łatwe cofanie zmian i audyt historii modyfikacji.

## Korzyści z Infrastructure as Code

Implementacja Infrastructure as Code przynosi organizacjom liczne korzyści, które znacząco wpływają na efektywność, niezawodność i skalowalność operacji IT.

**Spójność i powtarzalność** stanowią jedną z najważniejszych korzyści IaC. Gdy infrastruktura jest definiowana w kodzie, każde środowisko może być tworzone w identyczny sposób. Eliminuje to problem "działa na moim komputerze", gdzie aplikacja funkcjonuje poprawnie w środowisku deweloperskim, ale napotyka problemy w produkcji z powodu różnic w konfiguracji. Dzięki IaC, środowiska deweloperskie, testowe i produkcyjne mogą być identyczne, co znacząco redukuje ryzyko problemów związanych z różnicami w konfiguracji.

**Szybkość wdrażania** to kolejna kluczowa korzyść. Automatyzacja procesu tworzenia infrastruktury oznacza, że nowe środowiska mogą być tworzone w ciągu minut, a nie dni czy tygodni. To szczególnie ważne w organizacjach praktykujących ciągłe dostarczanie, gdzie szybkość reakcji na zmiany biznesowe jest kluczowa dla konkurencyjności.

**Redukcja błędów** wynika z eliminacji ręcznych procesów konfiguracyjnych. Ludzkie błędy są jedną z głównych przyczyn awarii systemów IT. Automatyzacja poprzez IaC znacząco redukuje prawdopodobieństwo popełnienia błędów, ponieważ te same, przetestowane procedury są wykonywane za każdym razem.

**Skalowalność** infrastruktury staje się znacznie łatwiejsza do zarządzania. Gdy organizacja potrzebuje więcej zasobów, może po prostu zmodyfikować parametry w kodzie IaC i automatycznie wdrożyć dodatkowe komponenty. To szczególnie ważne w środowiskach chmurowych, gdzie elastyczność jest jedną z głównych korzyści.

**Kontrola wersji i audytowalność** umożliwiają śledzenie wszystkich zmian w infrastrukturze. Każda modyfikacja jest dokumentowana, można łatwo określić, kto, kiedy i dlaczego wprowadził konkretną zmianę. To nie tylko ułatwia debugowanie problemów, ale także spełnia wymagania compliance w wielu branżach.

**Współpraca zespołowa** zostaje znacząco ulepszona, gdy infrastruktura jest traktowana jak kod. Zespoły mogą używać tych samych narzędzi i procesów, które sprawdziły się w rozwoju oprogramowania: code review, pull requesty, automatyczne testy i ciągła integracja.

## Narzędzia Infrastructure as Code

Ekosystem narzędzi IaC jest bogaty i różnorodny, oferując rozwiązania dostosowane do różnych potrzeb i środowisk. Można je podzielić na kilka kategorii w zależności od ich podejścia i zastosowania.

**Terraform** firmy HashiCorp jest jednym z najpopularniejszych narzędzi IaC. Charakteryzuje się podejściem cloud-agnostic, co oznacza, że może zarządzać infrastrukturą w różnych dostawcach chmury oraz on-premise. Terraform używa własnego języka konfiguracyjnego HCL (HashiCorp Configuration Language), który jest zarówno czytelny dla człowieka, jak i łatwy do parsowania przez maszyny. Kluczową cechą Terraform jest planowanie zmian - przed wprowadzeniem jakichkolwiek modyfikacji, narzędzie pokazuje dokładnie, co zostanie zmienione, dodane lub usunięte.

**AWS CloudFormation** to natywne narzędzie Amazon Web Services do zarządzania infrastrukturą AWS. Używa formatów JSON lub YAML do definiowania zasobów i ich zależności. CloudFormation oferuje głęboką integrację z ekosystemem AWS i jest często preferowane przez organizacje, które standardowo używają usług AWS.

**Azure Resource Manager (ARM) Templates** to odpowiednik CloudFormation dla Microsoft Azure. Podobnie jak CloudFormation, oferuje natywną integrację z usługami Azure i używa formatu JSON do definiowania infrastruktury.

**Google Cloud Deployment Manager** pełni analogiczną rolę w ekosystemie Google Cloud Platform, używając formatów YAML, Python lub Jinja2 do definiowania infrastruktury.

**Pulumi** to nowoczesne narzędzie IaC, które pozwala na definiowanie infrastruktury przy użyciu popularnych języków programowania takich jak Python, JavaScript, TypeScript, Go czy C#. To podejście jest szczególnie atrakcyjne dla zespołów deweloperskich, które mogą używać znajomych im języków i narzędzi.

**Ansible** to narzędzie automatyzacji, które może być używane do zarządzania infrastrukturą, choć jego główną domeną jest zarządzanie konfiguracją. Ansible używa prostego języka

YAML i nie wymaga instalacji agentów na zarządzanych systemach.

## Terraform - wprowadzenie i podstawy

Terraform wyróżnia się na tle innych narzędzi IaC kilkoma kluczowymi cechami, które czynią go szczególnie atrakcyjnym dla organizacji o różnorodnej infrastrukturze.

**Architektura Terraform** opiera się na koncepcji providerów, które są pluginami odpowiedzialnymi za komunikację z różnymi dostawcami usług. Istnieją providery dla wszystkich głównych dostawców chmury (AWS, Azure, Google Cloud), ale także dla usług SaaS (GitHub, Datadog, PagerDuty), baz danych (PostgreSQL, MySQL) i wielu innych systemów. Ta architektura sprawia, że Terraform może zarządzać praktycznie każdym aspektem nowoczesnej infrastruktury IT.

**HashiCorp Configuration Language (HCL)** to język konfiguracyjny zaprojektowany specjalnie dla Terraform. Jest on bardziej ekspresyjny niż JSON czy YAML, oferując funkcje takie jak zmienne, funkcje, wyrażenia warunkowe i pętle. Jednocześnie pozostaje czytelny i łatwy do nauki dla osób bez głębokiego doświadczenia programistycznego.

**Stan infrastruktury** jest jedną z najważniejszych koncepcji w Terraform. Narzędzie przechowuje informacje o aktualnym stanie zarządzanej infrastruktury w pliku state, który służy jako mapa między konfiguracją a rzeczywistymi zasobami. Ten mechanizm umożliwia Terraform określenie, jakie zmiany są potrzebne podczas każdego uruchomienia.

**Planowanie zmian** to proces, podczas którego Terraform analizuje różnice między pożądanym stanem (zdefiniowanym w konfiguracji) a aktualnym stanem (zapisanym w pliku state) i tworzy plan działań potrzebnych do osiągnięcia pożądanego stanu. Ten plan może być przejrzany przed wykonaniem, co daje kontrolę nad wprowadzanymi zmianami.

## Instalacja i konfiguracja Terraform

### Krok 1: Instalacja Terraform

**Linux (Ubuntu/Debian):**

```
Bash
```

```
# Dodaj klucz GPG HashiCorp
curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -

# Dodaj repozytorium HashiCorp
sudo apt-add-repository "deb [arch=amd64] https://apt.releases.hashicorp.com
$(lsb_release -cs) main"

# Aktualizuj listę pakietów
sudo apt update

# Zainstaluj Terraform
sudo apt install terraform

# Sprawdź instalację
terraform version
```

## Linux (CentOS/RHEL/Fedora):

Bash

```
# Dodaj repozytorium HashiCorp
sudo yum install -y yum-utils
sudo yum-config-manager --add-repo
https://rpm.releases.hashicorp.com/RHEL/hashicorp.repo

# Zainstaluj Terraform
sudo yum install terraform

# Sprawdź instalację
terraform version
```

## macOS:

Bash

```
# Używając Homebrew
brew tap hashicorp/tap
brew install hashicorp/tap/terraform

# Sprawdź instalację
terraform version
```

## Windows:

---

Plain Text

```
# Używając Chocolatey
choco install terraform

# Lub pobierz bezpośrednio z strony HashiCorp
# https://www.terraform.io/downloads.html
```

## Instalacja z binarki (uniwersalna metoda):

Bash

```
# Pobierz najnowszą wersję (sprawdź aktualną wersję na stronie Terraform)
wget
https://releases.hashicorp.com/terraform/1.6.0/terraform_1.6.0_linux_amd64.zip

# Rozpakuj
unzip terraform_1.6.0_linux_amd64.zip

# Przenieś do katalogu w PATH
sudo mv terraform /usr/local/bin/

# Sprawdź instalację
terraform version
```

## Krok 2: Konfiguracja środowiska

### Konfiguracja autocompletowania:

Bash

```
# Bash
terraform -install-autocomplete

# Zsh
echo 'autoload -U +X bashcompleteinit && bashcompleteinit' >> ~/.zshrc
echo 'complete -o nospace -C /usr/local/bin/terraform terraform' >> ~/.zshrc
```

### Konfiguracja zmiennych środowiskowych:

Bash

```
# Dodaj do ~/.bashrc lub ~/.zshrc
export TF_LOG=INFO # Poziom logowania (TRACE, DEBUG, INFO, WARN, ERROR)
export TF_LOG_PATH=/tmp/terraform.log # Ścieżka do pliku logów
export TF_DATA_DIR=~/.terraform.d # Katalog danych Terraform
export TF_PLUGIN_CACHE_DIR=~/.terraform.d/plugin-cache # Cache dla pluginów
```

## Krok 3: Pierwszy projekt Terraform

### Struktura katalogów:

Plain Text

```
terraform-project/
├─ main.tf          # Główna konfiguracja
├─ variables.tf     # Definicje zmiennych
├─ outputs.tf       # Definicje outputów
├─ terraform.tfvars # Wartości zmiennych
└─ versions.tf      # Wymagania wersji
```

### Przykład prostej konfiguracji (main.tf):

Plain Text

```
# Konfiguracja providera
terraform {
  required_version = ">= 1.0"
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
  }
}

# Konfiguracja providera AWS
provider "aws" {
  region = var.aws_region

  default_tags {
    tags = {
      Environment = var.environment
      Project      = var.project_name
      ManagedBy    = "Terraform"
    }
  }
}
```

```

    }
}

# Zasób VPC
resource "aws_vpc" "main" {
    cidr_block          = var.vpc_cidr
    enable_dns_hostnames = true
    enable_dns_support  = true

    tags = {
        Name = "${var.project_name}-vpc"
    }
}

# Subnet publiczny
resource "aws_subnet" "public" {
    count = length(var.public_subnet_cidrs)

    vpc_id          = aws_vpc.main.id
    cidr_block       = var.public_subnet_cidrs[count.index]
    availability_zone =
data.aws_availability_zones.available.names[count.index]
    map_public_ip_on_launch = true

    tags = {
        Name = "${var.project_name}-public-subnet-${count.index + 1}"
        Type = "Public"
    }
}

# Internet Gateway
resource "aws_internet_gateway" "main" {
    vpc_id = aws_vpc.main.id

    tags = {
        Name = "${var.project_name}-igw"
    }
}

# Route table dla subnetów publicznych
resource "aws_route_table" "public" {
    vpc_id = aws_vpc.main.id

    route {
        cidr_block = "0.0.0.0/0"
        gateway_id = aws_internet_gateway.main.id
    }
}

```

```

tags = {
  Name = "${var.project_name}-public-rt"
}
}

# Przypisanie route table do subnetów publicznych
resource "aws_route_table_association" "public" {
  count = length(aws_subnet.public)

  subnet_id      = aws_subnet.public[count.index].id
  route_table_id = aws_route_table.public.id
}

# Data source dla dostępnych stref dostępności
data "aws_availability_zones" "available" {
  state = "available"
}

```

## Definicje zmiennych (variables.tf):

Plain Text

```

variable "aws_region" {
  description = "Region AWS"
  type        = string
  default     = "us-west-2"
}

variable "environment" {
  description = "Środowisko (dev, staging, prod)"
  type        = string
  default     = "dev"
}

variable "project_name" {
  description = "Nazwa projektu"
  type        = string
  default     = "my-project"
}

variable "vpc_cidr" {
  description = "CIDR block dla VPC"
  type        = string
  default     = "10.0.0.0/16"

  validation {
    condition = can(cidrhost(var.vpc_cidr, 0))
  }
}

```



```

    error_message = "VPC CIDR musi być prawidłowym blokiem CIDR."
  }
}

variable "public_subnet_cidrs" {
  description = "Lista CIDR bloków dla subnetów publicznych"
  type        = list(string)
  default     = ["10.0.1.0/24", "10.0.2.0/24"]

  validation {
    condition     = length(var.public_subnet_cidrs) >= 2
    error_message = "Musi być zdefiniowanych co najmniej 2 subnety publiczne."
  }
}

```

## Definicje outputów (outputs.tf):

Plain Text

```

output "vpc_id" {
  description = "ID utworzonego VPC"
  value      = aws_vpc.main.id
}

output "vpc_cidr_block" {
  description = "CIDR block VPC"
  value      = aws_vpc.main.cidr_block
}

output "public_subnet_ids" {
  description = "Lista ID subnetów publicznych"
  value      = aws_subnet.public[*].id
}

output "internet_gateway_id" {
  description = "ID Internet Gateway"
  value      = aws_internet_gateway.main.id
}

output "availability_zones" {
  description = "Lista użytych stref dostępności"
  value      = aws_subnet.public[*].availability_zone
}

```

## Wartości zmiennych (terraform.tfvars):

Plain Text

```
aws_region    = "eu-west-1"
environment   = "development"
project_name  = "webapp-infrastructure"
vpc_cidr      = "10.0.0.0/16"
public_subnet_cidrs = [
  "10.0.1.0/24",
  "10.0.2.0/24",
  "10.0.3.0/24"
]
```

## Wymagania wersji (versions.tf):

Plain Text

```
terraform {
  required_version = ">= 1.0"

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
  }

  # Opcjonalnie: konfiguracja backend dla state
  backend "s3" {
    bucket = "my-terraform-state-bucket"
    key    = "infrastructure/terraform.tfstate"
    region = "eu-west-1"

    # Blokada state za pomocą DynamoDB
    dynamodb_table = "terraform-state-lock"
    encrypt         = true
  }
}
```

## Krok 4: Podstawowe polecenia Terraform

### Inicjalizacja projektu:

Bash

```
# Przejdź do katalogu projektu
cd terraform-project

# Inicjalizuj Terraform (pobierz providery)
terraform init

# Sprawdź składnię konfiguracji
terraform validate

# Sformatuj pliki konfiguracyjne
terraform fmt
```

## Planowanie i wdrażanie:

Bash

```
# Utwórz plan zmian
terraform plan

# Zapisz plan do pliku
terraform plan -out=tfplan

# Zastosuj zmiany
terraform apply

# Zastosuj zapisany plan
terraform apply tfplan

# Zastosuj zmiany bez potwierdzenia (ostrożnie!)
terraform apply -auto-approve
```

## Zarządzanie stanem:

Bash

```
# Wyświetl aktualny stan
terraform show

# Wyświetl listę zasobów w stanie
terraform state list

# Wyświetl szczegóły konkretnego zasobu
terraform state show aws_vpc.main
```

```
# Odśwież stan (synchronizuj z rzeczywistością)
terraform refresh
```

## Usuwanie zasobów:

Bash

```
# Usuń wszystkie zasoby zarządzane przez Terraform
terraform destroy

# Usuń konkretny zasób
terraform destroy -target=aws_instance.example

# Usuń z planem
terraform plan -destroy -out=destroy-plan
terraform apply destroy-plan
```

## Krok 5: Konfiguracja AWS CLI

Przed rozpoczęciem pracy z AWS przez Terraform, należy skonfigurować dostęp do AWS:

Bash

```
# Zainstaluj AWS CLI
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
"awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install

# Skonfiguruj credentials
aws configure
# AWS Access Key ID: [Twój Access Key]
# AWS Secret Access Key: [Twój Secret Key]
# Default region name: eu-west-1
# Default output format: json

# Sprawdź konfigurację
aws sts get-caller-identity
```

## Alternatywnie, użyj zmiennych środowiskowych:

Bash

```
export AWS_ACCESS_KEY_ID="your-access-key"
export AWS_SECRET_ACCESS_KEY="your-secret-key"
export AWS_DEFAULT_REGION="eu-west-1"
```

**Lub użyj AWS IAM Roles (zalecane dla EC2/ECS):**

Bash

```
# Przypisz rolę IAM do instancji EC2
# Terraform automatycznie użyje credentials z roli
```

To kończy wprowadzenie do Infrastructure as Code i podstaw Terraform. W następnym rozdziale omówimy zaawansowane funkcje Terraform i praktyczne przykłady zarządzania infrastrukturą.

## 14. Zaawansowane funkcje Terraform

### Moduły Terraform

Moduły są fundamentalnym mechanizmem organizacji i ponownego wykorzystania kodu w Terraform. Pozwalają na enkapsulację logicznie powiązanych zasobów w wielokrotnie używalne komponenty, co znacząco poprawia czytelność, utrzymywalność i skalowalność infrastruktury jako kodu.

Moduł w Terraform to po prostu katalog zawierający pliki konfiguracyjne `.tf`. Każdy katalog z plikami Terraform może być traktowany jako moduł. Główny katalog projektu (root module) również jest modulem, który może wywoływać inne moduły (child modules).

### Struktura modułów

**Standardowa struktura modułu:**

Plain Text

```
modules/
├── vpc/
│   ├── main.tf          # Główne zasoby modułu
│   └── variables.tf     # Zmienne wejściowe
```

— outputs.tf	# Wartości wyjściowe
— versions.tf	# Wymagania wersji
— README.md	# Dokumentacja modułu

## Przykład modułu VPC (modules/vpc/main.tf):

### Plain Text

```
# Lokalne wartości dla modułu
locals {
  common_tags = {
    Environment = var.environment
    Project      = var.project_name
    ManagedBy    = "Terraform"
    Module       = "vpc"
  }

  # Automatyczne generowanie nazw AZ na podstawie regionu
  azs = slice(data.aws_availability_zones.available.names, 0, var.az_count)
}

# Data source dla dostępnych stref dostępności
data "aws_availability_zones" "available" {
  state = "available"
}

# VPC
resource "aws_vpc" "main" {
  cidr_block           = var.vpc_cidr
  enable_dns_hostnames = var.enable_dns_hostnames
  enable_dns_support   = var.enable_dns_support

  tags = merge(local.common_tags, {
    Name = "${var.project_name}-vpc"
  })
}

# Internet Gateway
resource "aws_internet_gateway" "main" {
  count = var.create_igw ? 1 : 0

  vpc_id = aws_vpc.main.id

  tags = merge(local.common_tags, {
    Name = "${var.project_name}-igw"
  })
}
```

```

# Subnety publiczne
resource "aws_subnet" "public" {
  count = length(var.public_subnet_cidrs)

  vpc_id          = aws_vpc.main.id
  cidr_block      = var.public_subnet_cidrs[count.index]
  availability_zone = local.azs[count.index % length(local.azs)]
  map_public_ip_on_launch = var.map_public_ip_on_launch

  tags = merge(local.common_tags, {
    Name = "${var.project_name}-public-subnet-${count.index + 1}"
    Type = "Public"
    Tier = "Public"
  })
}

# Subnety prywatne
resource "aws_subnet" "private" {
  count = length(var.private_subnet_cidrs)

  vpc_id          = aws_vpc.main.id
  cidr_block      = var.private_subnet_cidrs[count.index]
  availability_zone = local.azs[count.index % length(local.azs)]

  tags = merge(local.common_tags, {
    Name = "${var.project_name}-private-subnet-${count.index + 1}"
    Type = "Private"
    Tier = "Private"
  })
}

# Elastic IP dla NAT Gateway
resource "aws_eip" "nat" {
  count = var.enable_nat_gateway ? (var.single_nat_gateway ? 1 :
length(var.public_subnet_cidrs)) : 0

  domain = "vpc"

  depends_on = [aws_internet_gateway.main]

  tags = merge(local.common_tags, {
    Name = "${var.project_name}-nat-eip-${count.index + 1}"
  })
}

# NAT Gateway
resource "aws_nat_gateway" "main" {

```

```

    count = var.enable_nat_gateway ? (var.single_nat_gateway ? 1 :
length(var.public_subnet_cidrs)) : 0

    allocation_id = aws_eip.nat[count.index].id
    subnet_id      = aws_subnet.public[count.index].id

    tags = merge(local.common_tags, {
        Name = "${var.project_name}-nat-gateway-${count.index + 1}"
    })

    depends_on = [aws_internet_gateway.main]
}

# Route table dla subnetów publicznych
resource "aws_route_table" "public" {
    count = length(var.public_subnet_cidrs) > 0 ? 1 : 0

    vpc_id = aws_vpc.main.id

    tags = merge(local.common_tags, {
        Name = "${var.project_name}-public-rt"
    })
}

# Route do Internet Gateway dla subnetów publicznych
resource "aws_route" "public_internet_gateway" {
    count = var.create_igw && length(var.public_subnet_cidrs) > 0 ? 1 : 0

    route_table_id      = aws_route_table.public[0].id
    destination_cidr_block = "0.0.0.0/0"
    gateway_id          = aws_internet_gateway.main[0].id

    timeouts {
        create = "5m"
    }
}

# Przypisanie route table do subnetów publicznych
resource "aws_route_table_association" "public" {
    count = length(var.public_subnet_cidrs)

    subnet_id      = aws_subnet.public[count.index].id
    route_table_id = aws_route_table.public[0].id
}

# Route tables dla subnetów prywatnych
resource "aws_route_table" "private" {
    count = var.enable_nat_gateway ? (var.single_nat_gateway ? 1 :

```



```

length(var.private_subnet_cidrs)) : length(var.private_subnet_cidrs)

    vpc_id = aws_vpc.main.id

    tags = merge(local.common_tags, {
        Name = var.single_nat_gateway ? "${var.project_name}-private-rt" :
"${var.project_name}-private-rt-${count.index + 1}"
    })
}

# Route do NAT Gateway dla subnetów prywatnych
resource "aws_route" "private_nat_gateway" {
    count = var.enable_nat_gateway ? length(aws_route_table.private) : 0

    route_table_id      = aws_route_table.private[count.index].id
    destination_cidr_block = "0.0.0.0/0"
    nat_gateway_id      = var.single_nat_gateway ?
aws_nat_gateway.main[0].id : aws_nat_gateway.main[count.index].id

    timeouts {
        create = "5m"
    }
}

# Przypisanie route tables do subnetów prywatnych
resource "aws_route_table_association" "private" {
    count = length(var.private_subnet_cidrs)

    subnet_id      = aws_subnet.private[count.index].id
    route_table_id = var.single_nat_gateway ? aws_route_table.private[0].id :
aws_route_table.private[count.index].id
}

# VPC Endpoints dla S3 (opcjonalnie)
resource "aws_vpc_endpoint" "s3" {
    count = var.enable_s3_endpoint ? 1 : 0

    vpc_id      = aws_vpc.main.id
    service_name = "com.amazonaws.${data.aws_region.current.name}.s3"

    tags = merge(local.common_tags, {
        Name = "${var.project_name}-s3-endpoint"
    })
}

# Przypisanie VPC Endpoint do route tables
resource "aws_vpc_endpoint_route_table_association" "s3_public" {
    count = var.enable_s3_endpoint && length(var.public_subnet_cidrs) > 0 ? 1 :

```

0

```
vpc_endpoint_id = aws_vpc_endpoint.s3[0].id
route_table_id  = aws_route_table.public[0].id
}

resource "aws_vpc_endpoint_route_table_association" "s3_private" {
  count = var.enable_s3_endpoint ? length(aws_route_table.private) : 0

  vpc_endpoint_id = aws_vpc_endpoint.s3[0].id
  route_table_id  = aws_route_table.private[count.index].id
}

# Data source dla aktualnego regionu
data "aws_region" "current" {}

# Network ACLs (opcjonalnie)
resource "aws_network_acl" "public" {
  count = var.create_network_acls ? 1 : 0

  vpc_id      = aws_vpc.main.id
  subnet_ids = aws_subnet.public[*].id

  # Reguły inbound
  ingress {
    protocol = -1
    rule_no  = 100
    action   = "allow"
    cidr_block = "0.0.0.0/0"
    from_port = 0
    to_port   = 0
  }

  # Reguły outbound
  egress {
    protocol = -1
    rule_no  = 100
    action   = "allow"
    cidr_block = "0.0.0.0/0"
    from_port = 0
    to_port   = 0
  }

  tags = merge(local.common_tags, {
    Name = "${var.project_name}-public-nacl"
  })
}
```

```

resource "aws_network_acl" "private" {
  count = var.create_network_acls ? 1 : 0

  vpc_id      = aws_vpc.main.id
  subnet_ids = aws_subnet.private[*].id

  # Reguły inbound
  ingress {
    protocol    = -1
    rule_no     = 100
    action      = "allow"
    cidr_block  = var.vpc_cidr
    from_port   = 0
    to_port     = 0
  }

  # Reguły outbound
  egress {
    protocol    = -1
    rule_no     = 100
    action      = "allow"
    cidr_block  = "0.0.0.0/0"
    from_port   = 0
    to_port     = 0
  }

  tags = merge(local.common_tags, {
    Name = "${var.project_name}-private-nacl"
  })
}

```

## Zmienne modułu (modules/vpc/variables.tf):

Plain Text

```

variable "project_name" {
  description = "Nazwa projektu używana w tagach i nazwach zasobów"
  type        = string
}

variable "environment" {
  description = "Środowisko (dev, staging, prod)"
  type        = string
}

variable "vpc_cidr" {
  description = "CIDR block dla VPC"
}

```

```
type          = string

validation {
  condition     = can(cidrhost(var.vpc_cidr, 0))
  error_message = "VPC CIDR musi być prawidłowym blokiem CIDR."
}
}

variable "public_subnet_cidrs" {
  description = "Lista CIDR bloków dla subnetów publicznych"
  type        = list(string)
  default     = []
}

variable "private_subnet_cidrs" {
  description = "Lista CIDR bloków dla subnetów prywatnych"
  type        = list(string)
  default     = []
}

variable "az_count" {
  description = "Liczba stref dostępności do użycia"
  type        = number
  default     = 2

  validation {
    condition     = var.az_count >= 1 && var.az_count <= 6
    error_message = "Liczba stref dostępności musi być między 1 a 6."
  }
}

variable "enable_dns_hostnames" {
  description = "Czy włączyć DNS hostnames w VPC"
  type        = bool
  default     = true
}

variable "enable_dns_support" {
  description = "Czy włączyć DNS support w VPC"
  type        = bool
  default     = true
}

variable "create_igw" {
  description = "Czy utworzyć Internet Gateway"
  type        = bool
  default     = true
}
```

```
variable "map_public_ip_on_launch" {
  description = "Czy automatycznie przypisywać publiczne IP do instancji w subnetach publicznych"
  type        = bool
  default     = true
}

variable "enable_nat_gateway" {
  description = "Czy utworzyć NAT Gateway dla subnetów prywatnych"
  type        = bool
  default     = true
}

variable "single_nat_gateway" {
  description = "Czy użyć pojedynczego NAT Gateway dla wszystkich subnetów prywatnych"
  type        = bool
  default     = false
}

variable "enable_s3_endpoint" {
  description = "Czy utworzyć VPC Endpoint dla S3"
  type        = bool
  default     = false
}

variable "create_network_acls" {
  description = "Czy utworzyć niestandardowe Network ACLs"
  type        = bool
  default     = false
}

variable "additional_tags" {
  description = "Dodatkowe tagi do zastosowania na wszystkich zasobach"
  type        = map(string)
  default     = {}
}
```

## Outputy modułu (modules/vpc/outputs.tf):

Plain Text

```
output "vpc_id" {
  description = "ID utworzonego VPC"
  value       = aws_vpc.main.id
}
```

```

output "vpc_arn" {
  description = "ARN VPC"
  value       = aws_vpc.main.arn
}

output "vpc_cidr_block" {
  description = "CIDR block VPC"
  value       = aws_vpc.main.cidr_block
}

output "internet_gateway_id" {
  description = "ID Internet Gateway"
  value       = var.create_igw ? aws_internet_gateway.main[0].id : null
}

output "public_subnet_ids" {
  description = "Lista ID subnetów publicznych"
  value       = aws_subnet.public[*].id
}

output "private_subnet_ids" {
  description = "Lista ID subnetów prywatnych"
  value       = aws_subnet.private[*].id
}

output "public_subnet_cidrs" {
  description = "Lista CIDR bloków subnetów publicznych"
  value       = aws_subnet.public[*].cidr_block
}

output "private_subnet_cidrs" {
  description = "Lista CIDR bloków subnetów prywatnych"
  value       = aws_subnet.private[*].cidr_block
}

output "public_route_table_id" {
  description = "ID route table dla subnetów publicznych"
  value       = length(aws_route_table.public) > 0 ?
aws_route_table.public[0].id : null
}

output "private_route_table_ids" {
  description = "Lista ID route tables dla subnetów prywatnych"
  value       = aws_route_table.private[*].id
}

output "nat_gateway_ids" {

```

```

    description = "Lista ID NAT Gateways"
    value       = aws_nat_gateway.main[*].id
  }

  output "nat_gateway_public_ips" {
    description = "Lista publicznych IP NAT Gateways"
    value       = aws_eip.nat[*].public_ip
  }

  output "availability_zones" {
    description = "Lista użytych stref dostępności"
    value       = local.azs
  }

  output "s3_vpc_endpoint_id" {
    description = "ID VPC Endpoint dla S3"
    value       = var.enable_s3_endpoint ? aws_vpc_endpoint.s3[0].id : null
  }

```

## Używanie modułów

### Wywołanie modułu w głównej konfiguracji:

Plain Text

```

# main.tf
module "vpc" {
  source = "../modules/vpc"

  project_name = var.project_name
  environment  = var.environment

  vpc_cidr = "10.0.0.0/16"

  public_subnet_cidrs = [
    "10.0.1.0/24",
    "10.0.2.0/24",
    "10.0.3.0/24"
  ]

  private_subnet_cidrs = [
    "10.0.11.0/24",
    "10.0.12.0/24",
    "10.0.13.0/24"
  ]
}

```

```

az_count = 3

enable_nat_gateway = true
single_nat_gateway = false
enable_s3_endpoint = true

additional_tags = {
  Owner      = "DevOps Team"
  CostCenter = "Engineering"
}
}

# Użycie outputów z modułu
resource "aws_security_group" "web" {
  name_prefix = "${var.project_name}-web-"
  vpc_id      = module.vpc.vpc_id

  ingress {
    from_port = 80
    to_port   = 80
    protocol  = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port = 443
    to_port   = 443
    protocol  = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port = 0
    to_port   = 0
    protocol  = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "${var.project_name}-web-sg"
  }
}

```

## Zarządzanie stanem (State Management)



Stan Terraform jest jednym z najważniejszych aspektów pracy z tym narzędziem. Plik state przechowuje mapowanie między konfiguracją Terraform a rzeczywistymi zasobami w infrastrukturze.

## Lokalny state vs. zdalny state

**Lokalny state** jest przechowywany w pliku `terraform.tfstate` w katalogu projektu. Jest to domyślne zachowanie Terraform, odpowiednie dla małych projektów lub pracy indywidualnej. Jednak ma ograniczenia:

- Brak współdzielenia między członkami zespołu
- Brak blokowania równoczesnych operacji
- Ryzyko utraty pliku state
- Trudności z backup i odzyskiwaniem

**Zdalny state** rozwiązuje te problemy poprzez przechowywanie pliku state w zdalnej lokalizacji, takiej jak AWS S3, Azure Storage Account, Google Cloud Storage, czy HashiCorp Consul.

## Konfiguracja zdalnego state w AWS S3

### Krok 1: Przygotowanie infrastruktury dla state

Plain Text

```
# backend-setup.tf
# Ten plik służy do utworzenia infrastruktury dla zdalnego state
# Uruchom go osobno przed konfiguracją głównego projektu

terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
  }
}

provider "aws" {
```

```

    region = var.aws_region
}

# S3 bucket dla state
resource "aws_s3_bucket" "terraform_state" {
    bucket = "${var.project_name}-terraform-
state-${random_id.bucket_suffix.hex}"

    tags = {
        Name          = "Terraform State Bucket"
        Environment    = var.environment
        Purpose        = "TerraformState"
    }
}

# Wersjonowanie dla S3 bucket
resource "aws_s3_bucket_versioning" "terraform_state" {
    bucket = aws_s3_bucket.terraform_state.id
    versioning_configuration {
        status = "Enabled"
    }
}

# Szyfrowanie dla S3 bucket
resource "aws_s3_bucket_server_side_encryption_configuration"
"terraform_state" {
    bucket = aws_s3_bucket.terraform_state.id

    rule {
        apply_server_side_encryption_by_default {
            sse_algorithm = "AES256"
        }
    }
}

# Blokowanie publicznego dostępu
resource "aws_s3_bucket_public_access_block" "terraform_state" {
    bucket = aws_s3_bucket.terraform_state.id

    block_public_acls          = true
    block_public_policy        = true
    ignore_public_acls         = true
    restrict_public_buckets    = true
}

# DynamoDB table dla blokowania state
resource "aws_dynamodb_table" "terraform_state_lock" {
    name = "${var.project_name}-terraform-state-lock"

```

```

billing_mode    = "PAY_PER_REQUEST"
hash_key       = "LockID"

attribute {
  name = "LockID"
  type = "S"
}

tags = {
  Name          = "Terraform State Lock Table"
  Environment    = var.environment
  Purpose        = "TerraformStateLock"
}

# Random ID dla unikalności nazwy bucket
resource "random_id" "bucket_suffix" {
  byte_length = 4
}

# Outputy
output "s3_bucket_name" {
  description = "Nazwa S3 bucket dla Terraform state"
  value       = aws_s3_bucket.terraform_state.bucket
}

output "dynamodb_table_name" {
  description = "Nazwa DynamoDB table dla blokowania state"
  value       = aws_dynamodb_table.terraform_state_lock.name
}

output "s3_bucket_region" {
  description = "Region S3 bucket"
  value       = aws_s3_bucket.terraform_state.region
}

```

## Krok 2: Konfiguracja backend w głównym projekcie

Plain Text

```

# versions.tf
terraform {
  required_version = ">= 1.0"

  required_providers {
    aws = {
      source = "hashicorp/aws"

```

```
    version = "~> 5.0"
  }
}

backend "s3" {
  bucket      = "my-project-terraform-state-a1b2c3d4" # Z outputu
poprzedniego kroku
  key         = "infrastructure/terraform.tfstate"
  region      = "eu-west-1"
  dynamodb_table = "my-project-terraform-state-lock" # Z outputu
poprzedniego kroku
  encrypt     = true

  # Opcjonalnie: dodatkowe zabezpieczenia
  kms_key_id = "arn:aws:kms:eu-west-1:123456789012:key/12345678-1234-1234-
1234-123456789012"
}
}
```

### Krok 3: Migracja lokalnego state do zdalnego

Bash

```
# 1. Dodaj konfigurację backend do versions.tf
# 2. Zainicjalizuj ponownie Terraform
terraform init

# Terraform zapyta czy chcesz skopiować istniejący state
# Odpowiedz "yes"

# 3. Sprawdź czy state został przeniesiony
terraform state list

# 4. Usuń lokalny plik state (opcjonalnie)
rm terraform.tfstate*
```

## Operacje na state

### Podstawowe operacje:

Bash

```
# Wyświetl aktualny state
terraform show
```

```
# Lista wszystkich zasobów w state
terraform state list

# Szczegóły konkretnego zasobu
terraform state show aws_vpc.main

# Przenieś zasób w state (zmiana nazwy)
terraform state mv aws_instance.old_name aws_instance.new_name

# Usuń zasób ze state (bez usuwania rzeczywistego zasobu)
terraform state rm aws_instance.example

# Import istniejącego zasobu do state
terraform import aws_instance.example i-1234567890abcdef0

# Odśwież state (synchronizuj z rzeczywistością)
terraform refresh

# Zastąp zasób (force recreation)
terraform apply -replace=aws_instance.example
```

## Zaawansowane operacje:

Bash

```
# Backup state
terraform state pull > terraform.tfstate.backup

# Przywróć state z backup
terraform state push terraform.tfstate.backup

# Sprawdź różnice między state a rzeczywistością
terraform plan -refresh-only

# Zastosuj tylko zmiany w state (bez modyfikacji zasobów)
terraform apply -refresh-only
```

## Workspace'y

Workspace'y pozwalają na zarządzanie wieloma instancjami tej samej konfiguracji Terraform, co jest przydatne dla różnych środowisk (dev, staging, prod).

Bash

```
# Lista workspace'ów
terraform workspace list

# Utwórz nowy workspace
terraform workspace new development
terraform workspace new staging
terraform workspace new production

# Przełącz na workspace
terraform workspace select development

# Sprawdź aktualny workspace
terraform workspace show

# Usuń workspace
terraform workspace delete development
```

## Użycie workspace'ów w konfiguracji:

Plain Text

```
# Różne konfiguracje dla różnych workspace'ów
locals {
  environment_configs = {
    development = {
      instance_type = "t3.micro"
      min_size      = 1
      max_size      = 2
      vpc_cidr      = "10.0.0.0/16"
    }
    staging = {
      instance_type = "t3.small"
      min_size      = 2
      max_size      = 4
      vpc_cidr      = "10.1.0.0/16"
    }
    production = {
      instance_type = "t3.medium"
      min_size      = 3
      max_size      = 10
      vpc_cidr      = "10.2.0.0/16"
    }
  }
}

current_config = local.environment_configs[terraform.workspace]
}
```

```
# Użycie konfiguracji specyficznej dla workspace
resource "aws_instance" "web" {
  ami           = data.aws_ami.amazon_linux.id
  instance_type = local.current_config.instance_type

  tags = {
    Name          = "${terraform.workspace}-web-server"
    Environment = terraform.workspace
  }
}

# Różne nazwy zasobów dla różnych workspace'ów
resource "aws_s3_bucket" "app_data" {
  bucket = "${var.project_name}-${terraform.workspace}-app-data"

  tags = {
    Environment = terraform.workspace
  }
}
```

## Funkcje i wyrażenia

Terraform oferuje bogaty zestaw wbudowanych funkcji, które pozwalają na dynamiczne generowanie wartości i manipulację danymi w konfiguracji.

### Funkcje stringów

Plain Text

```
locals {
  # Podstawowe operacje na stringach
  project_name = "my-awesome-project"
  environment  = "production"

  # Łączenie stringów
  resource_name = "${local.project_name}-${local.environment}"

  # Formatowanie
  formatted_name = format("%s-%s-%03d", local.project_name,
local.environment, 1)

  # Manipulacja wielkości liter
  upper_name = upper(local.project_name)
```

```

lower_name = lower(local.project_name)
title_name = title(local.project_name)

# Podział i łączenie
name_parts = split("-", local.project_name)
joined_name = join("_", local.name_parts)

# Zastępowanie
sanitized_name = replace(local.project_name, "-", "_")

# Przycinanie
trimmed_name = trim(" ${local.project_name} ", " ")

# Substring
short_name = substr(local.project_name, 0, 10)

# Regex
extracted_name = regex("([a-z]+)-([a-z]+)-([a-z]+)", local.project_name)[1]
}

```

## Funkcje kolekcji

Plain Text

```

locals {
  # Listy
  availability_zones = ["us-west-2a", "us-west-2b", "us-west-2c"]
  instance_types = ["t3.micro", "t3.small", "t3.medium"]

  # Długość listy
  az_count = length(local.availability_zones)

  # Element z listy
  first_az = local.availability_zones[0]
  last_az = local.availability_zones[length(local.availability_zones) - 1]

  # Slice listy
  first_two_azs = slice(local.availability_zones, 0, 2)

  # Konkatenacja list
  all_zones = concat(local.availability_zones, ["us-west-2d"])

  # Unikalne elementy
  unique_types = distinct(concat(local.instance_types, local.instance_types))

  # Sortowanie

```



```

sorted_zones = sort(local.availability_zones)

# Reverse
reversed_zones = reverse(local.availability_zones)

# Contains
has_micro = contains(local.instance_types, "t3.micro")

# Index
micro_index = index(local.instance_types, "t3.micro")

# Mapy
environment_configs = {
  dev = {
    instance_count = 1
    instance_type = "t3.micro"
  }
  prod = {
    instance_count = 3
    instance_type = "t3.large"
  }
}

# Klucze i wartości map
env_names = keys(local.environment_configs)
env_configs = values(local.environment_configs)

# Merge map
merged_config = merge(
  local.environment_configs.dev,
  {
    monitoring_enabled = true
  }
)

# Lookup w mapie
dev_instance_type = lookup(local.environment_configs, "dev",
{}) .instance_type

# Zipmap
az_to_subnet = zipmap(
  local.availability_zones,
  ["subnet-123", "subnet-456", "subnet-789"]
)
}

```

## Funkcje warunkowe i logiczne

## Plain Text

```
locals {
  environment = "production"
  enable_monitoring = true
  instance_count = 3

  # Podstawowe warunki
  is_production = local.environment == "production"
  monitoring_enabled = local.enable_monitoring ? "enabled" : "disabled"

  # Złożone warunki
  instance_type = local.environment == "production" ? "t3.large" : (
    local.environment == "staging" ? "t3.medium" : "t3.micro"
  )

  # Warunki z funkcjami
  backup_retention = local.is_production ? 30 : 7

  # Coalesce (pierwszy nie-null)
  database_name = coalesce(var.custom_db_name, "${var.project_name}-db")

  # Try (obsługa błędów)
  safe_value = try(var.potentially_missing_var, "default_value")

  # Can (sprawdzenie czy wyrażenie jest prawidłowe)
  is_valid_cidr = can(cidrhost(var.vpc_cidr, 0))
}

# Użycie w zasobach
resource "aws_instance" "web" {
  count = local.is_production ? 3 : 1

  ami          = data.aws_ami.amazon_linux.id
  instance_type = local.instance_type

  monitoring = local.enable_monitoring

  tags = merge(
    var.common_tags,
    {
      Name = "${var.project_name}-web-${count.index + 1}"
      Backup = local.is_production ? "required" : "optional"
    }
  )
}
```

## Funkcje dat i czasu

Plain Text

```
locals {
  # Aktualna data i czas
  current_timestamp = timestamp()
  current_date = formatdate("YYYY-MM-DD", timestamp())
  current_time = formatdate("hh:mm:ss", timestamp())

  # Formatowanie dat
  backup_date = formatdate("YYYY-MM-DD-hhmm", timestamp())
  log_timestamp = formatdate("DD/MMM/YYYY:hh:mm:ss ZZZ", timestamp())

  # Dodawanie czasu
  future_date = timeadd(timestamp(), "24h")
  past_date = timeadd(timestamp(), "-7d")

  # Porównywanie dat
  is_weekend = formatdate("EEE", timestamp()) == "Sat" || formatdate("EEE",
timestamp()) == "Sun"
}

# Użycie w tagach
resource "aws_s3_bucket" "backups" {
  bucket = "${var.project_name}-backups-${local.current_date}"

  tags = {
    CreatedDate = local.current_timestamp
    BackupDate  = local.backup_date
  }
}
```

## Funkcje kryptograficzne

Plain Text

```
locals {
  # Generowanie hashy
  config_hash = md5(jsonencode(var.application_config))
  file_hash = filemd5("${path.module}/config.json")

  # SHA
  secure_hash = sha256(var.sensitive_data)
  file_sha = filesha256("${path.module}/script.sh")
}
```

```

# Base64
encoded_config = base64encode(jsonencode(var.application_config))
decoded_config = jsondecode(base64decode(var.encoded_config))

# UUID
unique_id = uuid()

# Random
random_suffix = substr(uuid(), 0, 8)
}

# Użycie do generowania unikalnych nazw
resource "aws_s3_bucket" "unique" {
  bucket = "${var.project_name}-${local.random_suffix}"

  tags = {
    ConfigHash = local.config_hash
  }
}

```

## For expressions

For expressions pozwalają na transformację kolekcji danych w Terraform:

Plain Text

```

locals {
  # Podstawowe for expression
  availability_zones = ["us-west-2a", "us-west-2b", "us-west-2c"]

  # Tworzenie listy
  subnet_names = [
    for az in local.availability_zones : "${var.project_name}-subnet-${az}"
  ]

  # Tworzenie mapy
  az_to_subnet_name = {
    for az in local.availability_zones : az => "${var.project_name}-subnet-${az}"
  }

  # Z warunkiem
  production_azs = [
    for az in local.availability_zones : az
    if var.environment == "production" || az != "us-west-2c"
  ]
}

```

```

]

# Transformacja map
environments = {
  dev = {
    instance_type = "t3.micro"
    instance_count = 1
  }
  prod = {
    instance_type = "t3.large"
    instance_count = 3
  }
}

# Wyciągnięcie wartości z map
instance_types = [
  for env, config in local.environments : config.instance_type
]

# Tworzenie nowej mapy z transformacją
environment_tags = {
  for env, config in local.environments : env => {
    InstanceType = config.instance_type
    Environment = upper(env)
    Monitoring = config.instance_count > 1 ? "enabled" : "disabled"
  }
}

# Złożone transformacje
subnet_configs = [
  for i, az in local.availability_zones : {
    name = "${var.project_name}-subnet-${i + 1}"
    availability_zone = az
    cidr_block = cidrsubnet(var.vpc_cidr, 8, i + 1)
    public = i < 2 # Pierwsze dwa subnety publiczne
  }
]

# Użycie w zasobach
resource "aws_subnet" "main" {
  for_each = {
    for subnet in local.subnet_configs : subnet.name => subnet
  }

  vpc_id = aws_vpc.main.id
  cidr_block = each.value.cidr_block
  availability_zone = each.value.availability_zone

```

```
map_public_ip_on_launch = each.value.public

tags = {
  Name = each.value.name
  Type = each.value.public ? "Public" : "Private"
}
}
```

To kończy rozdział o zaawansowanych funkcjach Terraform. W następnym rozdziale omówimy praktyczne przykłady zarządzania infrastrukturą AWS z użyciem Terraform.

## 15. Praktyczne przykłady Terraform z AWS

### Kompletna infrastruktura aplikacji webowej

W tym rozdziale zbudujemy kompletną infrastrukturę dla aplikacji webowej w AWS, która będzie obejmować VPC, Load Balancer, Auto Scaling Group, RDS, ElastiCache, S3 i CloudFront. Ten przykład pokazuje, jak Terraform może być używany do zarządzania złożoną infrastrukturą produkcyjną.

### Struktura projektu

Plain Text

```
terraform-aws-webapp/
├─ main.tf                # Główna konfiguracja
├─ variables.tf           # Definicje zmiennych
├─ outputs.tf            # Outputy
├─ versions.tf           # Wymagania wersji i backend
├─ terraform.tfvars      # Wartości zmiennych
├─ data.tf               # Data sources
├─ locals.tf            # Lokalne wartości
├─ modules/
│   └─ vpc/              # Moduł VPC (z poprzedniego rozdziału)
│   └─ security-groups/  # Moduł Security Groups
│   └─ alb/              # Moduł Application Load Balancer
│   └─ asg/              # Moduł Auto Scaling Group
│   └─ rds/              # Moduł RDS
│   └─ s3/               # Moduł S3
├─ environments/
│   └─ dev/
```

```

├── terraform.tfvars
├── backend.tf
├── staging/
│   ├── terraform.tfvars
│   └── backend.tf
├── prod/
│   ├── terraform.tfvars
│   └── backend.tf
└── scripts/
    ├── user-data.sh      # User data script dla EC2
    └── deploy.sh         # Skrypt deployment

```

## Główna konfiguracja

### versions.tf:

Plain Text

```

terraform {
  required_version = ">= 1.0"

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
    random = {
      source  = "hashicorp/random"
      version = "~> 3.1"
    }
    tls = {
      source  = "hashicorp/tls"
      version = "~> 4.0"
    }
  }
}

backend "s3" {
  # Konfiguracja będzie w plikach backend.tf w katalogach environments/
}

provider "aws" {
  region = var.aws_region

  default_tags {
    tags = local.common_tags
  }
}

```

```
}  
}
```

## locals.tf:

### Plain Text

```
locals {  
  # Wspólne tagi dla wszystkich zasobów  
  common_tags = {  
    Project      = var.project_name  
    Environment  = var.environment  
    ManagedBy   = "Terraform"  
    Owner       = var.owner  
    CostCenter   = var.cost_center  
    CreatedDate = formatdate("YYYY-MM-DD", timestamp())  
  }  
  
  # Nazwy zasobów z prefiksem  
  name_prefix = "${var.project_name}-${var.environment}"  
  
  # Konfiguracje specyficzne dla środowiska  
  environment_config = {  
    dev = {  
      instance_type      = "t3.micro"  
      min_size           = 1  
      max_size           = 2  
      desired_capacity    = 1  
      db_instance_class  = "db.t3.micro"  
      db_allocated_storage = 20  
      cache_node_type     = "cache.t3.micro"  
      cache_num_nodes     = 1  
      enable_deletion_protection = false  
      backup_retention_period = 7  
    }  
    staging = {  
      instance_type      = "t3.small"  
      min_size           = 2  
      max_size           = 4  
      desired_capacity    = 2  
      db_instance_class  = "db.t3.small"  
      db_allocated_storage = 50  
      cache_node_type     = "cache.t3.small"  
      cache_num_nodes     = 2  
      enable_deletion_protection = false  
      backup_retention_period = 14  
    }  
  }  
}
```



```

prod = {
    instance_type      = "t3.medium"
    min_size           = 3
    max_size           = 10
    desired_capacity    = 3
    db_instance_class  = "db.t3.medium"
    db_allocated_storage = 100
    cache_node_type     = "cache.t3.medium"
    cache_num_nodes     = 3
    enable_deletion_protection = true
    backup_retention_period = 30
}
}

current_config = local.environment_config[var.environment]

# Availability Zones
azs = slice(data.aws_availability_zones.available.names, 0, var.az_count)

# Subnety
public_subnet_cidrs = [
    for i in range(var.az_count) : cidrsubnet(var.vpc_cidr, 8, i + 1)
]

private_subnet_cidrs = [
    for i in range(var.az_count) : cidrsubnet(var.vpc_cidr, 8, i + 10)
]

database_subnet_cidrs = [
    for i in range(var.az_count) : cidrsubnet(var.vpc_cidr, 8, i + 20)
]
}

```

## data.tf:

### Plain Text

```

# Dostępne strefy dostępności
data "aws_availability_zones" "available" {
    state = "available"
}

# Najnowszy Amazon Linux 2 AMI
data "aws_ami" "amazon_linux" {
    most_recent = true
    owners      = ["amazon"]
}

```

```

filter {
  name    = "name"
  values  = ["amzn2-ami-hvm-*-x86_64-gp2"]
}

filter {
  name    = "virtualization-type"
  values  = ["hvm"]
}
}

# Aktualny region
data "aws_region" "current" {}

# Aktualny caller identity
data "aws_caller_identity" "current" {}

# Route53 hosted zone (jeśli istnieje)
data "aws_route53_zone" "main" {
  count = var.domain_name != "" ? 1 : 0
  name  = var.domain_name
}

```

## variables.tf:

### Plain Text

```

variable "aws_region" {
  description = "Region AWS"
  type        = string
  default     = "eu-west-1"
}

variable "project_name" {
  description = "Nazwa projektu"
  type        = string

  validation {
    condition     = can(regex("^[a-z0-9-]+$", var.project_name))
    error_message = "Nazwa projektu może zawierać tylko małe litery, cyfry i myślniki."
  }
}

variable "environment" {
  description = "Środowisko (dev, staging, prod)"
  type        = string
}

```

```
validation {
    condition      = contains(["dev", "staging", "prod"], var.environment)
    error_message = "Środowisko musi być jednym z: dev, staging, prod."
}

variable "owner" {
    description = "Właściciel zasobów"
    type        = string
    default     = "DevOps Team"
}

variable "cost_center" {
    description = "Centrum kosztów"
    type        = string
    default     = "Engineering"
}

variable "vpc_cidr" {
    description = "CIDR block dla VPC"
    type        = string
    default     = "10.0.0.0/16"

    validation {
        condition      = can(cidrhost(var.vpc_cidr, 0))
        error_message = "VPC CIDR musi być prawidłowym blokiem CIDR."
    }
}

variable "az_count" {
    description = "Liczba stref dostępności"
    type        = number
    default     = 3

    validation {
        condition      = var.az_count >= 2 && var.az_count <= 6
        error_message = "Liczba stref dostępności musi być między 2 a 6."
    }
}

variable "domain_name" {
    description = "Nazwa domeny (opcjonalna)"
    type        = string
    default     = ""
}

variable "ssl_certificate_arn" {
```

```

    description = "ARN certyfikatu SSL (opcjonalny)"
    type        = string
    default     = ""
  }

  variable "db_username" {
    description = "Nazwa użytkownika bazy danych"
    type        = string
    default     = "admin"
    sensitive   = true
  }

  variable "db_password" {
    description = "Hasło bazy danych"
    type        = string
    sensitive   = true

    validation {
      condition     = length(var.db_password) >= 8
      error_message = "Hasło bazy danych musi mieć co najmniej 8 znaków."
    }
  }

  variable "enable_monitoring" {
    description = "Czy włączyć szczegółowe monitorowanie"
    type        = bool
    default     = true
  }

  variable "enable_backup" {
    description = "Czy włączyć automatyczne backup"
    type        = bool
    default     = true
  }

  variable "allowed_cidr_blocks" {
    description = "Lista CIDR bloków z dostępem do aplikacji"
    type        = list(string)
    default     = ["0.0.0.0/0"]
  }

```

## main.tf:

Plain Text

```

# VPC
module "vpc" {

```

```

source = "./modules/vpc"

project_name = var.project_name
environment  = var.environment

vpc_cidr      = var.vpc_cidr
public_subnet_cidrs = local.public_subnet_cidrs
private_subnet_cidrs = local.private_subnet_cidrs

az_count = var.az_count

enable_nat_gateway = true
single_nat_gateway = var.environment == "dev" ? true : false
enable_s3_endpoint = true

additional_tags = local.common_tags
}

# Database subnets
resource "aws_subnet" "database" {
  count = var.az_count

  vpc_id      = module.vpc.vpc_id
  cidr_block  = local.database_subnet_cidrs[count.index]
  availability_zone = local.azs[count.index]

  tags = merge(local.common_tags, {
    Name = "${local.name_prefix}-database-subnet-${count.index + 1}"
    Type = "Database"
  })
}

# Database subnet group
resource "aws_db_subnet_group" "main" {
  name          = "${local.name_prefix}-db-subnet-group"
  subnet_ids    = aws_subnet.database[*].id

  tags = merge(local.common_tags, {
    Name = "${local.name_prefix}-db-subnet-group"
  })
}

# Security Groups
module "security_groups" {
  source = "./modules/security-groups"

  project_name = var.project_name
  environment  = var.environment

```

```

vpc_id          = module.vpc.vpc_id
vpc_cidr        = var.vpc_cidr
allowed_cidr_blocks = var.allowed_cidr_blocks

tags = local.common_tags
}

# Application Load Balancer
module "alb" {
  source = "../modules/alb"

  project_name = var.project_name
  environment  = var.environment

  vpc_id          = module.vpc.vpc_id
  public_subnet_ids = module.vpc.public_subnet_ids
  security_group_ids = [module.security_groups.alb_security_group_id]

  ssl_certificate_arn = var.ssl_certificate_arn

  tags = local.common_tags
}

# Key Pair dla EC2
resource "tls_private_key" "main" {
  algorithm = "RSA"
  rsa_bits  = 4096
}

resource "aws_key_pair" "main" {
  key_name    = "${local.name_prefix}-key"
  public_key = tls_private_key.main.public_key_openssh

  tags = local.common_tags
}

# Auto Scaling Group
module "asg" {
  source = "../modules/asg"

  project_name = var.project_name
  environment  = var.environment

  vpc_id          = module.vpc.vpc_id
  private_subnet_ids = module.vpc.private_subnet_ids
  security_group_ids = [module.security_groups.web_security_group_id]

```

```

ami_id            = data.aws_ami.amazon_linux.id
instance_type     = local.current_config.instance_type
key_name          = aws_key_pair.main.key_name

min_size          = local.current_config.min_size
max_size          = local.current_config.max_size
desired_capacity  = local.current_config.desired_capacity

target_group_arn  = module.alb.target_group_arn

user_data_script  = file("${path.module}/scripts/user-data.sh")

enable_monitoring = var.enable_monitoring

tags = local.common_tags
}

# RDS Database
module "rds" {
    source = "../modules/rds"

    project_name = var.project_name
    environment  = var.environment

    vpc_id            = module.vpc.vpc_id
    db_subnet_group_name = aws_db_subnet_group.main.name
    security_group_ids = [module.security_groups.rds_security_group_id]

    db_instance_class      = local.current_config.db_instance_class
    allocated_storage       = local.current_config.db_allocated_storage

    db_name      = replace(var.project_name, "-", "")
    db_username  = var.db_username
    db_password  = var.db_password

    backup_retention_period = local.current_config.backup_retention_period
    enable_deletion_protection =
local.current_config.enable_deletion_protection

    enable_monitoring = var.enable_monitoring

    tags = local.common_tags
}

# ElastiCache Redis
resource "aws_elasticache_subnet_group" "main" {
    name      = "${local.name_prefix}-cache-subnet-group"
    subnet_ids = module.vpc.private_subnet_ids

```

```

    tags = local.common_tags
}

resource "aws_elasticache_replication_group" "main" {
    replication_group_id      = "${local.name_prefix}-redis"
    description               = "Redis cluster for ${var.project_name}
${var.environment}"

    node_type                 = local.current_config.cache_node_type
    port                     = 6379
    parameter_group_name     = "default.redis7"

    num_cache_clusters       = local.current_config.cache_num_nodes

    subnet_group_name        = aws_elasticache_subnet_group.main.name
    security_group_ids       =
[module.security_groups.cache_security_group_id]

    at_rest_encryption_enabled = true
    transit_encryption_enabled = true

    automatic_failover_enabled = local.current_config.cache_num_nodes > 1
    multi_az_enabled           = local.current_config.cache_num_nodes > 1

    snapshot_retention_limit   = var.enable_backup ? 5 : 0
    snapshot_window            = "03:00-05:00"
    maintenance_window         = "sun:05:00-sun:07:00"

    tags = local.common_tags
}

# S3 Buckets
module "s3" {
    source = "../modules/s3"

    project_name = var.project_name
    environment  = var.environment

    enable_versioning = var.environment == "prod"
    enable_encryption = true

    tags = local.common_tags
}

# CloudFront Distribution
resource "aws_cloudfront_origin_access_control" "main" {
    name = "${local.name_prefix}-oac"

```



```

    description                        = "OAC for ${var.project_name}
    ${var.environment}"
    origin_access_control_origin_type = "s3"
    signing_behavior                   = "always"
    signing_protocol                   = "sigv4"
}

resource "aws_cloudfront_distribution" "main" {
  origin {
    domain_name      = module.alb.dns_name
    origin_id        = "ALB-${local.name_prefix}"

    custom_origin_config {
      http_port      = 80
      https_port     = 443
      origin_protocol_policy = var.ssl_certificate_arn != "" ? "https-only" :
"http-only"
      origin_ssl_protocols = ["TLSv1.2"]
    }
  }

  origin {
    domain_name      = module.s3.static_bucket_domain_name
    origin_id        = "S3-${local.name_prefix}"
    origin_access_control_id = aws_cloudfront_origin_access_control.main.id
  }

  enabled          = true
  is_ipv6_enabled  = true
  comment          = "CloudFront distribution for ${var.project_name}
    ${var.environment}"
  default_root_object = "index.html"

  # Cache behavior dla aplikacji
  default_cache_behavior {
    allowed_methods      = ["DELETE", "GET", "HEAD", "OPTIONS", "PATCH",
"POST", "PUT"]
    cached_methods      = ["GET", "HEAD"]
    target_origin_id     = "ALB-${local.name_prefix}"
    compress             = true
    viewer_protocol_policy = "redirect-to-https"

    forwarded_values {
      query_string = true
      headers      = ["Host", "CloudFront-Forwarded-Proto"]

      cookies {
        forward = "all"
      }
    }
  }
}

```

```

    }
}

min_ttl      = 0
default_ttl  = 3600
max_ttl      = 86400
}

# Cache behavior dla statycznych plików
ordered_cache_behavior {
  path_pattern          = "/static/*"
  allowed_methods       = ["GET", "HEAD", "OPTIONS"]
  cached_methods        = ["GET", "HEAD"]
  target_origin_id      = "S3-`${local.name_prefix}`"
  compress              = true
  viewer_protocol_policy = "redirect-to-https"

  forwarded_values {
    query_string = false

    cookies {
      forward = "none"
    }
  }
}

min_ttl      = 0
default_ttl  = 86400
max_ttl      = 31536000
}

price_class = var.environment == "prod" ? "PriceClass_All" :
"PriceClass_100"

restrictions {
  geo_restriction {
    restriction_type = "none"
  }
}

viewer_certificate {
  cloudfront_default_certificate = var.ssl_certificate_arn == ""
  acm_certificate_arn           = var.ssl_certificate_arn != "" ?
var.ssl_certificate_arn : null
  ssl_support_method            = var.ssl_certificate_arn != "" ? "sni-
only" : null
  minimum_protocol_version      = var.ssl_certificate_arn != "" ?
"TLSv1.2_2021" : null
}

```

```

tags = local.common_tags
}

# Route53 Records (jeśli domena jest skonfigurowana)
resource "aws_route53_record" "main" {
  count = var.domain_name != "" ? 1 : 0

  zone_id = data.aws_route53_zone.main[0].zone_id
  name     = var.environment == "prod" ? var.domain_name :
"${var.environment}.${var.domain_name}"
  type     = "A"

  alias {
    name           = aws_cloudfront_distribution.main.domain_name
    zone_id        = aws_cloudfront_distribution.main.hosted_zone_id
    evaluate_target_health = false
  }
}

# CloudWatch Alarms
resource "aws_cloudwatch_metric_alarm" "high_cpu" {
  count = var.enable_monitoring ? 1 : 0

  alarm_name           = "${local.name_prefix}-high-cpu"
  comparison_operator = "GreaterThanThreshold"
  evaluation_periods   = "2"
  metric_name          = "CPUUtilization"
  namespace            = "AWS/EC2"
  period              = "300"
  statistic            = "Average"
  threshold            = "80"
  alarm_description    = "This metric monitors ec2 cpu utilization"
  alarm_actions        = [aws_sns_topic.alerts[0].arn]

  dimensions = {
    AutoScalingGroupName = module.asg.asg_name
  }

  tags = local.common_tags
}

resource "aws_cloudwatch_metric_alarm" "database_cpu" {
  count = var.enable_monitoring ? 1 : 0

  alarm_name           = "${local.name_prefix}-database-high-cpu"
  comparison_operator = "GreaterThanThreshold"
  evaluation_periods   = "2"

```

```

metric_name      = "CPUUtilization"
namespace        = "AWS/RDS"
period           = "300"
statistic        = "Average"
threshold        = "80"
alarm_description = "This metric monitors RDS cpu utilization"
alarm_actions     = [aws_sns_topic.alerts[0].arn]

dimensions = {
    DBInstanceIdentifier = module.rds.db_instance_id
}

tags = local.common_tags
}

# SNS Topic dla alertów
resource "aws_sns_topic" "alerts" {
    count = var.enable_monitoring ? 1 : 0

    name = "${local.name_prefix}-alerts"

    tags = local.common_tags
}

# IAM Role dla EC2 instances
resource "aws_iam_role" "ec2_role" {
    name = "${local.name_prefix}-ec2-role"

    assume_role_policy = jsonencode({
        Version = "2012-10-17"
        Statement = [
            {
                Action = "sts:AssumeRole"
                Effect = "Allow"
                Principal = {
                    Service = "ec2.amazonaws.com"
                }
            }
        ]
    })

    tags = local.common_tags
}

resource "aws_iam_role_policy" "ec2_policy" {
    name = "${local.name_prefix}-ec2-policy"
    role = aws_iam_role.ec2_role.id

```

```

policy = jsonencode({
  Version = "2012-10-17"
  Statement = [
    {
      Effect = "Allow"
      Action = [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject"
      ]
      Resource = [
        "${module.s3.app_bucket_arn}/*"
      ]
    },
    {
      Effect = "Allow"
      Action = [
        "s3:ListBucket"
      ]
      Resource = [
        module.s3.app_bucket_arn
      ]
    },
    {
      Effect = "Allow"
      Action = [
        "cloudwatch:PutMetricData",
        "ec2:DescribeVolumes",
        "ec2:DescribeTags",
        "logs:PutLogEvents",
        "logs:CreateLogGroup",
        "logs:CreateLogStream"
      ]
      Resource = "*"
    }
  ]
})

resource "aws_iam_instance_profile" "ec2_profile" {
  name = "${local.name_prefix}-ec2-profile"
  role = aws_iam_role.ec2_role.name

  tags = local.common_tags
}

```

## Moduły pomocnicze

## modules/security-groups/main.tf:

Plain Text

```
# Security Group dla ALB
resource "aws_security_group" "alb" {
  name_prefix = "${var.project_name}-${var.environment}-alb-"
  vpc_id      = var.vpc_id

  ingress {
    description = "HTTP"
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = var.allowed_cidr_blocks
  }

  ingress {
    description = "HTTPS"
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = var.allowed_cidr_blocks
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = merge(var.tags, {
    Name = "${var.project_name}-${var.environment}-alb-sg"
  })

  lifecycle {
    create_before_destroy = true
  }
}

# Security Group dla Web Servers
resource "aws_security_group" "web" {
  name_prefix = "${var.project_name}-${var.environment}-web-"
  vpc_id      = var.vpc_id

  ingress {
```

```

    description      = "HTTP from ALB"
    from_port        = 80
    to_port          = 80
    protocol          = "tcp"
    security_groups   = [aws_security_group.alb.id]
}

ingress {
    description = "SSH"
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = [var.vpc_cidr]
}

egress {
    from_port = 0
    to_port   = 0
    protocol  = "-1"
    cidr_blocks = ["0.0.0.0/0"]
}

tags = merge(var.tags, {
    Name = "${var.project_name}-${var.environment}-web-sg"
})

lifecycle {
    create_before_destroy = true
}
}

# Security Group dla RDS
resource "aws_security_group" "rds" {
    name_prefix = "${var.project_name}-${var.environment}-rds-"
    vpc_id      = var.vpc_id

    ingress {
        description      = "MySQL/Aurora"
        from_port        = 3306
        to_port          = 3306
        protocol          = "tcp"
        security_groups   = [aws_security_group.web.id]
    }

    tags = merge(var.tags, {
        Name = "${var.project_name}-${var.environment}-rds-sg"
    })
}

```

```

    lifecycle {
      create_before_destroy = true
    }
  }

# Security Group dla ElastiCache
resource "aws_security_group" "cache" {
  name_prefix = "${var.project_name}-${var.environment}-cache-"
  vpc_id      = var.vpc_id

  ingress {
    description      = "Redis"
    from_port        = 6379
    to_port          = 6379
    protocol          = "tcp"
    security_groups   = [aws_security_group.web.id]
  }

  tags = merge(var.tags, {
    Name = "${var.project_name}-${var.environment}-cache-sg"
  })

  lifecycle {
    create_before_destroy = true
  }
}

```

## scripts/user-data.sh:

Bash

```

#!/bin/bash

# Update system
yum update -y

# Install required packages
yum install -y \
  docker \
  amazon-cloudwatch-agent \
  awscli \
  htop \
  git

# Start and enable Docker
systemctl start docker
systemctl enable docker

```



```
# Add ec2-user to docker group
usermod -a -G docker ec2-user

# Install Docker Compose
curl -L "https://github.com/docker/compose/releases/latest/download/docker-
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
chmod +x /usr/local/bin/docker-compose

# Create application directory
mkdir -p /opt/app
cd /opt/app

# Create a simple web application
cat > docker-compose.yml << 'EOF'
version: '3.8'
services:
  web:
    image: nginx:alpine
    ports:
      - "80:80"
    volumes:
      - ./html:/usr/share/nginx/html
    restart: unless-stopped
EOF

# Create HTML content
mkdir -p html
cat > html/index.html << 'EOF'
<!DOCTYPE html>
<html>
<head>
  <title>My Web Application</title>
  <style>
    body { font-family: Arial, sans-serif; margin: 40px; }
    .container { max-width: 800px; margin: 0 auto; }
    .header { background: #f4f4f4; padding: 20px; border-radius: 5px; }
    .info { margin: 20px 0; padding: 15px; background: #e9e9e9; border-
radius: 5px; }
  </style>
</head>
<body>
  <div class="container">
    <div class="header">
      <h1>Welcome to My Web Application</h1>
      <p>This is a sample application deployed with Terraform on
AWS</p>
    </div>
```

```

    <div class="info">
        <h2>Instance Information</h2>
        <p><strong>Instance ID:</strong> <span id="instance-
id">Loading...</span></p>
        <p><strong>Availability Zone:</strong> <span id="az">Loading...
</span></p>
        <p><strong>Region:</strong> <span id="region">Loading...</span>
</p>
        <p><strong>Timestamp:</strong> <span id="timestamp"></span></p>
    </div>

    <div class="info">
        <h2>Application Status</h2>
        <p>✔ Web Server: Running</p>
        <p>✔ Database: Connected</p>
        <p>✔ Cache: Available</p>
        <p>✔ Load Balancer: Healthy</p>
    </div>
</div>

<script>
    // Get instance metadata
    fetch('/latest/meta-data/instance-id')
        .then(response => response.text())
        .then(data => document.getElementById('instance-id').textContent
= data)
        .catch(() => document.getElementById('instance-id').textContent =
'N/A');

    fetch('/latest/meta-data/placement/availability-zone')
        .then(response => response.text())
        .then(data => document.getElementById('az').textContent = data)
        .catch(() => document.getElementById('az').textContent = 'N/A');

    fetch('/latest/meta-data/placement/region')
        .then(response => response.text())
        .then(data => document.getElementById('region').textContent =
data)
        .catch(() => document.getElementById('region').textContent =
'N/A');

    document.getElementById('timestamp').textContent = new
Date().toLocaleString();
</script>
</body>
</html>
EOF

```

```
# Start the application
docker-compose up -d

# Configure CloudWatch agent
cat > /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json <<
'EOF'
{
  "metrics": {
    "namespace": "MyApp/EC2",
    "metrics_collected": {
      "cpu": {
        "measurement": [
          "cpu_usage_idle",
          "cpu_usage_iowait",
          "cpu_usage_user",
          "cpu_usage_system"
        ],
        "metrics_collection_interval": 60
      },
      "disk": {
        "measurement": [
          "used_percent"
        ],
        "metrics_collection_interval": 60,
        "resources": [
          "*"
        ]
      },
      "diskio": {
        "measurement": [
          "io_time"
        ],
        "metrics_collection_interval": 60,
        "resources": [
          "*"
        ]
      },
      "mem": {
        "measurement": [
          "mem_used_percent"
        ],
        "metrics_collection_interval": 60
      }
    }
  },
  "logs": {
    "logs_collected": {
```

```

        "files": {
            "collect_list": [
                {
                    "file_path": "/var/log/messages",
                    "log_group_name": "/aws/ec2/var/log/messages",
                    "log_stream_name": "{instance_id}"
                },
                {
                    "file_path": "/var/log/docker",
                    "log_group_name": "/aws/ec2/docker",
                    "log_stream_name": "{instance_id}"
                }
            ]
        }
    }
}
EOF

# Start CloudWatch agent
/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl \
    -a fetch-config \
    -m ec2 \
    -c file:/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json \
    -s

# Create a health check endpoint
cat > /opt/app/html/health << 'EOF'
OK
EOF

echo "User data script completed successfully" >> /var/log/user-data.log

```

## Pliki konfiguracyjne środowisk

### environments/prod/terraform.tfvars:

Plain Text

```

# Podstawowe ustawienia
aws_region    = "eu-west-1"
project_name  = "webapp"
environment   = "prod"
owner         = "DevOps Team"
cost_center   = "Engineering"

```

```

# Sieć
vpc_cidr = "10.0.0.0/16"
az_count = 3

# Domena i SSL
domain_name      = "example.com"
ssl_certificate_arn = "arn:aws:acm:eu-west-1:123456789012:certificate/12345678-1234-1234-1234-123456789012"

# Baza danych
db_username = "admin"
# db_password będzie ustawione przez zmienną środowiskową TF_VAR_db_password

# Monitorowanie i backup
enable_monitoring = true
enable_backup     = true

# Bezpieczeństwo
allowed_cidr_blocks = [
  "10.0.0.0/8",    # Internal networks
  "172.16.0.0/12", # Internal networks
  "192.168.0.0/16" # Internal networks
]

```

## environments/prod/backend.tf:

Plain Text

```

terraform {
  backend "s3" {
    bucket      = "webapp-terraform-state-prod"
    key         = "infrastructure/terraform.tfstate"
    region     = "eu-west-1"
    dynamodb_table = "webapp-terraform-state-lock-prod"
    encrypt     = true
  }
}

```

## Outputs

### outputs.tf:

Plain Text

```
output "vpc_id" {
  description = "ID VPC"
  value       = module.vpc.vpc_id
}

output "alb_dns_name" {
  description = "DNS name Load Balancera"
  value       = module.alb.dns_name
}

output "alb_zone_id" {
  description = "Zone ID Load Balancera"
  value       = module.alb.zone_id
}

output "cloudfront_domain_name" {
  description = "Domain name CloudFront distribution"
  value       = aws_cloudfront_distribution.main.domain_name
}

output "cloudfront_distribution_id" {
  description = "ID CloudFront distribution"
  value       = aws_cloudfront_distribution.main.id
}

output "database_endpoint" {
  description = "Endpoint bazy danych RDS"
  value       = module.rds.db_instance_endpoint
  sensitive   = true
}

output "redis_endpoint" {
  description = "Endpoint Redis cluster"
  value       =
aws_elasticache_replication_group.main.primary_endpoint_address
  sensitive   = true
}

output "s3_app_bucket" {
  description = "Nazwa S3 bucket dla aplikacji"
  value       = module.s3.app_bucket_name
}

output "s3_static_bucket" {
  description = "Nazwa S3 bucket dla plików statycznych"
  value       = module.s3.static_bucket_name
}
```

```

output "application_url" {
  description = "URL aplikacji"
  value = var.domain_name != "" ? (
    var.environment == "prod" ?
      "https://${var.domain_name}" :
      "https://${var.environment}.${var.domain_name}"
    ) : "https://${aws_cloudfront_distribution.main.domain_name}"
}

output "ssh_private_key" {
  description = "Prywatny klucz SSH (zapisz w bezpiecznym miejscu)"
  value       = tls_private_key.main.private_key_pem
  sensitive   = true
}

output "asg_name" {
  description = "Nazwa Auto Scaling Group"
  value       = module.asg.asg_name
}

output "security_groups" {
  description = "ID Security Groups"
  value = {
    alb   = module.security_groups.alb_security_group_id
    web   = module.security_groups.web_security_group_id
    rds   = module.security_groups.rds_security_group_id
    cache = module.security_groups.cache_security_group_id
  }
}

```

## Deployment i zarządzanie

### scripts/deploy.sh:

Bash

```

#!/bin/bash

set -e

# Kolory dla outputu
RED='\033[0;31m'
GREEN='\033[0;32m'
YELLOW='\033[1;33m'
NC='\033[0m' # No Color

```

```
# Funkcje pomocnicze
log_info() {
    echo -e "${GREEN}[INFO]${NC} $1"
}

log_warn() {
    echo -e "${YELLOW}[WARN]${NC} $1"
}

log_error() {
    echo -e "${RED}[ERROR]${NC} $1"
}

# Sprawdź argumenty
if [ $# -ne 1 ]; then
    log_error "Usage: $0 <environment>"
    log_error "Available environments: dev, staging, prod"
    exit 1
fi

ENVIRONMENT=$1
SCRIPT_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
PROJECT_DIR="$(dirname "$SCRIPT_DIR)"
ENV_DIR="$PROJECT_DIR/environments/$ENVIRONMENT"

# Sprawdź czy środowisko istnieje
if [ ! -d "$ENV_DIR" ]; then
    log_error "Environment directory not found: $ENV_DIR"
    exit 1
fi

log_info "Deploying to environment: $ENVIRONMENT"

# Przejdź do katalogu projektu
cd "$PROJECT_DIR"

# Sprawdź czy Terraform jest zainstalowany
if ! command -v terraform &> /dev/null; then
    log_error "Terraform is not installed"
    exit 1
fi

# Sprawdź czy AWS CLI jest skonfigurowane
if ! aws sts get-caller-identity &> /dev/null; then
    log_error "AWS CLI is not configured or credentials are invalid"
    exit 1
fi
```



```
# Skopiuj konfigurację środowiska
log_info "Copying environment configuration..."
cp "$ENV_DIR/terraform.tfvars" .
if [ -f "$ENV_DIR/backend.tf" ]; then
    cp "$ENV_DIR/backend.tf" .
fi

# Sprawdź czy hasło bazy danych jest ustawione
if [ -z "$TF_VAR_db_password" ]; then
    log_warn "Database password not set in TF_VAR_db_password"
    read -s -p "Enter database password: " DB_PASSWORD
    echo
    export TF_VAR_db_password="$DB_PASSWORD"
fi

# Inicjalizuj Terraform
log_info "Initializing Terraform..."
terraform init -upgrade

# Sprawdź składnię
log_info "Validating Terraform configuration..."
terraform validate

# Sformatuj pliki
log_info "Formatting Terraform files..."
terraform fmt -recursive

# Utwórz plan
log_info "Creating Terraform plan..."
terraform plan -out=tfplan

# Zapytaj o potwierdzenie
if [ "$ENVIRONMENT" = "prod" ]; then
    log_warn "You are about to deploy to PRODUCTION environment!"
    read -p "Are you sure you want to continue? (yes/no): " CONFIRM
    if [ "$CONFIRM" != "yes" ]; then
        log_info "Deployment cancelled"
        exit 0
    fi
fi

# Zastosuj zmiany
log_info "Applying Terraform plan..."
terraform apply tfplan

# Wyczyść plan
rm -f tfplan
```

```
# Wyświetl ważne outputy
log_info "Deployment completed successfully!"
log_info "Application URL: $(terraform output -raw application_url)"

# Zapisz klucz SSH (jeśli nie istnieje)
SSH_KEY_FILE="$HOME/.ssh/${ENVIRONMENT}_key.pem"
if [ ! -f "$SSH_KEY_FILE" ]; then
    log_info "Saving SSH private key to $SSH_KEY_FILE"
    terraform output -raw ssh_private_key > "$SSH_KEY_FILE"
    chmod 600 "$SSH_KEY_FILE"
    log_info "SSH key saved. You can connect to instances using:"
    log_info "ssh -i $SSH_KEY_FILE ec2-user@<instance-ip>"
fi

log_info "Deployment script completed!"
```

## Użycie:

Bash

```
# Nadaj uprawnienia wykonywania
chmod +x scripts/deploy.sh

# Deploy do środowiska dev
export TF_VAR_db_password="your-secure-password"
./scripts/deploy.sh dev

# Deploy do środowiska prod
export TF_VAR_db_password="your-production-password"
./scripts/deploy.sh prod
```

Ten przykład pokazuje kompletną infrastrukturę aplikacji webowej z wszystkimi niezbędnymi komponentami: VPC z subnetami publicznymi i prywatnymi, Load Balancer, Auto Scaling Group, RDS, ElastiCache, S3, CloudFront, monitoring i alerting. Infrastruktura jest w pełni skalowalna i gotowa do użycia w środowisku produkcyjnym.

To kończy rozdział o praktycznych przykładach Terraform z AWS. W następnym rozdziale przejdziemy do AWS i usług chmurowych.

## 16. Wprowadzenie do AWS dla DevOps

# Amazon Web Services w kontekście DevOps

Amazon Web Services (AWS) to wiodąca platforma chmurowa, która rewolucjonizuje sposób, w jaki organizacje podchodzą do DevOps. AWS oferuje kompleksowy ekosystem usług, które wspierają każdy aspekt cyklu życia oprogramowania - od planowania i rozwoju, przez testowanie i wdrażanie, aż po monitorowanie i optymalizację.

Kluczową zaletą AWS w kontekście DevOps jest elastyczność i skalowalność infrastruktury. Tradycyjne podejście do zarządzania infrastrukturą wymagało znacznych inwestycji w sprzęt, długich czasów provisioning i ograniczonej elastyczności. AWS zmienia ten paradygmat, oferując infrastrukturę na żądanie, która może być skalowana w górę lub w dół w ciągu minut, a płatność następuje tylko za rzeczywiście wykorzystane zasoby.

Model chmury AWS idealnie wspiera filozofię DevOps poprzez umożliwienie szybkiego eksperymentowania, automatyzacji procesów i ciągłego dostarczania wartości biznesowej. Zespoły DevOps mogą skupić się na dostarczaniu funkcjonalności biznesowych zamiast zarządzania infrastrukturą, co znacząco przyspiesza time-to-market i zwiększa produktywność.

AWS oferuje ponad 200 usług, które można podzielić na kilka kluczowych kategorii istotnych dla DevOps: usługi obliczeniowe (EC2, Lambda, ECS, EKS), przechowywania danych (S3, EBS, EFS), baz danych (RDS, DynamoDB, ElastiCache), sieci (VPC, CloudFront, Route 53), bezpieczeństwa (IAM, KMS, WAF), monitorowania (CloudWatch, X-Ray), oraz narzędzi DevOps (CodePipeline, CodeBuild, CodeDeploy).

## Kluczowe usługi AWS dla DevOps

### Usługi obliczeniowe

**Amazon EC2 (Elastic Compute Cloud)** stanowi fundament wielu rozwiązań AWS. EC2 oferuje wirtualne serwery w chmurze, które mogą być uruchamiane, zatrzymywane i skalowane zgodnie z potrzebami. W kontekście DevOps, EC2 jest często używane do hostowania aplikacji, uruchamiania agentów CI/CD, oraz jako platforma dla kontenerów i orkiestracji.

EC2 oferuje różnorodne typy instancji zoptymalizowane pod konkretne zastosowania: instancje ogólnego przeznaczenia (t3, m5), zoptymalizowane pod obliczenia (c5), pamięć (r5, x1), przechowywanie danych (i3, d2), oraz akcelerowane obliczenia (p3, g4). Ta różnorodność pozwala na optymalizację kosztów i wydajności dla konkretnych workloadów.

Auto Scaling Groups (ASG) umożliwiają automatyczne skalowanie liczby instancji EC2 w odpowiedzi na zmieniające się obciążenie. To kluczowa funkcjonalność dla aplikacji, które muszą radzić sobie ze zmiennym ruchem, zapewniając jednocześnie optymalizację kosztów poprzez automatyczne zmniejszanie liczby instancji podczas niskiego obciążenia.

**AWS Lambda** reprezentuje paradygmat serverless computing, gdzie kod jest wykonywany w odpowiedzi na zdarzenia bez konieczności zarządzania serwerami. Lambda idealnie nadaje się do automatyzacji zadań DevOps, przetwarzania zdarzeń, integracji systemów oraz implementacji mikrouslug. Płatność następuje tylko za czas wykonania kodu, co czyni Lambda bardzo ekonomicznym rozwiązaniem dla sporadycznych zadań.

**Amazon ECS (Elastic Container Service)** i **EKS (Elastic Kubernetes Service)** to zarządzane usługi orkiestracji kontenerów. ECS to natywne rozwiązanie AWS do uruchamiania kontenerów Docker, podczas gdy EKS oferuje w pełni zarządzany Kubernetes. Obie usługi integrują się głęboko z ekosystemem AWS, oferując automatyczne skalowanie, load balancing, service discovery i integrację z innymi usługami AWS.

## Usługi przechowywania danych

**Amazon S3 (Simple Storage Service)** to obiektowy system przechowywania danych, który oferuje praktycznie nieograniczoną skalowalność, wysoką dostępność i trwałość danych. W kontekście DevOps, S3 jest używane do przechowywania artefaktów build, backup, logów, statycznych zasobów aplikacji webowych, oraz jako backend dla Terraform state.

S3 oferuje różne klasy przechowywania zoptymalizowane pod różne wzorce dostępu: Standard dla często używanych danych, Infrequent Access (IA) dla rzadziej używanych danych, Glacier dla archiwizacji długoterminowej, oraz Intelligent Tiering dla automatycznej optymalizacji kosztów.

**Amazon EBS (Elastic Block Store)** zapewnia trwałe, wysokowydajne przechowywanie blokowe dla instancji EC2. EBS oferuje różne typy wolumenów: gp3 i gp2 dla ogólnego

zastosowania, io1 i io2 dla wysokiej wydajności IOPS, st1 dla throughput-optimized workloads, oraz sc1 dla cold storage.

**Amazon EFS (Elastic File System)** to w pełni zarządzany system plików NFS, który może być współdzielony między wieloma instancjami EC2. EFS jest szczególnie przydatny dla aplikacji wymagających współdzielonego dostępu do plików, takich jak content management systems, web serving, czy data analytics.

## Usługi bazodanowe

**Amazon RDS (Relational Database Service)** to zarządzana usługa baz danych relacyjnych, która obsługuje MySQL, PostgreSQL, MariaDB, Oracle, Microsoft SQL Server oraz Amazon Aurora. RDS automatyzuje rutynowe zadania administracyjne takie jak provisioning sprzętu, setup bazy danych, patching, backup i recovery.

Aurora to natywna baza danych AWS, która oferuje wydajność i dostępność komercyjnych baz danych przy koszcie open source. Aurora automatycznie replikuje dane w trzech strefach dostępności i oferuje funkcje takie jak automatic failover, backup, recovery oraz point-in-time restore.

**Amazon DynamoDB** to w pełni zarządzana baza danych NoSQL, która oferuje jednomilisekundowe opóźnienia przy dowolnej skali. DynamoDB jest idealna dla aplikacji wymagających przewidywalnej wydajności przy wysokim obciążeniu, takich jak gaming, IoT, mobile applications czy real-time analytics.

**Amazon ElastiCache** oferuje w pełni zarządzane implementacje Redis i Memcached, które są używane jako cache w pamięci dla poprawy wydajności aplikacji. ElastiCache może znacząco zmniejszyć opóźnienia i obciążenie baz danych poprzez cache często używanych danych.

## Usługi sieciowe

**Amazon VPC (Virtual Private Cloud)** umożliwia tworzenie logicznie izolowanych sekcji chmury AWS, gdzie można uruchamiać zasoby w wirtualnej sieci. VPC oferuje pełną kontrolę nad środowiskiem sieciowym, włączając wybór zakresów IP, tworzenie subnetów, konfigurację route tables i network gateways.

Subnety w VPC mogą być publiczne (z dostępem do internetu przez Internet Gateway) lub prywatne (bez bezpośredniego dostępu do internetu). NAT Gateway umożliwia instancjom w prywatnych subnetach dostęp do internetu dla aktualizacji i pobierania pakietów, zachowując jednocześnie bezpieczeństwo.

**Elastic Load Balancing (ELB)** oferuje trzy typy load balancerów: Application Load Balancer (ALB) dla ruchu HTTP/HTTPS z zaawansowanym routowaniem, Network Load Balancer (NLB) dla ruchu TCP/UDP z ultra-niskimi opóźnieniami, oraz Classic Load Balancer dla prostych przypadków użycia.

**Amazon CloudFront** to globalna sieć dostarczania treści (CDN), która cache i dostarcza treści z lokalizacji najbliższych użytkownikom końcowym. CloudFront integruje się z innymi usługami AWS i oferuje funkcje takie jak SSL/TLS termination, custom headers, lambda@edge dla przetwarzania na brzegu sieci.

**Amazon Route 53** to skalowalny system DNS, który oferuje domain registration, DNS routing, health checking oraz traffic management. Route 53 może routować ruch na podstawie różnych kryteriów takich jak geolokalizacja, latencja, weighted routing czy failover.

## Konfiguracja konta AWS

### Tworzenie konta AWS

#### Krok 1: Rejestracja konta

Przejdź na stronę <https://aws.amazon.com> i kliknij "Create an AWS Account". Proces rejestracji wymaga podania:

- Adresu email i hasła
- Informacji kontaktowych
- Informacji płatniczych (karta kredytowa)
- Weryfikacji tożsamości (telefon)
- Wyboru planu wsparcia

AWS oferuje Free Tier, który umożliwia bezpłatne korzystanie z wielu usług przez pierwsze 12 miesięcy oraz stałe bezpłatne limity dla niektórych usług.

## Krok 2: Zabezpieczenie root account

Po utworzeniu konta, pierwszym krokiem powinno być zabezpieczenie root account:

Bash

```
# Włącz MFA (Multi-Factor Authentication) dla root account
# 1. Zaloguj się do AWS Console jako root user
# 2. Przejdź do "My Security Credentials"
# 3. W sekcji "Multi-factor authentication (MFA)" kliknij "Activate MFA"
# 4. Wybierz typ MFA device (Virtual MFA device, U2F security key, lub
Hardware MFA device)
# 5. Skonfiguruj wybrany typ MFA zgodnie z instrukcjami
```

## Krok 3: Tworzenie IAM użytkowników

Nigdy nie używaj root account do codziennych operacji. Utwórz dedykowanych użytkowników IAM:

Bash

```
# Zainstaluj AWS CLI
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
"awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install

# Sprawdź instalację
aws --version
```

## Konfiguracja IAM (Identity and Access Management)

### Tworzenie grup IAM:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```

{
  "Effect": "Allow",
  "Action": [
    "ec2:*",
    "s3:*",
    "rds:*",
    "iam:ListRoles",
    "iam:PassRole",
    "cloudformation:*",
    "cloudwatch:*",
    "logs:*"
  ],
  "Resource": "*"
}
]
}

```

## Tworzenie użytkownika DevOps:

Bash

```

# Utwórz grupę DevOps
aws iam create-group --group-name DevOps

# Utwórz policy dla grupy DevOps
cat > devops-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:*",
        "s3:*",
        "rds:*",
        "elasticloadbalancing:*",
        "autoscaling:*",
        "cloudwatch:*",
        "cloudformation:*",
        "iam:ListRoles",
        "iam:PassRole",
        "iam:CreateRole",
        "iam>DeleteRole",
        "iam:AttachRolePolicy",
        "iam:DetachRolePolicy",
        "iam>CreateInstanceProfile",
        "iam>DeleteInstanceProfile",

```



```

        "iam:AddRoleToInstanceProfile",
        "iam:RemoveRoleFromInstanceProfile",
        "logs:*",
        "sns:*",
        "sqs:*",
        "lambda:*",
        "apigateway:*",
        "route53:*",
        "cloudfront:*",
        "acm:*"
    ],
    "Resource": "*"
}
]
}
EOF

# Utwórz policy
aws iam create-policy \
    --policy-name DevOpsPolicy \
    --policy-document file:///devops-policy.json

# Przypisz policy do grupy
aws iam attach-group-policy \
    --group-name DevOps \
    --policy-arn arn:aws:iam::ACCOUNT-ID:policy/DevOpsPolicy

# Utwórz użytkownika
aws iam create-user --user-name devops-user

# Dodaj użytkownika do grupy
aws iam add-user-to-group \
    --user-name devops-user \
    --group-name DevOps

# Utwórz access keys
aws iam create-access-key --user-name devops-user

```

## Konfiguracja AWS CLI:

Bash

```

# Skonfiguruj AWS CLI z credentials nowego użytkownika
aws configure
# AWS Access Key ID: [Access Key z poprzedniego kroku]
# AWS Secret Access Key: [Secret Key z poprzedniego kroku]
# Default region name: eu-west-1

```

```
# Default output format: json

# Sprawdź konfigurację
aws sts get-caller-identity

# Utwórz profile dla różnych środowisk
aws configure set profile.dev.region eu-west-1
aws configure set profile.dev.aws_access_key_id YOUR_DEV_ACCESS_KEY
aws configure set profile.dev.aws_secret_access_key YOUR_DEV_SECRET_KEY

aws configure set profile.prod.region eu-west-1
aws configure set profile.prod.aws_access_key_id YOUR_PROD_ACCESS_KEY
aws configure set profile.prod.aws_secret_access_key YOUR_PROD_SECRET_KEY

# Użycie profili
aws s3 ls --profile dev
aws s3 ls --profile prod
```

## Organizacja zasobów AWS

### Tagging Strategy:

Spójne tagowanie zasobów jest kluczowe dla zarządzania kosztami, bezpieczeństwa i organizacji:

JSON

```
{
  "TaggingPolicy": {
    "RequiredTags": [
      {
        "Key": "Environment",
        "Values": ["dev", "staging", "prod"]
      },
      {
        "Key": "Project",
        "Values": ["webapp", "api", "analytics"]
      },
      {
        "Key": "Owner",
        "Values": ["team-alpha", "team-beta", "devops"]
      },
      {
        "Key": "CostCenter",
        "Values": ["engineering", "marketing", "operations"]
      }
    ]
  }
}
```

```

    }
  ],
  "OptionalTags": [
    {
      "Key": "Backup",
      "Values": ["required", "optional", "none"]
    },
    {
      "Key": "Monitoring",
      "Values": ["enabled", "disabled"]
    },
    {
      "Key": "CreatedBy",
      "Values": ["terraform", "manual", "cloudformation"]
    }
  ]
}

```

## Naming Conventions:

Bash

```

# Konwencja nazewnictwa zasobów
# Format: {project}-{environment}-{service}-{resource-type}-{identifier}

# Przykłady:
# webapp-prod-web-ec2-01
# webapp-prod-db-rds-primary
# webapp-prod-cache-elasticache-cluster
# webapp-prod-storage-s3-assets
# webapp-prod-cdn-cloudfront-main

# VPC i subnety:
# webapp-prod-vpc
# webapp-prod-public-subnet-1a
# webapp-prod-private-subnet-1a
# webapp-prod-database-subnet-1a

# Security Groups:
# webapp-prod-web-sg
# webapp-prod-db-sg
# webapp-prod-cache-sg
# webapp-prod-alb-sg

```

# AWS Organizations i Account Strategy

Dla większych organizacji zaleca się użycie AWS Organizations do zarządzania wieloma kontami:

## Multi-Account Strategy:

Plain Text

Root Organization

- └ Security Account (Centralne logowanie, audyt)
- └ Shared Services Account (DNS, monitoring)
- └ Development Account (środowiska dev/test)
- └ Staging Account (środowisko staging)
- └ Production Account (środowisko produkcyjne)
- └ Logging Account (centralne logi i audyt)

## Service Control Policies (SCPs):

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "ec2:TerminateInstances"
      ],
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "aws:PrincipalTag/Role": "Admin"
        }
      }
    },
    {
      "Effect": "Deny",
      "Action": [
        "rds:DeleteDBInstance",
        "rds:DeleteDBCluster"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
```

```
        "aws:RequestedRegion": "us-east-1"
      }
    }
  }
]
}
```

## AWS CLI i SDK

### Zaawansowane użycie AWS CLI

#### Konfiguracja i profile:

Bash

```
# Wyświetl aktualną konfigurację
aws configure list

# Wyświetl wszystkie profile
aws configure list-profiles

# Użyj konkretnego profilu
export AWS_PROFILE=production
aws s3 ls

# Lub jednorazowo
aws s3 ls --profile production

# Konfiguracja przez zmienne środowiskowe
export AWS_ACCESS_KEY_ID=your_access_key
export AWS_SECRET_ACCESS_KEY=your_secret_key
export AWS_DEFAULT_REGION=eu-west-1
export AWS_DEFAULT_OUTPUT=json
```

#### Filtrowanie i formatowanie outputu:

Bash

```
# Użyj JMESPath do filtrowania
aws ec2 describe-instances \
  --query 'Reservations[*].Instances[*].\
[InstanceId,State.Name,InstanceType]' \
  --output table
```

```
# Filtruj instancje według tagów
aws ec2 describe-instances \
  --filters "Name=tag:Environment,Values=production" \
  --query 'Reservations[*].Instances[*].[InstanceId,Tags[?
Key==`Name`.Value|[0]]' \
  --output table

# Wyświetl tylko running instances
aws ec2 describe-instances \
  --filters "Name=instance-state-name,Values=running" \
  --query 'Reservations[*].Instances[*].
[InstanceId,PublicIpAddress,PrivateIpAddress]' \
  --output table

# Formatuj output jako CSV
aws ec2 describe-instances \
  --query 'Reservations[*].Instances[*].
[InstanceId,InstanceType,State.Name]' \
  --output text | tr '\t' ','
```

## Bulk operations:

### Bash

```
# Zatrzymaj wszystkie instancje z konkretnym tagiem
aws ec2 describe-instances \
  --filters "Name=tag:Environment,Values=dev" "Name=instance-state-
name,Values=running" \
  --query 'Reservations[*].Instances[*].InstanceId' \
  --output text | xargs aws ec2 stop-instances --instance-ids

# Utwórz snapshots wszystkich wolumenów EBS
aws ec2 describe-volumes \
  --filters "Name=tag:Backup,Values=required" \
  --query 'Volumes[*].VolumeId' \
  --output text | while read volume_id; do
    aws ec2 create-snapshot \
      --volume-id $volume_id \
      --description "Automated backup $(date +%Y-%m-%d)"
done

# Usuń stare snapshots (starsze niż 30 dni)
aws ec2 describe-snapshots \
  --owner-ids self \
  --query "Snapshots[?StartTime<='$(date -d '30 days ago' --iso-
```

```
8601)'].SnapshotId" \  
--output text | xargs -n1 aws ec2 delete-snapshot --snapshot-id
```

## Skrypty automatyzacji AWS

### Skrypt backup EC2 instances:

Bash

```
#!/bin/bash  
  
# backup-ec2.sh - Automatyczny backup instancji EC2  
  
set -e  
  
# Konfiguracja  
RETENTION_DAYS=7  
TAG_FILTER="Name=tag:Backup,Values=required"  
LOG_FILE="/var/log/ec2-backup.log"  
  
# Funkcja logowania  
log() {  
    echo "$(date '+%Y-%m-%d %H:%M:%S') - $1" | tee -a "$LOG_FILE"  
}  
  
# Funkcja tworzenia snapshot  
create_snapshot() {  
    local instance_id=$1  
    local instance_name=$2  
  
    log "Creating snapshots for instance: $instance_id ($instance_name)"  
  
    # Pobierz wszystkie wolumeny przypisane do instancji  
    local volumes=$(aws ec2 describe-instances \  
        --instance-ids "$instance_id" \  
        --query  
'Reservations[*].Instances[*].BlockDeviceMappings[*].Ebs.VolumeId' \  
        --output text)  
  
    for volume_id in $volumes; do  
        if [ -n "$volume_id" ]; then  
            local snapshot_description="Automated backup of $instance_name  
($instance_id) - $(date '+%Y-%m-%d %H:%M:%S')"  
  
            local snapshot_id=$(aws ec2 create-snapshot \  
                --volume-id "$volume_id" \  
                --description "$snapshot_description" \  
                --tags "Name=tag:Backup,tag:Backup-Id=$instance_id" \  
                --retain-count $RETENTION_DAYS  
                --output text)
```

```

        --description "$snapshot_description" \
        --tag-specifications "ResourceType=snapshot,Tags=
[{{Key=Name,Value=$instance_name-backup}},{{Key=InstanceId,Value=$instance_id}},
{{Key=CreatedBy,Value=automated-backup}},
{{Key=RetentionDays,Value=$RETENTION_DAYS}}]" \
        --query 'SnapshotId' \
        --output text)

    log "Created snapshot: $snapshot_id for volume: $volume_id"
fi
done
}

# Funkcja usuwania starych snapshots
cleanup_old_snapshots() {
    log "Cleaning up snapshots older than $RETENTION_DAYS days"

    local cutoff_date=$(date -d "$RETENTION_DAYS days ago" --iso-8601)

    local old_snapshots=$(aws ec2 describe-snapshots \
        --owner-ids self \
        --filters "Name=tag:CreatedBy,Values=automated-backup" \
        --query "Snapshots[?StartTime<='$cutoff_date'].SnapshotId" \
        --output text)

    for snapshot_id in $old_snapshots; do
        if [ -n "$snapshot_id" ]; then
            aws ec2 delete-snapshot --snapshot-id "$snapshot_id"
            log "Deleted old snapshot: $snapshot_id"
        fi
    done
}

# Główna logika
main() {
    log "Starting EC2 backup process"

    # Pobierz instancje do backup
    local instances=$(aws ec2 describe-instances \
        --filters "$TAG_FILTER" "Name=instance-state-name,Values=running" \
        --query 'Reservations[*].Instances[*].[InstanceId,Tags[?
Key==`Name`.Value][0]]' \
        --output text)

    if [ -z "$instances" ]; then
        log "No instances found for backup"
        exit 0
    fi
}

```



```

# Przetwórz każdą instancję
echo "$instances" | while read instance_id instance_name; do
    if [ -n "$instance_id" ]; then
        create_snapshot "$instance_id" "${instance_name:-unnamed}"
    fi
done

# Wyczyść stare snapshots
cleanup_old_snapshots

log "EC2 backup process completed"
}

# Sprawdź czy AWS CLI jest skonfigurowane
if ! aws sts get-caller-identity &>/dev/null; then
    log "ERROR: AWS CLI is not configured or credentials are invalid"
    exit 1
fi

# Uruchom główną funkcję
main "$@"

```

## Skrypt monitorowania kosztów:

Bash

```

#!/bin/bash

# cost-monitor.sh - Monitoring kosztów AWS

set -e

# Konfiguracja
COST_THRESHOLD=1000 # USD
NOTIFICATION_EMAIL="devops@company.com"
SNS_TOPIC_ARN="arn:aws:sns:us-east-1:123456789012:cost-alerts"

# Pobierz koszty z ostatnich 30 dni
get_monthly_costs() {
    local start_date=$(date -d '30 days ago' '+%Y-%m-%d')
    local end_date=$(date '+%Y-%m-%d')

    aws ce get-cost-and-usage \
        --time-period Start="$start_date",End="$end_date" \
        --granularity MONTHLY \
        --metrics BlendedCost \

```

```

        --group-by Type=DIMENSION,Key=SERVICE \
        --query 'ResultsByTime[0].Groups[*].
[Keys[0],Metrics.BlendedCost.Amount]' \
        --output text | sort -k2 -nr
    }

# Pobierz przewidywane koszty na koniec miesiąca
get_forecasted_costs() {
    local start_date=$(date +%Y-%m-%d')
    local end_date=$(date -d 'next month' +%Y-%m-01')

    aws ce get-cost-and-usage \
        --time-period Start="$start_date",End="$end_date" \
        --granularity MONTHLY \
        --metrics BlendedCost \
        --query 'ResultsByTime[0].Total.BlendedCost.Amount' \
        --output text
}

# Wyślij alert
send_alert() {
    local message=$1

    # Wyślij przez SNS
    aws sns publish \
        --topic-arn "$SNS_TOPIC_ARN" \
        --subject "AWS Cost Alert" \
        --message "$message"

    echo "Alert sent: $message"
}

# Główna logika
main() {
    echo "AWS Cost Monitoring Report - $(date)"
    echo "======"

    # Pobierz aktualne koszty
    echo "Current month costs by service:"
    get_monthly_costs | head -10

    # Pobierz przewidywane koszty
    local forecasted_cost=$(get_forecasted_costs)
    echo ""
    echo "Forecasted cost for this month: \${forecasted_cost}"

    # Sprawdź czy przekroczono próg
    if (( $(echo "$forecasted_cost > $COST_THRESHOLD" | bc -l) )); then

```

```

        local alert_message="WARNING: Forecasted AWS costs
(\${forecasted_cost}) exceed threshold (\${COST_THRESHOLD})"
        send_alert "$alert_message"
    fi

    # Sprawdź największe koszty
    local top_service_cost=$(get_monthly_costs | head -1 | awk '{print $2}')
    local top_service_name=$(get_monthly_costs | head -1 | awk '{print $1}')

    echo ""
    echo "Top cost service: $top_service_name (\${top_service_cost})"

    # Alert jeśli jeden serwis kosztuje więcej niż 50% budżetu
    local half_threshold=$(echo "$COST_THRESHOLD * 0.5" | bc -l)
    if (( $(echo "$top_service_cost > $half_threshold" | bc -l) )); then
        local alert_message="WARNING: Service $top_service_name costs
\${top_service_cost}, which is more than 50% of budget"
        send_alert "$alert_message"
    fi
}

# Sprawdź zależności
if ! command -v bc &> /dev/null; then
    echo "ERROR: bc calculator is required but not installed"
    exit 1
fi

if ! aws sts get-caller-identity &>/dev/null; then
    echo "ERROR: AWS CLI is not configured or credentials are invalid"
    exit 1
fi

# Uruchom główną funkcję
main "$@"

```

To kończy wprowadzenie do AWS dla DevOps. W następnym rozdziale omówimy kluczowe usługi AWS dla CI/CD i automatyzacji.

## 17. Kluczowe usługi AWS dla CI/CD

### AWS Developer Tools - przegląd ekosystemu

Amazon Web Services oferuje kompletny zestaw narzędzi deweloperskich, które umożliwiają implementację zaawansowanych pipeline'ów CI/CD w pełni zintegrowanych z ekosystemem AWS. Te narzędzia zostały zaprojektowane z myślą o skalowalności, bezpieczeństwie i łatwości integracji z innymi usługami AWS.

Ekosystem AWS Developer Tools składa się z kilku kluczowych komponentów: AWS CodeCommit dla hostowania repozytoriów Git, AWS CodeBuild dla kompilacji i testowania kodu, AWS CodeDeploy dla automatyzacji wdrożeń, AWS CodePipeline dla orkiestracji całego procesu CI/CD, oraz AWS CodeStar dla zarządzania projektami. Dodatkowo, AWS oferuje CodeGuru dla analizy kodu i optymalizacji wydajności, X-Ray dla distributed tracing, oraz CloudFormation dla Infrastructure as Code.

Kluczową zaletą używania natywnych narzędzi AWS jest głęboka integracja z innymi usługami platformy. Pipeline'y mogą automatycznie skalować zasoby obliczeniowe, integrować się z systemami monitorowania CloudWatch, wykorzystywać IAM dla bezpieczeństwa, oraz seamlessly deployować do różnych usług AWS takich jak EC2, ECS, EKS, Lambda czy Elastic Beanstalk.

## AWS CodeCommit - zarządzanie kodem źródłowym

AWS CodeCommit to w pełni zarządzana usługa kontroli wersji, która hostuje prywatne repozytoria Git w chmurze AWS. CodeCommit oferuje wysoką dostępność, skalowalność i bezpieczeństwo, eliminując potrzebę zarządzania własnymi serwerami Git.

### Konfiguracja CodeCommit

#### Krok 1: Tworzenie repozytorium

Bash

```
# Utwórz nowe repozytorium
aws codecommit create-repository \
  --repository-name my-web-app \
  --repository-description "Repository for my web application"

# Wyświetl informacje o repozytorium
aws codecommit get-repository \
  --repository-name my-web-app
```

```
# Lista wszystkich repozytoriów
aws codecommit list-repositories
```

## Krok 2: Konfiguracja dostępu

Bash

```
# Utwórz IAM użytkownika dla CodeCommit
aws iam create-user --user-name codecommit-user

# Przypisz policy dla CodeCommit
cat > codecommit-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codecommit:GitPull",
        "codecommit:GitPush",
        "codecommit:ListRepositories",
        "codecommit:ListBranches",
        "codecommit:GetRepository",
        "codecommit:GetBranch",
        "codecommit:GetCommit",
        "codecommit:GetDifferences",
        "codecommit:GetReferences",
        "codecommit:BatchGetRepositories",
        "codecommit>CreateBranch",
        "codecommit>DeleteBranch"
      ],
      "Resource": "*"
    }
  ]
}
EOF

aws iam create-policy \
  --policy-name CodeCommitDeveloperAccess \
  --policy-document file://codecommit-policy.json

aws iam attach-user-policy \
  --user-name codecommit-user \
  --policy-arn arn:aws:iam::ACCOUNT-ID:policy/CodeCommitDeveloperAccess

# Utwórz Git credentials
aws iam create-service-specific-credential \
```

```
--user-name codecommit-user \  
--service-name codecommit.amazonaws.com
```

### Krok 3: Klonowanie i praca z repozytorium

Bash

```
# Skonfiguruj Git credentials helper  
git config --global credential.helper '!aws codecommit credential-helper $@'  
git config --global credential.UseHttpPath true  
  
# Klonuj repozytorium  
git clone https://git-codecommit.eu-west-1.amazonaws.com/v1/repos/my-web-app  
  
cd my-web-app  
  
# Utwórz przykładową aplikację  
cat > app.py << 'EOF'  
from flask import Flask, jsonify  
import os  
  
app = Flask(__name__)  
  
@app.route('/')  
def hello():  
    return jsonify({  
        'message': 'Hello from AWS CodeCommit!',  
        'version': os.environ.get('APP_VERSION', '1.0.0'),  
        'environment': os.environ.get('ENVIRONMENT', 'development')  
    })  
  
@app.route('/health')  
def health():  
    return jsonify({'status': 'healthy'})  
  
if __name__ == '__main__':  
    app.run(host='0.0.0.0', port=5000, debug=True)  
EOF  
  
cat > requirements.txt << 'EOF'  
Flask==2.3.3  
gunicorn==21.2.0  
EOF  
  
cat > Dockerfile << 'EOF'  
FROM python:3.11-slim
```

```
WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY app.py .

EXPOSE 5000

CMD ["gunicorn", "--bind", "0.0.0.0:5000", "app:app"]
EOF

# Commit i push
git add .
git commit -m "Initial commit with Flask application"
git push origin main
```

## Zaawansowane funkcje CodeCommit

### Branch protection i pull requests:

```
Bash

# Utwórz branch protection rule
aws codecommit put-repository-triggers \
  --repository-name my-web-app \
  --triggers '[
    {
      "name": "MainBranchProtection",
      "destinationArn": "arn:aws:sns:eu-west-1:ACCOUNT-ID:codecommit-
notifications",
      "customData": "Protection for main branch",
      "branches": ["main"],
      "events": ["all"]
    }
  ]'

# Utwórz approval rule template
cat > approval-rule.json << 'EOF'
{
  "approvalRuleTemplateName": "RequireApproval",
  "approvalRuleTemplateContent": "{\n\"Version\": \"2018-11-08\",
\n\"DestinationReferences\": [\n\"refs/heads/main\"], \n\"Statements\": [{\n\"Type\":
\n\"Approvers\", \n\"NumberOfApprovalsNeeded\": 2, \n\"ApprovalPoolMembers\":
[\n\"arn:aws:sts::ACCOUNT-ID:assumed-role/CodeCommitReviewer/*\"]}]}",
  "approvalRuleTemplateDescription": "Require 2 approvals for main branch"
```

```
}  
EOF
```

```
aws codecommit create-approval-rule-template \  
  --cli-input-json file://approval-rule.json
```

## AWS CodeBuild - kompilacja i testowanie

AWS CodeBuild to w pełni zarządzana usługa kompilacji, która kompiluje kod źródłowy, uruchamia testy i produkuje artefakty gotowe do wdrożenia. CodeBuild automatycznie skaluje się w zależności od potrzeb i płacisz tylko za czas kompilacji.

### Konfiguracja CodeBuild

#### Krok 1: Tworzenie buildspec.yml

YAML

```
# buildspec.yml  
version: 0.2  
  
env:  
  variables:  
    DOCKER_REGISTRY: "123456789012.dkr.ecr.eu-west-1.amazonaws.com"  
    IMAGE_REPO_NAME: "my-web-app"  
    IMAGE_TAG: "latest"  
  parameter-store:  
    DATABASE_URL: "/myapp/database/url"  
    API_KEY: "/myapp/api/key"  
  
phases:  
  install:  
    runtime-versions:  
      python: 3.11  
      docker: 20  
    commands:  
      - echo "Installing dependencies..."  
      - pip install --upgrade pip  
      - pip install -r requirements.txt  
      - pip install pytest pytest-cov flake8 black  
  
pre_build:  
  commands:
```



```

- echo "Running pre-build tasks..."
- echo "Logging in to Amazon ECR..."
- aws ecr get-login-password --region $AWS_DEFAULT_REGION | docker
login --username AWS --password-stdin $DOCKER_REGISTRY
- echo "Running code quality checks..."
- flake8 app.py --max-line-length=88
- black --check app.py
- echo "Running unit tests..."
- python -m pytest tests/ -v --cov=app --cov-report=xml --cov-
report=html
- echo "Build started on `date`"
- COMMIT_HASH=$(echo $CODEBUILD_RESOLVED_SOURCE_VERSION | cut -c 1-7)
- IMAGE_TAG=${COMMIT_HASH:=latest}

```

build:

commands:

```

- echo "Build phase started on `date`"
- echo "Building the Docker image..."
- docker build -t $IMAGE_REPO_NAME:$IMAGE_TAG .
- docker tag $IMAGE_REPO_NAME:$IMAGE_TAG
$DOCKER_REGISTRY/$IMAGE_REPO_NAME:$IMAGE_TAG
- docker tag $IMAGE_REPO_NAME:$IMAGE_TAG
$DOCKER_REGISTRY/$IMAGE_REPO_NAME:latest

```

post\_build:

commands:

```

- echo "Build completed on `date`"
- echo "Pushing the Docker images..."
- docker push $DOCKER_REGISTRY/$IMAGE_REPO_NAME:$IMAGE_TAG
- docker push $DOCKER_REGISTRY/$IMAGE_REPO_NAME:latest
- echo "Writing image definitions file..."
- printf ' [{"name": "my-web-app", "imageUri": "%s"} ]'
$DOCKER_REGISTRY/$IMAGE_REPO_NAME:$IMAGE_TAG > imagedefinitions.json
- echo "Generating deployment artifacts..."
- aws s3 cp coverage.xml s3://my-build-artifacts/coverage-
reports/coverage-$CODEBUILD_BUILD_NUMBER.xml
- aws s3 cp htmlcov/ s3://my-build-artifacts/coverage-
reports/html-$CODEBUILD_BUILD_NUMBER/ --recursive

```

artifacts:

files:

```

- imagedefinitions.json
- appspec.yml
- scripts/*

```

name: BuildArtifact-\$(date +%Y-%m-%d)

cache:

paths:

```
- '/root/.cache/pip/**/*'  
- '/var/lib/docker/**/*'
```

```
reports:  
  pytest_reports:  
    files:  
      - coverage.xml  
    base-directory: '.'  
    file-format: 'COBERTURAXML'  
  
  code_quality:  
    files:  
      - flake8-report.txt  
    base-directory: '.'  
    file-format: 'CHECKSTYLXML'
```

## Krok 2: Tworzenie projektu CodeBuild

Bash

```
# Utwórz S3 bucket dla artefaktów  
aws s3 mb s3://my-build-artifacts-$(date +%s)  
  
# Utwórz ECR repository  
aws ecr create-repository --repository-name my-web-app  
  
# Utwórz IAM role dla CodeBuild  
cat > codebuild-trust-policy.json << 'EOF'  
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Principal": {  
        "Service": "codebuild.amazonaws.com"  
      },  
      "Action": "sts:AssumeRole"  
    }  
  ]  
}  
EOF  
  
aws iam create-role \  
  --role-name CodeBuildServiceRole \  
  --assume-role-policy-document file://codebuild-trust-policy.json  
  
# Przypisz policy do roli
```

```
cat > codebuild-policy.json << 'EOF'
```

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "s3:GetObject",
        "s3:GetObjectVersion",
        "s3:PutObject",
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "ecr:GetAuthorizationToken",
        "ecr:InitiateLayerUpload",
        "ecr:UploadLayerPart",
        "ecr:CompleteLayerUpload",
        "ecr:PutImage",
        "ssm:GetParameters",
        "ssm:GetParameter",
        "codecommit:GitPull"
      ],
      "Resource": "*"
    }
  ]
}
EOF
```

```
aws iam create-policy \
  --policy-name CodeBuildServicePolicy \
  --policy-document file://codebuild-policy.json
```

```
aws iam attach-role-policy \
  --role-name CodeBuildServiceRole \
  --policy-arn arn:aws:iam::ACCOUNT-ID:policy/CodeBuildServicePolicy
```

```
# Utwórz projekt CodeBuild
```

```
cat > codebuild-project.json << 'EOF'
```

```
{
  "name": "my-web-app-build",
  "description": "Build project for my web application",
  "source": {
    "type": "CODECOMMIT",
    "location": "https://git-codecommit.eu-west-1.amazonaws.com/v1/repos/my-web-app",

```

```

        "buildspec": "buildspec.yml"
    },
    "artifacts": {
        "type": "S3",
        "location": "my-build-artifacts-123456789/artifacts"
    },
    "environment": {
        "type": "LINUX_CONTAINER",
        "image": "aws/codebuild/amazonlinux2-x86_64-standard:4.0",
        "computeType": "BUILD_GENERAL1_MEDIUM",
        "privilegedMode": true,
        "environmentVariables": [
            {
                "name": "AWS_DEFAULT_REGION",
                "value": "eu-west-1"
            },
            {
                "name": "AWS_ACCOUNT_ID",
                "value": "123456789012"
            }
        ]
    },
    "serviceRole": "arn:aws:iam::123456789012:role/CodeBuildServiceRole",
    "timeoutInMinutes": 30,
    "cache": {
        "type": "S3",
        "location": "my-build-artifacts-123456789/cache"
    }
}
EOF

```

```
aws codebuild create-project --cli-input-json file://codebuild-project.json
```

### Krok 3: Uruchomienie build

Bash

```

# Uruchom build
aws codebuild start-build --project-name my-web-app-build

# Sprawdź status build
BUILD_ID=$(aws codebuild list-builds-for-project \
    --project-name my-web-app-build \
    --query 'ids[0]' \
    --output text)

aws codebuild batch-get-builds --ids $BUILD_ID

```

```
# Pobierz logi build
aws logs get-log-events \
  --log-group-name /aws/codebuild/my-web-app-build \
  --log-stream-name $BUILD_ID
```

## Zaawansowane konfiguracje CodeBuild

### Multi-stage builds z różnymi środowiskami:

YAML

```
# buildspec-multi-stage.yml
version: 0.2

batch:
  fast-fail: false
  build-list:
    - identifier: unit_tests
      env:
        compute-type: BUILD_GENERAL1_SMALL
        image: aws/codebuild/amazonlinux2-x86_64-standard:4.0
      buildspec: |
        version: 0.2
        phases:
          install:
            runtime-versions:
              python: 3.11
            commands:
              - pip install -r requirements.txt
              - pip install pytest pytest-cov
          build:
            commands:
              - python -m pytest tests/unit/ -v --cov=app
        reports:
          unit_test_reports:
            files:
              - coverage.xml
            file-format: 'COBERTURAXML'

    - identifier: integration_tests
      env:
        compute-type: BUILD_GENERAL1_MEDIUM
        image: aws/codebuild/amazonlinux2-x86_64-standard:4.0
      buildspec: |
        version: 0.2
```

```

    phases:
      install:
        runtime-versions:
          python: 3.11
          docker: 20
        commands:
          - pip install -r requirements.txt
          - pip install pytest requests
      pre_build:
        commands:
          - docker-compose up -d database redis
          - sleep 30
      build:
        commands:
          - python -m pytest tests/integration/ -v
      post_build:
        commands:
          - docker-compose down

- identifier: security_scan
  env:
    compute-type: BUILD_GENERAL1_SMALL
    image: aws/codebuild/amazonlinux2-x86_64-standard:4.0
  buildspec: |
    version: 0.2
    phases:
      install:
        commands:
          - pip install bandit safety
      build:
        commands:
          - bandit -r app.py -f json -o bandit-report.json
          - safety check --json --output safety-report.json
  artifacts:
    files:
      - bandit-report.json
      - safety-report.json

- identifier: docker_build
  depend-on:
    - unit_tests
    - integration_tests
    - security_scan
  env:
    compute-type: BUILD_GENERAL1_MEDIUM
    image: aws/codebuild/amazonlinux2-x86_64-standard:4.0
    privileged-mode: true
  buildspec: |

```

```
version: 0.2
phases:
  pre_build:
    commands:
      - aws ecr get-login-password --region $AWS_DEFAULT_REGION |
docker login --username AWS --password-stdin $DOCKER_REGISTRY
  build:
    commands:
      - docker build -t $IMAGE_REPO_NAME:$IMAGE_TAG .
      - docker tag $IMAGE_REPO_NAME:$IMAGE_TAG
$DOCKER_REGISTRY/$IMAGE_REPO_NAME:$IMAGE_TAG
  post_build:
    commands:
      - docker push $DOCKER_REGISTRY/$IMAGE_REPO_NAME:$IMAGE_TAG
```

## AWS CodeDeploy - automatyzacja wdrożeń

AWS CodeDeploy automatyzuje wdrożenia aplikacji do różnych usług obliczeniowych AWS, takich jak EC2, Lambda, ECS oraz on-premises servers. CodeDeploy oferuje różne strategie deployment, monitoring wdrożeń oraz automatyczne rollback w przypadku problemów.

### Konfiguracja CodeDeploy dla EC2

#### Krok 1: Przygotowanie instancji EC2

Bash

```
# Utwórz IAM role dla instancji EC2
cat > ec2-codedeploy-trust-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
```

```
aws iam create-role \  
  --role-name EC2CodeDeployRole \  
  --assume-role-policy-document file://ec2-codedeploy-trust-policy.json  
  
# Przypisz policy do roli  
aws iam attach-role-policy \  
  --role-name EC2CodeDeployRole \  
  --policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy  
  
aws iam attach-role-policy \  
  --role-name EC2CodeDeployRole \  
  --policy-arn arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess  
  
# Utwórz instance profile  
aws iam create-instance-profile --instance-profile-name  
EC2CodeDeployInstanceProfile  
aws iam add-role-to-instance-profile \  
  --instance-profile-name EC2CodeDeployInstanceProfile \  
  --role-name EC2CodeDeployRole  
  
# User data script dla instalacji CodeDeploy agent  
cat > user-data.sh << 'EOF'  
#!/bin/bash  
yum update -y  
yum install -y ruby wget docker  
  
# Zainstaluj CodeDeploy agent  
cd /home/ec2-user  
wget https://aws-codedeploy-eu-west-1.s3.eu-west-  
1.amazonaws.com/latest/install  
chmod +x ./install  
./install auto  
  
# Uruchom Docker  
systemctl start docker  
systemctl enable docker  
usermod -a -G docker ec2-user  
  
# Zainstaluj Docker Compose  
curl -L "https://github.com/docker/compose/releases/latest/download/docker-  
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose  
chmod +x /usr/local/bin/docker-compose  
  
# Utwórz katalog aplikacji  
mkdir -p /opt/myapp  
chown ec2-user:ec2-user /opt/myapp  
EOF
```



```
# Uruchom instancję EC2
aws ec2 run-instances \
  --image-id ami-0c02fb55956c7d316 \
  --count 2 \
  --instance-type t3.micro \
  --key-name my-key-pair \
  --security-group-ids sg-12345678 \
  --subnet-id subnet-12345678 \
  --iam-instance-profile Name=EC2CodeDeployInstanceProfile \
  --user-data file:///user-data.sh \
  --tag-specifications 'ResourceType=instance,Tags=
[{{Key=Name,Value=WebServer}},{{Key=Environment,Value=production}},
{{Key=Application,Value=my-web-app}}]'
```

## Krok 2: Konfiguracja aplikacji CodeDeploy

Bash

```
# Utwórz IAM role dla CodeDeploy
cat > codedeploy-trust-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "codedeploy.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF

aws iam create-role \
  --role-name CodeDeployServiceRole \
  --assume-role-policy-document file:///codedeploy-trust-policy.json

aws iam attach-role-policy \
  --role-name CodeDeployServiceRole \
  --policy-arn arn:aws:iam::aws:policy/service-role/AWSCodeDeployRole

# Utwórz aplikację CodeDeploy
aws deploy create-application \
  --application-name my-web-app \
  --compute-platform Server
```

```
# Utwórz deployment group
aws deploy create-deployment-group \
  --application-name my-web-app \
  --deployment-group-name production \
  --service-role-arn arn:aws:iam::123456789012:role/CodeDeployServiceRole \
  --ec2-tag-filters Key=Application,Value=my-web-app,Type=KEY_AND_VALUE \
  --deployment-config-name CodeDeployDefault.AllAtOneTime \
  --auto-rollback-configuration
enabled=true,events=DEPLOYMENT_FAILURE,DEPLOYMENT_STOP_ON_ALARM,DEPLOYMENT_STOP_ON_INSTANCE_FAILURE \
  --alarm-configuration enabled=true,alarms=
[{"name=AliasErrorMetricGreaterThanZeroAlarm}]
```

### Krok 3: Przygotowanie appspec.yml

#### YAML

```
# appspec.yml
version: 0.0
os: linux
files:
  - source: /
    destination: /opt/myapp
    overwrite: yes
permissions:
  - object: /opt/myapp
    pattern: "*"
    owner: ec2-user
    group: ec2-user
    mode: 755
  - object: /opt/myapp/scripts
    pattern: "*"
    owner: ec2-user
    group: ec2-user
    mode: 755
hooks:
  BeforeInstall:
    - location: scripts/stop_application.sh
      timeout: 300
      runas: root
  AfterInstall:
    - location: scripts/install_dependencies.sh
      timeout: 300
      runas: root
  ApplicationStart:
    - location: scripts/start_application.sh
      timeout: 300
```

```
runas: root
ApplicationStop:
  - location: scripts/stop_application.sh
    timeout: 300
    runas: root
ValidateService:
  - location: scripts/validate_service.sh
    timeout: 300
    runas: ec2-user
```

## Krok 4: Skrypty deployment

Bash

```
# scripts/stop_application.sh
#!/bin/bash
set -e

echo "Stopping application..."

# Zatrzymaj Docker containers
if [ -f /opt/myapp/docker-compose.yml ]; then
    cd /opt/myapp
    /usr/local/bin/docker-compose down || true
fi

# Zatrzymaj nginx jeśli działa
systemctl stop nginx || true

echo "Application stopped successfully"
```

Bash

```
# scripts/install_dependencies.sh
#!/bin/bash
set -e

echo "Installing dependencies..."

# Aktualizuj system
yum update -y

# Zainstaluj nginx jeśli nie jest zainstalowany
if ! command -v nginx &> /dev/null; then
    amazon-linux-extras install nginx1 -y
```

```
fi

# Skopiuj konfigurację nginx
if [ -f /opt/myapp/nginx.conf ]; then
    cp /opt/myapp/nginx.conf /etc/nginx/nginx.conf
fi

echo "Dependencies installed successfully"
```

Bash

```
# scripts/start_application.sh
#!/bin/bash
set -e

echo "Starting application..."

cd /opt/myapp

# Uruchom aplikację przez Docker Compose
if [ -f docker-compose.yml ]; then
    /usr/local/bin/docker-compose up -d

    # Poczekaj na uruchomienie aplikacji
    sleep 30
fi

# Uruchom nginx
systemctl start nginx
systemctl enable nginx

echo "Application started successfully"
```

Bash

```
# scripts/validate_service.sh
#!/bin/bash
set -e

echo "Validating service..."

# Sprawdź czy aplikacja odpowiada
max_attempts=30
attempt=1

while [ $attempt -le $max_attempts ]; do
```

```

if curl -f http://localhost/health; then
    echo "Service validation successful"
    exit 0
fi

echo "Attempt $attempt failed, retrying in 10 seconds..."
sleep 10
attempt=$((attempt + 1))
done

echo "Service validation failed after $max_attempts attempts"
exit 1

```

## Blue/Green Deployment z CodeDeploy

Bash

```

# Utwórz deployment group dla Blue/Green
aws deploy create-deployment-group \
  --application-name my-web-app \
  --deployment-group-name blue-green-production \
  --service-role-arn arn:aws:iam::123456789012:role/CodeDeployServiceRole \
  --blue-green-deployment-configuration '{
    "terminateBlueInstancesOnDeploymentSuccess": {
      "action": "TERMINATE",
      "terminationWaitTimeInMinutes": 5
    },
    "deploymentReadyOption": {
      "actionOnTimeout": "CONTINUE_DEPLOYMENT"
    },
    "greenFleetProvisioningOption": {
      "action": "COPY_AUTO_SCALING_GROUP"
    }
  }' \
  --load-balancer-info '{
    "targetGroupInfoList": [
      {
        "name": "my-web-app-tg"
      }
    ]
  }' \
  --auto-scaling-groups my-web-app-asg \
  --deployment-config-name CodeDeployDefault.BlueGreenAllAtOnce

```

# AWS CodePipeline - orkiestracja CI/CD

AWS CodePipeline to usługa ciągłej integracji i ciągłego dostarczania, która automatyzuje kroki release procesu. Pipeline składa się z etapów (stages), a każdy etap zawiera akcje (actions), które są wykonywane sekwencyjnie lub równolegle.

## Tworzenie kompletnego pipeline

JSON

```
{
  "pipeline": {
    "name": "my-web-app-pipeline",
    "roleArn": "arn:aws:iam::123456789012:role/CodePipelineServiceRole",
    "artifactStore": {
      "type": "S3",
      "location": "my-pipeline-artifacts-bucket"
    },
    "stages": [
      {
        "name": "Source",
        "actions": [
          {
            "name": "SourceAction",
            "actionTypeId": {
              "category": "Source",
              "owner": "AWS",
              "provider": "CodeCommit",
              "version": "1"
            },
            "configuration": {
              "RepositoryName": "my-web-app",
              "BranchName": "main",
              "PollForSourceChanges": "false"
            },
            "outputArtifacts": [
              {
                "name": "SourceOutput"
              }
            ]
          }
        ]
      }
    ],
    "name": "Build",
```

```

    "actions": [
      {
        "name": "BuildAction",
        "actionTypeId": {
          "category": "Build",
          "owner": "AWS",
          "provider": "CodeBuild",
          "version": "1"
        },
        "configuration": {
          "ProjectName": "my-web-app-build"
        },
        "inputArtifacts": [
          {
            "name": "SourceOutput"
          }
        ],
        "outputArtifacts": [
          {
            "name": "BuildOutput"
          }
        ]
      }
    ]
  },
  {
    "name": "Test",
    "actions": [
      {
        "name": "UnitTests",
        "actionTypeId": {
          "category": "Test",
          "owner": "AWS",
          "provider": "CodeBuild",
          "version": "1"
        },
        "configuration": {
          "ProjectName": "my-web-app-unit-tests"
        },
        "inputArtifacts": [
          {
            "name": "SourceOutput"
          }
        ],
        "runOrder": 1
      },
      {
        "name": "IntegrationTests",

```

```

        "actionTypeId": {
            "category": "Test",
            "owner": "AWS",
            "provider": "CodeBuild",
            "version": "1"
        },
        "configuration": {
            "ProjectName": "my-web-app-integration-tests"
        },
        "inputArtifacts": [
            {
                "name": "BuildOutput"
            }
        ],
        "runOrder": 2
    }
]
},
{
    "name": "Deploy-Staging",
    "actions": [
        {
            "name": "DeployToStaging",
            "actionTypeId": {
                "category": "Deploy",
                "owner": "AWS",
                "provider": "CodeDeploy",
                "version": "1"
            },
            "configuration": {
                "ApplicationName": "my-web-app",
                "DeploymentGroupName": "staging"
            },
            "inputArtifacts": [
                {
                    "name": "BuildOutput"
                }
            ]
        }
    ]
},
{
    "name": "Approval",
    "actions": [
        {
            "name": "ManualApproval",
            "actionTypeId": {
                "category": "Approval",

```



```

        "owner": "AWS",
        "provider": "Manual",
        "version": "1"
    },
    "configuration": {
        "CustomData": "Please review the staging
deployment and approve for production",
        "NotificationArn": "arn:aws:sns:eu-west-
1:123456789012:pipeline-approvals"
    }
}
]
},
{
    "name": "Deploy-Production",
    "actions": [
        {
            "name": "DeployToProduction",
            "actionTypeId": {
                "category": "Deploy",
                "owner": "AWS",
                "provider": "CodeDeploy",
                "version": "1"
            },
            "configuration": {
                "ApplicationName": "my-web-app",
                "DeploymentGroupName": "production"
            },
            "inputArtifacts": [
                {
                    "name": "BuildOutput"
                }
            ]
        }
    ]
}
]
}
]
}
}

```

## Zaawansowane funkcje Pipeline

### Multi-region deployment:

JSON

```

{
  "name": "Deploy-Multi-Region",
  "actions": [
    {
      "name": "DeployToEuWest1",
      "actionTypeId": {
        "category": "Deploy",
        "owner": "AWS",
        "provider": "CloudFormation",
        "version": "1"
      },
      "configuration": {
        "ActionMode": "CREATE_UPDATE",
        "StackName": "my-web-app-eu-west-1",
        "TemplatePath": "BuildOutput::infrastructure/template.yml",
        "Capabilities": "CAPABILITY_IAM",
        "RoleArn":
"arn:aws:iam::123456789012:role/CloudFormationRole",
        "ParameterOverrides": "
{\\"Environment\\":\\"production\\",\\"Region\\":\\"eu-west-1\\"}"
      },
      "inputArtifacts": [
        {
          "name": "BuildOutput"
        }
      ],
      "region": "eu-west-1",
      "runOrder": 1
    },
    {
      "name": "DeployToUsEast1",
      "actionTypeId": {
        "category": "Deploy",
        "owner": "AWS",
        "provider": "CloudFormation",
        "version": "1"
      },
      "configuration": {
        "ActionMode": "CREATE_UPDATE",
        "StackName": "my-web-app-us-east-1",
        "TemplatePath": "BuildOutput::infrastructure/template.yml",
        "Capabilities": "CAPABILITY_IAM",
        "RoleArn":
"arn:aws:iam::123456789012:role/CloudFormationRole",
        "ParameterOverrides": "
{\\"Environment\\":\\"production\\",\\"Region\\":\\"us-east-1\\"}"
      },

```

```

        "inputArtifacts": [
            {
                "name": "BuildOutput"
            }
        ],
        "region": "us-east-1",
        "runOrder": 1
    }
]
}

```

## Pipeline z Lambda functions:

Python

```

# lambda-deployment-function.py
import json
import boto3
import urllib3

def lambda_handler(event, context):
    """
    Custom deployment function for CodePipeline
    """

    codepipeline = boto3.client('codepipeline')

    # Pobierz job data
    job_id = event['CodePipeline.job']['id']
    job_data = event['CodePipeline.job']['data']

    try:
        # Pobierz parametry z job data
        user_parameters = job_data['actionConfiguration']['configuration']
    except ['UserParameters']:
        params = json.loads(user_parameters)

    # Wykonaj custom deployment logic
    deployment_result = perform_custom_deployment(params)

    if deployment_result['success']:
        # Oznacz job jako sukces
        codepipeline.put_job_success_result(jobId=job_id)

    return {
        'statusCode': 200,
        'body': json.dumps('Deployment successful')
    }

```

```

    }
    else:
        # Oznacz job jako failure
        codepipeline.put_job_failure_result(
            jobId=job_id,
            failureDetails={'message': deployment_result['error'],
'type': 'JobFailed'}
        )

        return {
            'statusCode': 500,
            'body': json.dumps('Deployment failed')
        }

except Exception as e:
    # Oznacz job jako failure
    codepipeline.put_job_failure_result(
        jobId=job_id,
        failureDetails={'message': str(e), 'type': 'JobFailed'}
    )

    return {
        'statusCode': 500,
        'body': json.dumps(f'Error: {str(e)}')
    }

def perform_custom_deployment(params):
    """
    Wykonaj custom deployment logic
    """
    try:
        # Przykład: deploy do external service
        http = urllib3.PoolManager()

        deployment_data = {
            'application': params['application'],
            'version': params['version'],
            'environment': params['environment']
        }

        response = http.request(
            'POST',
            params['deployment_endpoint'],
            body=json.dumps(deployment_data),
            headers={'Content-Type': 'application/json'}
        )

        if response.status == 200:

```

```
        return {'success': True}
    else:
        return {'success': False, 'error': f'Deployment API returned
{response.status}'}

except Exception as e:
    return {'success': False, 'error': str(e)}
```

To kończy rozdział o kluczowych usługach AWS dla CI/CD. W następnym rozdziale omówimy monitoring i logging w AWS.

## 18. Monitoring i logging w AWS

### Amazon CloudWatch - kompleksowe monitorowanie

Amazon CloudWatch to centralny system monitorowania i observability dla zasobów AWS i aplikacji. CloudWatch zbiera i śledzi metryki, monitoruje pliki logów, ustawia alarmy oraz automatycznie reaguje na zmiany w środowisku AWS. Jest to fundament strategii monitorowania w ekosystemie AWS.

CloudWatch oferuje kilka kluczowych komponentów: CloudWatch Metrics dla zbierania i przechowywania metryk, CloudWatch Logs dla centralizacji logów, CloudWatch Alarms dla alertowania, CloudWatch Events (teraz EventBridge) dla automatyzacji opartej na zdarzeniach, oraz CloudWatch Dashboards dla wizualizacji danych.

Kluczową zaletą CloudWatch jest jego głęboka integracja z usługami AWS. Większość usług automatycznie publikuje metryki do CloudWatch, co zapewnia natychmiastową widoczność stanu infrastruktury bez dodatkowej konfiguracji. CloudWatch oferuje również możliwość tworzenia custom metrics, co pozwala na monitorowanie specyficznych aspektów aplikacji.

### Konfiguracja CloudWatch Metrics

#### Podstawowe metryki AWS:

Bash

```
# Wyświetl dostępne metryki dla EC2
aws cloudwatch list-metrics --namespace AWS/EC2
```

```
# Pobierz metryki CPU dla konkretnej instancji
aws cloudwatch get-metric-statistics \
  --namespace AWS/EC2 \
  --metric-name CPUUtilization \
  --dimensions Name=InstanceId,Value=i-1234567890abcdef0 \
  --start-time 2024-01-01T00:00:00Z \
  --end-time 2024-01-01T23:59:59Z \
  --period 3600 \
  --statistics Average,Maximum

# Pobierz metryki dla Load Balancer
aws cloudwatch get-metric-statistics \
  --namespace AWS/ApplicationELB \
  --metric-name RequestCount \
  --dimensions Name=LoadBalancer,Value=app/my-load-
balancer/50dc6c495c0c9188 \
  --start-time 2024-01-01T00:00:00Z \
  --end-time 2024-01-01T23:59:59Z \
  --period 300 \
  --statistics Sum
```

## Custom metrics:

### Python

```
# custom-metrics.py
import boto3
import time
import psutil
import requests
from datetime import datetime

cloudwatch = boto3.client('cloudwatch')

def publish_system_metrics():
    """Publikuj custom metryki systemowe"""

    # CPU usage
    cpu_percent = psutil.cpu_percent(interval=1)

    # Memory usage
    memory = psutil.virtual_memory()
    memory_percent = memory.percent

    # Disk usage
    disk = psutil.disk_usage('/')
    disk_percent = (disk.used / disk.total) * 100
```

```

# Network I/O
network = psutil.net_io_counters()

# Publikuj metryki
cloudwatch.put_metric_data(
    Namespace='Custom/System',
    MetricData=[
        {
            'MetricName': 'CPUUtilization',
            'Value': cpu_percent,
            'Unit': 'Percent',
            'Dimensions': [
                {
                    'Name': 'InstanceId',
                    'Value': get_instance_id()
                }
            ]
        },
        {
            'MetricName': 'MemoryUtilization',
            'Value': memory_percent,
            'Unit': 'Percent',
            'Dimensions': [
                {
                    'Name': 'InstanceId',
                    'Value': get_instance_id()
                }
            ]
        },
        {
            'MetricName': 'DiskUtilization',
            'Value': disk_percent,
            'Unit': 'Percent',
            'Dimensions': [
                {
                    'Name': 'InstanceId',
                    'Value': get_instance_id()
                }
            ]
        }
    ]
)

```

```

def publish_application_metrics():
    """Publikuj metryki aplikacji"""

    try:

```

```

# Sprawdź response time aplikacji
start_time = time.time()
response = requests.get('http://localhost:5000/health', timeout=5)
response_time = (time.time() - start_time) * 1000

# Status aplikacji
app_status = 1 if response.status_code == 200 else 0

cloudwatch.put_metric_data(
    Namespace='Custom/Application',
    MetricData=[
        {
            'MetricName': 'ResponseTime',
            'Value': response_time,
            'Unit': 'Milliseconds',
            'Dimensions': [
                {
                    'Name': 'Environment',
                    'Value': 'production'
                },
                {
                    'Name': 'Service',
                    'Value': 'web-app'
                }
            ]
        },
        {
            'MetricName': 'HealthStatus',
            'Value': app_status,
            'Unit': 'Count',
            'Dimensions': [
                {
                    'Name': 'Environment',
                    'Value': 'production'
                },
                {
                    'Name': 'Service',
                    'Value': 'web-app'
                }
            ]
        }
    ]
)

```

```

except Exception as e:
    # Publikuj metrykę błędu
    cloudwatch.put_metric_data(
        Namespace='Custom/Application',

```



```

        MetricData=[
            {
                'MetricName': 'HealthCheckErrors',
                'Value': 1,
                'Unit': 'Count',
                'Dimensions': [
                    {
                        'Name': 'Environment',
                        'Value': 'production'
                    },
                    {
                        'Name': 'Service',
                        'Value': 'web-app'
                    }
                ]
            }
        ]
    )

def get_instance_id():
    """Pobierz Instance ID z metadata"""
    try:
        response = requests.get(
            'http://169.254.169.254/latest/meta-data/instance-id',
            timeout=2
        )
        return response.text
    except:
        return 'unknown'

def main():
    """Główna pętla monitorowania"""
    while True:
        try:
            publish_system_metrics()
            publish_application_metrics()
            print(f"Metrics published at {datetime.now()}")
        except Exception as e:
            print(f"Error publishing metrics: {e}")

        time.sleep(60) # Publikuj co minutę

if __name__ == "__main__":
    main()

```

## CloudWatch Alarms

## Tworzenie alarmów:

Bash

```
# Alarm dla wysokiego CPU
aws cloudwatch put-metric-alarm \
  --alarm-name "High-CPU-Utilization" \
  --alarm-description "Alarm when CPU exceeds 80%" \
  --metric-name CPUUtilization \
  --namespace AWS/EC2 \
  --statistic Average \
  --period 300 \
  --threshold 80 \
  --comparison-operator GreaterThanThreshold \
  --evaluation-periods 2 \
  --alarm-actions arn:aws:sns:eu-west-1:123456789012:high-cpu-topic \
  --dimensions Name=InstanceId,Value=i-1234567890abcdef0

# Alarm dla błędów aplikacji
aws cloudwatch put-metric-alarm \
  --alarm-name "Application-Health-Check-Failures" \
  --alarm-description "Alarm when health check fails" \
  --metric-name HealthCheckErrors \
  --namespace Custom/Application \
  --statistic Sum \
  --period 300 \
  --threshold 3 \
  --comparison-operator GreaterThanThreshold \
  --evaluation-periods 1 \
  --alarm-actions arn:aws:sns:eu-west-1:123456789012:app-alerts-topic \
  --dimensions Name=Environment,Value=production Name=Service,Value=web-app

# Composite alarm
aws cloudwatch put-composite-alarm \
  --alarm-name "Application-Critical-Issues" \
  --alarm-description "Critical issues with application" \
  --alarm-rule "(ALARM('High-CPU-Utilization') OR ALARM('Application-Health-Check-Failures'))" \
  --actions-enabled \
  --alarm-actions arn:aws:sns:eu-west-1:123456789012:critical-alerts-topic
```

## Auto Scaling z CloudWatch:

Bash

```
# Utwórz scaling policy
aws autoscaling put-scaling-policy \
  --auto-scaling-group-name my-web-app-asg \
  --policy-name scale-up-policy \
  --policy-type TargetTrackingScaling \
  --target-tracking-configuration '{
    "TargetValue": 70.0,
    "PredefinedMetricSpecification": {
      "PredefinedMetricType": "ASGAverageCPUUtilization"
    },
    "ScaleOutCooldown": 300,
    "ScaleInCooldown": 300
  }'

# Custom metric scaling
aws autoscaling put-scaling-policy \
  --auto-scaling-group-name my-web-app-asg \
  --policy-name scale-on-response-time \
  --policy-type TargetTrackingScaling \
  --target-tracking-configuration '{
    "TargetValue": 200.0,
    "CustomizedMetricSpecification": {
      "MetricName": "ResponseTime",
      "Namespace": "Custom/Application",
      "Dimensions": [
        {
          "Name": "Environment",
          "Value": "production"
        }
      ],
      "Statistic": "Average"
    }
  }'
```

## CloudWatch Logs

### Konfiguracja CloudWatch Logs Agent:

Bash

```
# Zainstaluj CloudWatch Logs Agent
wget https://s3.amazonaws.com/aws-cloudwatch/downloads/latest/awslogs-agent-setup.py
sudo python3 awslogs-agent-setup.py --region eu-west-1
```

```
# Konfiguracja awslogs.conf
cat > /etc/awslogs/awslogs.conf << 'EOF'
[general]
state_file = /var/lib/awslogs/agent-state

[/var/log/messages]
file = /var/log/messages
log_group_name = /aws/ec2/var/log/messages
log_stream_name = {instance_id}
datetime_format = %b %d %H:%M:%S

[/var/log/application.log]
file = /opt/myapp/logs/application.log
log_group_name = /aws/application/logs
log_stream_name = {instance_id}
datetime_format = %Y-%m-%d %H:%M:%S

[/var/log/nginx/access.log]
file = /var/log/nginx/access.log
log_group_name = /aws/nginx/access
log_stream_name = {instance_id}
datetime_format = %d/%b/%Y:%H:%M:%S %z

[/var/log/nginx/error.log]
file = /var/log/nginx/error.log
log_group_name = /aws/nginx/error
log_stream_name = {instance_id}
datetime_format = %Y/%m/%d %H:%M:%S
EOF

# Uruchom agent
sudo systemctl start awslogsd
sudo systemctl enable awslogsd
```

## CloudWatch Logs Insights:

### SQL

```
-- Znajdź błędy w logach aplikacji
fields @timestamp, @message
| filter @message like /ERROR/
| sort @timestamp desc
| limit 100

-- Analiza response time
fields @timestamp, @message
| filter @message like /response_time/
```

```

| parse @message "response_time: * ms" as response_time
| stats avg(response_time), max(response_time), min(response_time) by bin(5m)

-- Top IP addresses w logach nginx
fields @timestamp, @message
| parse @message "*" - - [*] "\" * * \"*\" \"*\" as ip, timestamp,
method, path, protocol, status, size, referer, user_agent
| stats count() as requests by ip
| sort requests desc
| limit 10

-- Błędy 5xx w ostatniej godzinie
fields @timestamp, @message
| filter @timestamp > @timestamp - 1h
| parse @message "*" - - [*] "\" * * \"*\" \"*\" as ip, timestamp,
method, path, protocol, status, size, referer, user_agent
| filter status >= 500
| stats count() as error_count by status, path
| sort error_count desc

```

## Metric Filters:

### Bash

```

# Utwórz metric filter dla błędów aplikacji
aws logs put-metric-filter \
  --log-group-name /aws/application/logs \
  --filter-name ApplicationErrors \
  --filter-pattern "ERROR" \
  --metric-transformations \

metricName=ApplicationErrorCount,metricNamespace=Custom/Application,metricValue=1,defaultValue=0

# Metric filter dla response time
aws logs put-metric-filter \
  --log-group-name /aws/application/logs \
  --filter-name ResponseTimeMetric \
  --filter-pattern "[timestamp, level, message=\"response_time:\", time,
unit=\"ms\"]" \
  --metric-transformations \

metricName=ResponseTime,metricNamespace=Custom/Application,metricValue='$time
',defaultValue=0

# Metric filter dla HTTP status codes
aws logs put-metric-filter \

```

```
--log-group-name /aws/nginx/access \  
--filter-name HTTP5xxErrors \  
--filter-pattern "[ip, id, user, timestamp, request, status_code=5*,  
size, referer, user_agent]" \  
--metric-transformations \  
  
metricName=HTTP5xxErrors,metricNamespace=Custom/WebServer,metricValue=1,defaultValue=0
```

## AWS X-Ray - distributed tracing

AWS X-Ray pomaga w analizie i debugowaniu aplikacji rozproszonych poprzez śledzenie requestów przez różne komponenty aplikacji. X-Ray jest szczególnie przydatny w architekturach mikrousług, gdzie request może przechodzić przez wiele usług.

### Konfiguracja X-Ray

#### Instalacja X-Ray daemon:

Bash

```
# Pobierz i zainstaluj X-Ray daemon  
wget https://s3.us-east-2.amazonaws.com/aws-xray-assets.us-east-2/xray-daemon/aws-xray-daemon-3.x.zip  
unzip aws-xray-daemon-3.x.zip  
sudo cp xray /usr/local/bin/  
sudo chmod +x /usr/local/bin/xray  
  
# Utwórz systemd service  
cat > /etc/systemd/system/xray.service << 'EOF'  
[Unit]  
Description=AWS X-Ray Daemon  
After=network.target  
  
[Service]  
Type=simple  
User=xray  
ExecStart=/usr/local/bin/xray -o -n eu-west-1  
Restart=on-failure  
RestartSec=10  
  
[Install]  
WantedBy=multi-user.target
```

EOF

```
# Utwórz użytkownika dla X-Ray
sudo useradd --system --shell /bin/false xray

# Uruchom service
sudo systemctl daemon-reload
sudo systemctl start xray
sudo systemctl enable xray
```

## Instrumentacja aplikacji Python:

Python

```
# app-with-xray.py
from flask import Flask, request, jsonify
import requests
import boto3
import time
import random

# X-Ray imports
from aws_xray_sdk.core import xray_recorder, patch_all
from aws_xray_sdk.core.context import Context
from aws_xray_sdk.core.plugins import EC2Plugin
from aws_xray_sdk.flask.middleware import XRayMiddleware

# Patch AWS SDK calls
patch_all()

app = Flask(__name__)

# Konfiguruj X-Ray
xray_recorder.configure(
    context=Context(),
    service='my-web-app',
    plugins=('EC2Plugin',),
    daemon_address='127.0.0.1:2000'
)

# Dodaj X-Ray middleware
XRayMiddleware(app, xray_recorder)

# AWS clients
dynamodb = boto3.resource('dynamodb', region_name='eu-west-1')
s3 = boto3.client('s3', region_name='eu-west-1')
```

```

@app.route('/')
@xray_recorder.capture('home_page')
def home():
    # Dodaj custom metadata
    xray_recorder.current_subsegment().put_metadata('user_agent',
request.headers.get('User-Agent'))
    xray_recorder.current_subsegment().put_annotation('endpoint', 'home')

    return jsonify({
        'message': 'Hello from X-Ray instrumented app!',
        'trace_id': xray_recorder.current_trace_entity().trace_id
    })

@app.route('/api/users/<user_id>')
@xray_recorder.capture('get_user')
def get_user(user_id):
    # Dodaj annotations dla filtrowania
    xray_recorder.current_subsegment().put_annotation('user_id', user_id)
    xray_recorder.current_subsegment().put_annotation('operation',
'get_user')

    # Symuluj wywołanie do bazy danych
    user_data = get_user_from_database(user_id)

    # Symuluj wywołanie do external API
    profile_data = get_user_profile_from_api(user_id)

    return jsonify({
        'user': user_data,
        'profile': profile_data
    })

@xray_recorder.capture('database_query')
def get_user_from_database(user_id):
    # Symuluj opóźnienie bazy danych
    time.sleep(random.uniform(0.1, 0.5))

    try:
        table = dynamodb.Table('users')
        response = table.get_item(Key={'user_id': user_id})

        # Dodaj metadata o query
        xray_recorder.current_subsegment().put_metadata('table_name',
'users')
        xray_recorder.current_subsegment().put_metadata('query_type',
'get_item')

        return response.get('Item', {})

```



```

except Exception as e:
    # Dodaj informacje o błędzie
    xray_recorder.current_subsegment().add_exception(e)
    raise

@xray_recorder.capture('external_api_call')
def get_user_profile_from_api(user_id):
    # Symuluj wywołanie do external API
    time.sleep(random.uniform(0.2, 0.8))

    try:
        # Dodaj annotations
        xray_recorder.current_subsegment().put_annotation('api_endpoint',
'user_profile')
        xray_recorder.current_subsegment().put_annotation('external_service',
'profile_api')

        # Symuluj API call
        response = requests.get(
            f'https://api.example.com/profiles/{user_id}',
            timeout=5
        )

        # Dodaj HTTP metadata
        xray_recorder.current_subsegment().put_metadata('http_status',
response.status_code)
        xray_recorder.current_subsegment().put_metadata('response_size',
len(response.content))

        return response.json()
    except requests.exceptions.RequestException as e:
        # Dodaj informacje o błędzie
        xray_recorder.current_subsegment().add_exception(e)
        return {'error': 'Profile service unavailable'}

@app.route('/api/upload', methods=['POST'])
@xray_recorder.capture('file_upload')
def upload_file():
    if 'file' not in request.files:
        return jsonify({'error': 'No file provided'}), 400

    file = request.files['file']

    # Dodaj annotations
    xray_recorder.current_subsegment().put_annotation('operation',
'file_upload')
    xray_recorder.current_subsegment().put_annotation('file_size',
len(file.read()))

```

```

file.seek(0) # Reset file pointer

try:
    # Upload do S3
    s3.upload_fileobj(
        file,
        'my-app-uploads',
        file.filename,
        ExtraArgs={'ServerSideEncryption': 'AES256'}
    )

    # Dodaj metadata o upload
    xray_recorder.current_subsegment().put_metadata('s3_bucket', 'my-app-uploads')
    xray_recorder.current_subsegment().put_metadata('s3_key',
file.filename)

    return jsonify({'message': 'File uploaded successfully'})
except Exception as e:
    xray_recorder.current_subsegment().add_exception(e)
    return jsonify({'error': 'Upload failed'}), 500

@app.route('/health')
def health():
    return jsonify({'status': 'healthy'})

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)

```

## X-Ray Service Map i analiza:

Python

```

# xray-analysis.py
import boto3
from datetime import datetime, timedelta

xray = boto3.client('xray', region_name='eu-west-1')

def get_service_statistics():
    """Pobierz statystyki usług"""

    end_time = datetime.utcnow()
    start_time = end_time - timedelta(hours=1)

    response = xray.get_service_graph(
        TimeRangeType='TimeRangeByStartTime',

```

```

        StartTime=start_time,
        EndTime=end_time
    )

    print("Service Statistics:")
    print("=====")

    for service in response['Services']:
        name = service.get('Name', 'Unknown')
        stats = service.get('SummaryStatistics', {})

        print(f"\nService: {name}")
        print(f"  Request Count: {stats.get('TotalCount', 0)}")
        print(f"  Error Count: {stats.get('ErrorStatistics',
        {}).get('TotalCount', 0)}")
        print(f"  Fault Count: {stats.get('FaultStatistics',
        {}).get('TotalCount', 0)}")
        print(f"  Response Time: {stats.get('TotalTime', 0):.2f}s")

        if stats.get('TotalCount', 0) > 0:
            avg_response_time = stats.get('TotalTime', 0) /
stats.get('TotalCount', 1)
            error_rate = (stats.get('ErrorStatistics', {}).get('TotalCount',
0) / stats.get('TotalCount', 1)) * 100
            print(f"  Avg Response Time: {avg_response_time:.3f}s")
            print(f"  Error Rate: {error_rate:.2f}%")

def get_trace_summaries():
    """Pobierz podsumowania trace'ów"""

    end_time = datetime.utcnow()
    start_time = end_time - timedelta(hours=1)

    response = xray.get_trace_summaries(
        TimeRangeType='TimeRangeByStartTime',
        StartTime=start_time,
        EndTime=end_time,
        FilterExpression='service("my-web-app") AND error'
    )

    print("\nError Traces:")
    print("=====")

    for trace in response['TraceSummaries']:
        trace_id = trace['Id']
        duration = trace.get('Duration', 0)
        error_count = len([s for s in trace.get('ServiceIds', []) if
s.get('ErrorStatistics', {}).get('TotalCount', 0) > 0])

```

```

print(f"\nTrace ID: {trace_id}")
print(f"  Duration: {duration:.3f}s")
print(f"  Services with errors: {error_count}")

# Pobierz szczegóły trace
trace_detail = xray.batch_get_traces(TraceIds=[trace_id])

for trace_data in trace_detail['Traces']:
    for segment in trace_data['Segments']:
        segment_doc = json.loads(segment['Document'])
        if segment_doc.get('error'):
            print(f"  Error in {segment_doc.get('name', 'Unknown')}:
{segment_doc.get('cause', {}).get('exceptions', [{}])[0].get('message',
'Unknown error')}")

def analyze_performance_bottlenecks():
    """Analizuj bottlenecki wydajności"""

    end_time = datetime.utcnow()
    start_time = end_time - timedelta(hours=1)

    # Znajdź najwolniejsze trace'y
    response = xray.get_trace_summaries(
        TimeRangeType='TimeRangeByStartTime',
        StartTime=start_time,
        EndTime=end_time,
        FilterExpression='service("my-web-app") AND duration > 2'
    )

    print("\nSlow Traces (>2s):")
    print("=====")

    for trace in sorted(response['TraceSummaries'], key=lambda x:
x.get('Duration', 0), reverse=True)[:10]:
        trace_id = trace['Id']
        duration = trace.get('Duration', 0)

        print(f"\nTrace ID: {trace_id}")
        print(f"  Duration: {duration:.3f}s")

        # Pobierz szczegóły trace
        trace_detail = xray.batch_get_traces(TraceIds=[trace_id])

        for trace_data in trace_detail['Traces']:
            for segment in trace_data['Segments']:
                segment_doc = json.loads(segment['Document'])
                subsegments = segment_doc.get('subsegments', [])

```

```

        # Znajdź najwolniejsze subsegmenty
        slow_subsegments = sorted(
            [(s.get('name', 'Unknown'), s.get('end_time', 0) -
             s.get('start_time', 0))
             for s in subsegments],
            key=lambda x: x[1], reverse=True
       )[:3]

        print(f"  Slowest operations:")
        for name, duration in slow_subsegments:
            print(f"    {name}: {duration:.3f}s")

if __name__ == "__main__":
    get_service_statistics()
    get_trace_summaries()
    analyze_performance_bottlenecks()

```

## AWS CloudTrail - audyt i compliance

AWS CloudTrail rejestruje wywołania API AWS dla konta, dostarczając szczegółowe logi aktywności użytkowników, ról i usług AWS. CloudTrail jest kluczowy dla bezpieczeństwa, compliance i troubleshootingu.

### Konfiguracja CloudTrail

Bash

```

# Utwórz S3 bucket dla CloudTrail logs
aws s3 mb s3://my-cloudtrail-logs-$(date +%s)

# Utwórz CloudTrail
aws cloudtrail create-trail \
  --name my-organization-trail \
  --s3-bucket-name my-cloudtrail-logs-123456789 \
  --include-global-service-events \
  --is-multi-region-trail \
  --enable-log-file-validation

# Uruchom logging
aws cloudtrail start-logging --name my-organization-trail

```

```
# Sprawdź status
aws cloudtrail get-trail-status --name my-organization-trail
```

## CloudTrail Insights:

Bash

```
# Włącz CloudTrail Insights
aws cloudtrail put-insight-selectors \
  --trail-name my-organization-trail \
  --insight-selectors '[
    {
      "InsightType": "ApiCallRateInsight"
    }
  ]'
```

## Analiza logów CloudTrail:

Python

```
# cloudtrail-analysis.py
import boto3
import json
from datetime import datetime, timedelta
from collections import defaultdict, Counter

cloudtrail = boto3.client('cloudtrail')
s3 = boto3.client('s3')

def analyze_api_calls():
    """Analizuj wywołania API"""

    end_time = datetime.utcnow()
    start_time = end_time - timedelta(hours=24)

    response = cloudtrail.lookup_events(
        StartTime=start_time,
        EndTime=end_time,
        MaxItems=1000
    )

    # Statystyki wywołań API
    api_calls = Counter()
    users = Counter()
    source_ips = Counter()
```

```

error_events = []

for event in response['Events']:
    event_name = event.get('EventName', 'Unknown')
    username = event.get('Username', 'Unknown')
    source_ip = event.get('SourceIPAddress', 'Unknown')

    api_calls[event_name] += 1
    users[username] += 1
    source_ips[source_ip] += 1

    # Sprawdź błędy
    if event.get('ErrorCode') or event.get('ErrorMessage'):
        error_events.append({
            'event_name': event_name,
            'username': username,
            'error_code': event.get('ErrorCode'),
            'error_message': event.get('ErrorMessage'),
            'timestamp': event.get('EventTime')
        })

print("CloudTrail Analysis Report")
print("=====")

print(f"\nTop 10 API Calls:")
for api_call, count in api_calls.most_common(10):
    print(f"  {api_call}: {count}")

print(f"\nTop 10 Users:")
for user, count in users.most_common(10):
    print(f"  {user}: {count}")

print(f"\nTop 10 Source IPs:")
for ip, count in source_ips.most_common(10):
    print(f"  {ip}: {count}")

if error_events:
    print(f"\nError Events ({len(error_events)}):")
    for error in error_events[:10]:
        print(f"  {error['timestamp']}: {error['event_name']} by {error['username']} - {error['error_code']}: {error['error_message']}")

def detect_suspicious_activity():
    """Wykryj podejrzaną aktywność"""

    end_time = datetime.utcnow()
    start_time = end_time - timedelta(hours=1)

```

```

# Szukaj podejrzanych wzorców
suspicious_events = [
    'CreateUser',
    'AttachUserPolicy',
    'CreateRole',
    'PutRolePolicy',
    'CreateAccessKey',
    'DeleteTrail',
    'StopLogging',
    'PutBucketPolicy'
]

response = cloudtrail.lookup_events(
    StartTime=start_time,
    EndTime=end_time,
    LookupAttributes=[
        {
            'AttributeKey': 'EventName',
            'AttributeValue': event_name
        } for event_name in suspicious_events
    ]
)

print("\nSuspicious Activity Detection:")
print("=====")

if response['Events']:
    for event in response['Events']:
        print(f"ALERT: {event['EventName']} by {event.get('Username',
'Unknown')} from {event.get('SourceIPAddress', 'Unknown')} at
{event['EventTime']}")
    else:
        print("No suspicious activity detected in the last hour.")

def compliance_report():
    """Generuj raport compliance"""

    end_time = datetime.utcnow()
    start_time = end_time - timedelta(days=7)

    # Sprawdź kluczowe zdarzenia compliance
    compliance_events = [
        'DeleteBucket',
        'DeleteDBInstance',
        'TerminateInstances',
        'DeleteVolume',
        'DeleteSnapshot'
    ]

```



```

print("\nCompliance Report (Last 7 days):")
print("=====")

for event_name in compliance_events:
    response = cloudtrail.lookup_events(
        StartTime=start_time,
        EndTime=end_time,
        LookupAttributes=[
            {
                'AttributeKey': 'EventName',
                'AttributeValue': event_name
            }
        ]
    )

    if response['Events']:
        print(f"\n{event_name} Events:")
        for event in response['Events']:
            resources = [r.get('ResourceName', 'Unknown') for r in
event.get('Resources', [])]
            print(f"    {event['EventTime']}: {event.get('Username',
'Unknown')} deleted {'|'.join(resources)}")

if __name__ == "__main__":
    analyze_api_calls()
    detect_suspicious_activity()
    compliance_report()

```

To kończy rozdział o monitoringu i loggingu w AWS. W następnym rozdziale przejdziemy do Kubernetes w AWS (EKS).

## 19. Wprowadzenie do Kubernetes i EKS

### Kubernetes - orkiestracja kontenerów

Kubernetes to open-source platforma do automatyzacji wdrażania, skalowania i zarządzania aplikacjami kontenerowymi. Kubernetes, często nazywany K8s, został pierwotnie opracowany przez Google i jest obecnie utrzymywany przez Cloud Native Computing Foundation (CNCF). W kontekście DevOps, Kubernetes stanowi kluczowy element nowoczesnej infrastruktury, umożliwiając efektywne zarządzanie aplikacjami w środowiskach produkcyjnych.

Kubernetes rozwiązuje fundamentalne wyzwania związane z uruchamianiem aplikacji kontenerowych na dużą skalę. Podczas gdy Docker umożliwia pakowanie aplikacji w kontenery, Kubernetes zapewnia platformę do orkiestracji tych kontenerów w klastrach składających się z wielu maszyn. Kubernetes automatyzuje zadania takie jak load balancing, service discovery, storage orchestration, automated rollouts i rollbacks, self-healing oraz secret i configuration management.

Architektura Kubernetes opiera się na modelu master-worker, gdzie control plane (master nodes) zarządza klastrem, a worker nodes uruchamiają aplikacje. Control plane składa się z kilku komponentów: API Server jako centralny punkt komunikacji, etcd jako distributed key-value store dla stanu klastra, Scheduler odpowiedzialny za przypisywanie podów do węzłów, oraz Controller Manager zarządzający różnymi kontrolerami klastra.

Worker nodes zawierają kubelet (agent komunikujący się z control plane), kube-proxy (zarządzający ruchem sieciowym), oraz container runtime (Docker, containerd, lub CRI-O). Ta architektura zapewnia wysoką dostępność, skalowalność i odporność na awarie.

## Amazon EKS - zarządzany Kubernetes

Amazon Elastic Kubernetes Service (EKS) to w pełni zarządzana usługa Kubernetes w AWS, która eliminuje złożoność związaną z instalacją, operowaniem i skalowaniem control plane Kubernetes. EKS automatycznie zarządza dostępnością i skalowalnością węzłów control plane w wielu strefach dostępności, zapewniając wysoką dostępność i bezpieczeństwo.

Kluczowe zalety EKS w porównaniu do self-managed Kubernetes obejmują automatyczne aktualizacje i patching control plane, integrację z usługami AWS (IAM, VPC, ELB, EBS, EFS), certyfikację zgodności z upstream Kubernetes, oraz wsparcie dla AWS Fargate umożliwiające uruchamianie podów bez zarządzania węzłami.

EKS integruje się głęboko z ekosystemem AWS, oferując native support dla AWS Load Balancer Controller, EBS CSI Driver, EFS CSI Driver, AWS VPC CNI, oraz AWS IAM Authenticator. Ta integracja umożliwia seamless wykorzystanie istniejącej infrastruktury AWS i narzędzi bezpieczeństwa.

## Konfiguracja EKS

## Krok 1: Przygotowanie środowiska

Bash

```
# Zainstaluj kubectl
curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
chmod +x kubectl
sudo mv kubectl /usr/local/bin/

# Sprawdź instalację
kubectl version --client

# Zainstaluj eksctl
curl --silent --location
"https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname
-s)_amd64.tar.gz" | tar xz -C /tmp
sudo mv /tmp/eksctl /usr/local/bin

# Sprawdź instalację
eksctl version

# Zainstaluj AWS CLI v2 (jeśli nie jest zainstalowany)
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
"awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install

# Skonfiguruj AWS credentials
aws configure
```

## Krok 2: Tworzenie klastra EKS

YAML

```
# cluster-config.yaml
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-eks-cluster
  region: eu-west-1
  version: "1.28"
  tags:
    Environment: production
    Project: my-web-app
```

ManagedBy: eksctl

#### # VPC Configuration

vpc:

cidr: "10.0.0.0/16"

nat:

gateway: Single # Single NAT gateway dla oszczędności kosztów w dev

clusterEndpoints:

privateAccess: true

publicAccess: true

publicAccessCIDRs: ["0.0.0.0/0"]

#### # IAM Configuration

iam:

withOIDC: true

serviceAccounts:

- metadata:

name: aws-load-balancer-controller

namespace: kube-system

wellKnownPolicies:

awsLoadBalancerController: true

- metadata:

name: ebs-csi-controller-sa

namespace: kube-system

wellKnownPolicies:

ebsCSISController: true

- metadata:

name: cluster-autoscaler

namespace: kube-system

wellKnownPolicies:

autoScaler: true

#### # Managed Node Groups

managedNodeGroups:

- name: worker-nodes

instanceType: t3.medium

minSize: 2

maxSize: 10

desiredCapacity: 3

volumeSize: 50

volumeType: gp3

amiFamily: AmazonLinux2

#### # Tagging

tags:

NodeGroup: worker-nodes

Environment: production

```
# SSH Access
ssh:
  allow: true
  publicKeyName: my-eks-keypair

# IAM
iam:
  attachPolicyARNs:
    - arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy
    - arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
    - arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
    - arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess
  withAddonPolicies:
    autoScaler: true
    ebs: true
    efs: true
    cloudWatch: true

# Fargate Profiles
fargateProfiles:
  - name: fargate-default
    selectors:
      - namespace: default
        labels:
          compute-type: fargate
      - namespace: kube-system
        labels:
          compute-type: fargate

# Add-ons
addons:
  - name: vpc-cni
    version: latest
    configurationValues: |-
      env:
        ENABLE_PREFIX_DELEGATION: "true"
        WARM_PREFIX_TARGET: "1"
  - name: coredns
    version: latest
  - name: kube-proxy
    version: latest
  - name: aws-ebs-csi-driver
    version: latest
    serviceAccountRoleARN: arn:aws:iam::ACCOUNT-ID:role/AmazonEKS_EBS_CSI_DriverRole

# CloudWatch Logging
cloudWatch:
```

```
clusterLogging:
  enable: ["api", "audit", "authenticator", "controllerManager",
"scheduler"]
  logRetentionInDays: 30
```

### Krok 3: Utworzenie klastra

Bash

```
# Utwórz klaster
eksctl create cluster -f cluster-config.yaml

# Sprawdź status klastra
eksctl get cluster --region eu-west-1

# Skonfiguruj kubectl context
aws eks update-kubeconfig --region eu-west-1 --name my-eks-cluster

# Sprawdź połączenie
kubectl get nodes
kubectl get pods --all-namespaces

# Sprawdź add-ons
kubectl get pods -n kube-system
```

## Konfiguracja dodatkowych komponentów

### AWS Load Balancer Controller:

Bash

```
# Pobierz IAM policy
curl -o iam_policy.json https://raw.githubusercontent.com/kubernetes-
sigs/aws-load-balancer-controller/v2.7.2/docs/install/iam_policy.json

# Utwórz IAM policy
aws iam create-policy \
  --policy-name AWSLoadBalancerControllerIAMPolicy \
  --policy-document file://iam_policy.json

# Zainstaluj AWS Load Balancer Controller przez Helm
helm repo add eks https://aws.github.io/eks-charts
helm repo update

helm install aws-load-balancer-controller eks/aws-load-balancer-controller \
```

```
-n kube-system \  
--set clusterName=my-eks-cluster \  
--set serviceAccount.create=false \  
--set serviceAccount.name=aws-load-balancer-controller
```

# Sprawdź instalację

```
kubectl get deployment -n kube-system aws-load-balancer-controller
```

## Cluster Autoscaler:

YAML

```
# cluster-autoscaler.yaml  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: cluster-autoscaler  
  namespace: kube-system  
  labels:  
    app: cluster-autoscaler  
spec:  
  selector:  
    matchLabels:  
      app: cluster-autoscaler  
  template:  
    metadata:  
      labels:  
        app: cluster-autoscaler  
      annotations:  
        prometheus.io/scrape: 'true'  
        prometheus.io/port: '8085'  
    spec:  
      serviceAccountName: cluster-autoscaler  
      containers:  
        - image: registry.k8s.io/autoscaling/cluster-autoscaler:v1.28.2  
          name: cluster-autoscaler  
          resources:  
            limits:  
              cpu: 100m  
              memory: 600Mi  
            requests:  
              cpu: 100m  
              memory: 600Mi  
          command:  
            - ./cluster-autoscaler  
            - --v=4  
            - --stderrthreshold=info
```

```

- --cloud-provider=aws
- --skip-nodes-with-local-storage=false
- --expander=least-waste
- --node-group-auto-discovery=asg:tag=k8s.io/cluster-
autoscaler/enabled,k8s.io/cluster-autoscaler/my-eks-cluster
- --balance-similar-node-groups
- --skip-nodes-with-system-pods=false
env:
- name: AWS_REGION
  value: eu-west-1
volumeMounts:
- name: ssl-certs
  mountPath: /etc/ssl/certs/ca-certificates.crt
  readOnly: true
  imagePullPolicy: "Always"
volumes:
- name: ssl-certs
  hostPath:
    path: "/etc/ssl/certs/ca-bundle.crt"
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: cluster-autoscaler
  labels:
    k8s-addon: cluster-autoscaler.addons.k8s.io
    k8s-app: cluster-autoscaler
rules:
- apiGroups: [""]
  resources: ["events", "endpoints"]
  verbs: ["create", "patch"]
- apiGroups: [""]
  resources: ["pods/eviction"]
  verbs: ["create"]
- apiGroups: [""]
  resources: ["pods/status"]
  verbs: ["update"]
- apiGroups: [""]
  resources: ["endpoints"]
  resourceName: ["cluster-autoscaler"]
  verbs: ["get", "update"]
- apiGroups: [""]
  resources: ["nodes"]
  verbs: ["watch", "list", "get", "update"]
- apiGroups: [""]
  resources: ["namespaces", "pods", "services", "replicationcontrollers",
"persistentvolumeclaims", "persistentvolumes"]
  verbs: ["watch", "list", "get"]

```



```

- apiGroups: ["extensions"]
  resources: ["replicasets", "daemonsets"]
  verbs: ["watch", "list", "get"]
- apiGroups: ["policy"]
  resources: ["poddisruptionbudgets"]
  verbs: ["watch", "list"]
- apiGroups: ["apps"]
  resources: ["statefulsets", "replicasets", "daemonsets"]
  verbs: ["watch", "list", "get"]
- apiGroups: ["storage.k8s.io"]
  resources: ["storageclasses", "csinodes", "csidrivers",
"csistoragecapacities"]
  verbs: ["watch", "list", "get"]
- apiGroups: ["batch", "extensions"]
  resources: ["jobs"]
  verbs: ["get", "list", "watch", "patch"]
- apiGroups: ["coordination.k8s.io"]
  resources: ["leases"]
  verbs: ["create"]
- apiGroups: ["coordination.k8s.io"]
  resourceNames: ["cluster-autoscaler"]
  resources: ["leases"]
  verbs: ["get", "update"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: cluster-autoscaler
  namespace: kube-system
  labels:
    k8s-addon: cluster-autoscaler.addons.k8s.io
    k8s-app: cluster-autoscaler
rules:
- apiGroups: [""]
  resources: ["configmaps"]
  verbs: ["create","list","watch"]
- apiGroups: [""]
  resources: ["configmaps"]
  resourceNames: ["cluster-autoscaler-status", "cluster-autoscaler-priority-
expander"]
  verbs: ["delete", "get", "update", "watch"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: cluster-autoscaler
  labels:
    k8s-addon: cluster-autoscaler.addons.k8s.io

```

```
k8s-app: cluster-autoscaler
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-autoscaler
subjects:
- kind: ServiceAccount
  name: cluster-autoscaler
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: cluster-autoscaler
  namespace: kube-system
  labels:
    k8s-addon: cluster-autoscaler.addons.k8s.io
    k8s-app: cluster-autoscaler
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: cluster-autoscaler
subjects:
- kind: ServiceAccount
  name: cluster-autoscaler
  namespace: kube-system
```

#### Bash

```
# Zastosuj konfigurację
kubectl apply -f cluster-autoscaler.yaml

# Sprawdź logi
kubectl logs -f deployment/cluster-autoscaler -n kube-system
```

## Deployment aplikacji na EKS

### Przygotowanie aplikacji

#### Dockerfile dla aplikacji:

Plain Text

```

# Dockerfile
FROM python:3.11-slim

WORKDIR /app

# Zainstaluj zależności systemowe
RUN apt-get update && apt-get install -y \
    gcc \
    && rm -rf /var/lib/apt/lists/*

# Skopiuj requirements i zainstaluj zależności Python
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Skopiuj kod aplikacji
COPY . .

# Utwórz non-root user
RUN useradd --create-home --shell /bin/bash app \
    && chown -R app:app /app
USER app

# Health check
HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \
    CMD curl -f http://localhost:5000/health || exit 1

EXPOSE 5000

CMD ["unicorn", "--bind", "0.0.0.0:5000", "--workers", "4", "--timeout",
    "120", "app:app"]

```

## Aplikacja Flask z health checks:

Python

```

# app.py
from flask import Flask, jsonify, request
import os
import time
import logging
import psutil
from datetime import datetime

# Konfiguracja logowania
logging.basicConfig(
    level=logging.INFO,

```

```

        format='%(%asctime)s - %(name)s - %(levelname)s - %(message)s'
    )
    logger = logging.getLogger(__name__)

    app = Flask(__name__)

    # Konfiguracja z environment variables
    app.config.update(
        DEBUG=os.environ.get('DEBUG', 'False').lower() == 'true',
        SECRET_KEY=os.environ.get('SECRET_KEY', 'dev-secret-key'),
        DATABASE_URL=os.environ.get('DATABASE_URL', 'sqlite:///app.db'),
        REDIS_URL=os.environ.get('REDIS_URL', 'redis://localhost:6379/0')
    )

    @app.route('/')
    def home():
        """Główna strona aplikacji"""
        return jsonify({
            'message': 'Hello from Kubernetes!',
            'version': os.environ.get('APP_VERSION', '1.0.0'),
            'environment': os.environ.get('ENVIRONMENT', 'development'),
            'pod_name': os.environ.get('HOSTNAME', 'unknown'),
            'timestamp': datetime.utcnow().isoformat()
        })

    @app.route('/health')
    def health():
        """Health check endpoint"""
        try:
            # Sprawdź podstawowe metryki systemu
            cpu_percent = psutil.cpu_percent(interval=1)
            memory = psutil.virtual_memory()

            # Sprawdź czy aplikacja jest w dobrym stanie
            if cpu_percent > 90:
                return jsonify({
                    'status': 'unhealthy',
                    'reason': 'High CPU usage',
                    'cpu_percent': cpu_percent
                }), 503

            if memory.percent > 90:
                return jsonify({
                    'status': 'unhealthy',
                    'reason': 'High memory usage',
                    'memory_percent': memory.percent
                }), 503

```

```

        return jsonify({
            'status': 'healthy',
            'timestamp': datetime.utcnow().isoformat(),
            'uptime': time.time() - start_time,
            'cpu_percent': cpu_percent,
            'memory_percent': memory.percent,
            'pod_name': os.environ.get('HOSTNAME', 'unknown')
        })

    except Exception as e:
        logger.error(f"Health check failed: {e}")
        return jsonify({
            'status': 'unhealthy',
            'reason': str(e)
        }), 503

@app.route('/ready')
def ready():
    """Readiness probe endpoint"""
    try:
        # Sprawdź czy aplikacja jest gotowa do obsługi ruchu
        # Tutaj można dodać sprawdzenia połączeń z bazą danych, cache, etc.

        return jsonify({
            'status': 'ready',
            'timestamp': datetime.utcnow().isoformat(),
            'pod_name': os.environ.get('HOSTNAME', 'unknown')
        })

    except Exception as e:
        logger.error(f"Readiness check failed: {e}")
        return jsonify({
            'status': 'not ready',
            'reason': str(e)
        }), 503

@app.route('/metrics')
def metrics():
    """Endpoint dla metryk Prometheus"""
    try:
        cpu_percent = psutil.cpu_percent()
        memory = psutil.virtual_memory()

        metrics_data = f"""
# HELP app_cpu_usage_percent CPU usage percentage
# TYPE app_cpu_usage_percent gauge
app_cpu_usage_percent {cpu_percent}

```

```

# HELP app_memory_usage_percent Memory usage percentage
# TYPE app_memory_usage_percent gauge
app_memory_usage_percent {memory.percent}

# HELP app_uptime_seconds Application uptime in seconds
# TYPE app_uptime_seconds counter
app_uptime_seconds {time.time() - start_time}

# HELP app_info Application information
# TYPE app_info gauge
app_info{{version="{os.environ.get('APP_VERSION', '1.0.0')}",environment="
{os.environ.get('ENVIRONMENT', 'development')}}}} 1
"""

    return metrics_data, 200, {'Content-Type': 'text/plain; charset=utf-
8'}

except Exception as e:
    logger.error(f"Metrics endpoint failed: {e}")
    return "Error generating metrics", 500

@app.route('/api/data')
def get_data():
    """API endpoint z przykładowymi danymi"""
    return jsonify({
        'data': [
            {'id': 1, 'name': 'Item 1', 'value': 100},
            {'id': 2, 'name': 'Item 2', 'value': 200},
            {'id': 3, 'name': 'Item 3', 'value': 300}
        ],
        'total': 3,
        'pod_name': os.environ.get('HOSTNAME', 'unknown')
    })

@app.errorhandler(404)
def not_found(error):
    return jsonify({'error': 'Not found'}), 404

@app.errorhandler(500)
def internal_error(error):
    logger.error(f"Internal server error: {error}")
    return jsonify({'error': 'Internal server error'}), 500

# Zapisz czas startu aplikacji
start_time = time.time()

if __name__ == '__main__':

```

```
port = int(os.environ.get('PORT', 5000))
app.run(host='0.0.0.0', port=port, debug=app.config['DEBUG'])
```

## Kubernetes manifests

### Deployment:

#### YAML

```
# deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-web-app
  namespace: default
  labels:
    app: my-web-app
    version: v1
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  selector:
    matchLabels:
      app: my-web-app
  template:
    metadata:
      labels:
        app: my-web-app
        version: v1
    annotations:
      prometheus.io/scrape: "true"
      prometheus.io/port: "5000"
      prometheus.io/path: "/metrics"
    spec:
      serviceAccountName: my-web-app-sa
      securityContext:
        runAsNonRoot: true
        runAsUser: 1000
        fsGroup: 1000
      containers:
        - name: web-app
          image: 123456789012.dkr.ecr.eu-west-1.amazonaws.com/my-web-app:latest
```

```
imagePullPolicy: Always
ports:
- containerPort: 5000
  name: http
  protocol: TCP
env:
- name: ENVIRONMENT
  value: "production"
- name: APP_VERSION
  value: "1.0.0"
- name: DATABASE_URL
  valueFrom:
    secretKeyRef:
      name: app-secrets
      key: database-url
- name: SECRET_KEY
  valueFrom:
    secretKeyRef:
      name: app-secrets
      key: secret-key
resources:
  requests:
    memory: "256Mi"
    cpu: "250m"
  limits:
    memory: "512Mi"
    cpu: "500m"
livenessProbe:
  httpGet:
    path: /health
    port: 5000
  initialDelaySeconds: 30
  periodSeconds: 10
  timeoutSeconds: 5
  failureThreshold: 3
readinessProbe:
  httpGet:
    path: /ready
    port: 5000
  initialDelaySeconds: 5
  periodSeconds: 5
  timeoutSeconds: 3
  failureThreshold: 3
volumeMounts:
- name: app-config
  mountPath: /app/config
  readOnly: true
- name: tmp
```



```

        mountPath: /tmp
volumes:
- name: app-config
  configMap:
    name: app-config
- name: tmp
  emptyDir: {}
affinity:
  podAntiAffinity:
    preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 100
        podAffinityTerm:
          labelSelector:
            matchExpressions:
              - key: app
                operator: In
                values:
                  - my-web-app
          topologyKey: kubernetes.io/hostname
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: my-web-app-sa
  namespace: default
  annotations:
    eks.amazonaws.com/role-arn: arn:aws:iam::123456789012:role/MyWebAppRole
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
  namespace: default
data:
  app.conf: |
    [app]
    debug = false
    log_level = INFO

    [database]
    pool_size = 10
    max_overflow = 20

    [cache]
    ttl = 3600
---
apiVersion: v1
kind: Secret

```

```
metadata:
  name: app-secrets
  namespace: default
type: Opaque
data:
  database-url: cG9zdGdyZXNxbDovL3VzZXI6cGFzc0BkYi5leGFtcGxllmNvbS9teWRi  #
base64 encoded
  secret-key: bXktdjc2VjcmV0LWtleS1mb3ItcHJvZHVjdGlvbg==  # base64 encoded
```

## Service i Ingress:

### YAML

```
# service.yaml
apiVersion: v1
kind: Service
metadata:
  name: my-web-app-service
  namespace: default
  labels:
    app: my-web-app
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: "nlb"
    service.beta.kubernetes.io/aws-load-balancer-cross-zone-load-balancing-
enabled: "true"
spec:
  type: ClusterIP
  ports:
    - port: 80
      targetPort: 5000
      protocol: TCP
      name: http
  selector:
    app: my-web-app
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-web-app-ingress
  namespace: default
  annotations:
    kubernetes.io/ingress.class: alb
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/healthcheck-path: /health
    alb.ingress.kubernetes.io/healthcheck-interval-seconds: '30'
    alb.ingress.kubernetes.io/healthcheck-timeout-seconds: '5'
```

```

alb.ingress.kubernetes.io/healthy-threshold-count: '2'
alb.ingress.kubernetes.io/unhealthy-threshold-count: '3'
alb.ingress.kubernetes.io/ssl-redirect: '443'
alb.ingress.kubernetes.io/certificate-arn: arn:aws:acm:eu-west-
1:123456789012:certificate/12345678-1234-1234-1234-123456789012
alb.ingress.kubernetes.io/listen-ports: '[{"HTTP": 80}, {"HTTPS": 443}]'
spec:
  rules:
  - host: myapp.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: my-web-app-service
            port:
              number: 80

```

## HorizontalPodAutoscaler:

YAML

```

# hpa.yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: my-web-app-hpa
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: my-web-app
  minReplicas: 3
  maxReplicas: 20
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
      target:

```

```
    type: Utilization
    averageUtilization: 80
behavior:
  scaleDown:
    stabilizationWindowSeconds: 300
    policies:
      - type: Percent
        value: 10
        periodSeconds: 60
  scaleUp:
    stabilizationWindowSeconds: 60
    policies:
      - type: Percent
        value: 50
        periodSeconds: 60
      - type: Pods
        value: 2
        periodSeconds: 60
    selectPolicy: Max
```

To kończy wprowadzenie do Kubernetes i EKS. W następnym rozdziale omówimy zaawansowane funkcje EKS i zarządzanie klastrem.

## 20. Zaawansowane funkcje EKS i zarządzanie klastrem

### Monitoring i observability w EKS

Efektywne monitorowanie klastra Kubernetes jest kluczowe dla zapewnienia wysokiej dostępności, wydajności i bezpieczeństwa aplikacji. EKS oferuje natywną integrację z CloudWatch, ale dla kompleksowego monitorowania zaleca się implementację stack'u składającego się z Prometheus, Grafana, Jaeger i innych narzędzi CNCF.

### Prometheus i Grafana

#### Instalacja Prometheus Operator:

```
Bash
```

```

# Dodaj Helm repository
helm repo add prometheus-community https://prometheus-
community.github.io/helm-charts
helm repo update

# Utwórz namespace dla monitorowania
kubectl create namespace monitoring

# Zainstaluj kube-prometheus-stack
helm install prometheus prometheus-community/kube-prometheus-stack \
  --namespace monitoring \
  --set
prometheus.prometheusSpec.storageSpec.volumeClaimTemplate.spec.storageClassNa
me=gp2 \
  --set
prometheus.prometheusSpec.storageSpec.volumeClaimTemplate.spec.resources.requ
ests.storage=50Gi \
  --set grafana.persistence.enabled=true \
  --set grafana.persistence.storageClassName=gp2 \
  --set grafana.persistence.size=10Gi \
  --set grafana.adminPassword=admin123 \
  --set
alertmanager.alertmanagerSpec.storage.volumeClaimTemplate.spec.storageClassNa
me=gp2 \
  --set
alertmanager.alertmanagerSpec.storage.volumeClaimTemplate.spec.resources.requ
ests.storage=10Gi

```

## Custom ServiceMonitor dla aplikacji:

YAML

```

# app-servicemonitor.yaml
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: my-web-app-monitor
  namespace: monitoring
  labels:
    app: my-web-app
    release: prometheus
spec:
  selector:
    matchLabels:
      app: my-web-app
  namespaceSelector:

```

```
    matchNames:
      - default
  endpoints:
    - port: http
      path: /metrics
      interval: 30s
      scrapeTimeout: 10s
  ---
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: my-web-app-rules
  namespace: monitoring
  labels:
    app: my-web-app
    release: prometheus
spec:
  groups:
    - name: my-web-app.rules
      rules:
        - alert: HighErrorRate
          expr: |
            (
              rate(flask_http_request_exceptions_total[5m]) /
              rate(flask_http_request_total[5m])
            ) > 0.1
          for: 5m
          labels:
            severity: warning
            service: my-web-app
          annotations:
            summary: "High error rate detected"
            description: "Error rate is {{ $value | humanizePercentage }} for {{
$labels.instance }}"

        - alert: HighResponseTime
          expr: |
            histogram_quantile(0.95,
rate(flask_http_request_duration_seconds_bucket[5m])) > 2
          for: 5m
          labels:
            severity: warning
            service: my-web-app
          annotations:
            summary: "High response time detected"
            description: "95th percentile response time is {{ $value }}s for {{
$labels.instance }}"
```

```

- alert: PodCrashLooping
  expr: |
    rate(kube_pod_container_status_restarts_total[15m]) > 0
  for: 5m
  labels:
    severity: critical
    service: my-web-app
  annotations:
    summary: "Pod is crash looping"
    description: "Pod {{ $labels.pod }} in namespace {{ $labels.namespace }} is crash looping"

- alert: HighMemoryUsage
  expr: |
    (
      container_memory_working_set_bytes{pod=~"my-web-app-.*"} /
      container_spec_memory_limit_bytes{pod=~"my-web-app-.*"}
    ) > 0.9
  for: 5m
  labels:
    severity: warning
    service: my-web-app
  annotations:
    summary: "High memory usage"
    description: "Memory usage is {{ $value | humanizePercentage }} for {{ $labels.pod }}"

```

## Grafana Dashboard dla EKS:

JSON

```

{
  "dashboard": {
    "id": null,
    "title": "EKS Cluster Overview",
    "tags": ["kubernetes", "eks"],
    "timezone": "browser",
    "panels": [
      {
        "id": 1,
        "title": "Cluster CPU Usage",
        "type": "stat",
        "targets": [
          {
            "expr": "100 - (avg(irate(node_cpu_seconds_total{mode=\"idle\"}[5m])) * 100)",
            "legendFormat": "CPU Usage %"

```

```

    }
  ],
  "fieldConfig": {
    "defaults": {
      "unit": "percent",
      "min": 0,
      "max": 100,
      "thresholds": {
        "steps": [
          {"color": "green", "value": null},
          {"color": "yellow", "value": 70},
          {"color": "red", "value": 90}
        ]
      }
    }
  }
},
{
  "id": 2,
  "title": "Cluster Memory Usage",
  "type": "stat",
  "targets": [
    {
      "expr": "100 * (1 -
((avg_over_time(node_memory_MemFree_bytes[10m]) +
avg_over_time(node_memory_Cached_bytes[10m]) +
avg_over_time(node_memory_Buffers_bytes[10m])) /
avg_over_time(node_memory_MemTotal_bytes[10m])))",
      "legendFormat": "Memory Usage %"
    }
  ]
},
{
  "id": 3,
  "title": "Pod Status",
  "type": "piechart",
  "targets": [
    {
      "expr": "sum by (phase) (kube_pod_status_phase)",
      "legendFormat": "{{ phase }}"
    }
  ]
},
{
  "id": 4,
  "title": "Application Response Time",
  "type": "graph",
  "targets": [

```



```

    {
      "expr": "histogram_quantile(0.50,
rate(flask_http_request_duration_seconds_bucket[5m]))",
      "legendFormat": "50th percentile"
    },
    {
      "expr": "histogram_quantile(0.95,
rate(flask_http_request_duration_seconds_bucket[5m]))",
      "legendFormat": "95th percentile"
    },
    {
      "expr": "histogram_quantile(0.99,
rate(flask_http_request_duration_seconds_bucket[5m]))",
      "legendFormat": "99th percentile"
    }
  ]
}
],
"time": {
  "from": "now-1h",
  "to": "now"
},
"refresh": "30s"
}
}

```

## Logging z Fluent Bit

### Konfiguracja Fluent Bit dla EKS:

#### YAML

```

# fluent-bit-config.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: fluent-bit-config
  namespace: amazon-cloudwatch
data:
  fluent-bit.conf: |
    [SERVICE]
      Flush                    5
      Grace                   30
      Log_Level               info
      Daemon                  off
      Parsers_File             parsers.conf

```

```
HTTP_Server      On
HTTP_Listen      0.0.0.0
HTTP_Port        2020
storage.path     /var/fluent-bit/state/flb-storage/
storage.sync     normal
storage.checksum  off
storage.backlog.mem_limit 5M
```

```
@INCLUDE application-log.conf
```

```
@INCLUDE dataplane-log.conf
```

```
@INCLUDE host-log.conf
```

```
application-log.conf: |
```

```
[INPUT]
```

```
  Name          tail
  Tag            application.*
  Exclude_Path   /var/log/containers/cloudwatch-agent*,
/var/log/containers/fluent-bit*, /var/log/containers/aws-node*,
/var/log/containers/kube-proxy*
  Path          /var/log/containers/*.log
  multiline.parser  docker, cri
  DB            /var/fluent-bit/state/flb_container.db
  Mem_Buf_Limit 50MB
  Skip_Long_Lines On
  Refresh_Interval 10
  Rotate_Wait   30
  storage.type  filesystem
  Read_from_Head ${READ_FROM_HEAD}
```

```
[FILTER]
```

```
  Name          kubernetes
  Match          application.*
  Kube_URL       https://kubernetes.default.svc:443
  Kube_CA_File   /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
  Kube_Token_File /var/run/secrets/kubernetes.io/serviceaccount/token
  Kube_Tag_Prefix application.var.log.containers.
  Merge_Log      On
  Merge_Log_Key  log_processed
  K8S-Logging.Parser On
  K8S-Logging.Exclude Off
  Labels         Off
  Annotations    Off
```

```
[OUTPUT]
```

```
  Name          cloudwatch_logs
  Match          application.*
```

```

    region                ${AWS_REGION}
    log_group_name
/aws/containerinsights/${CLUSTER_NAME}/application
    log_stream_prefix     ${HOST_NAME}-
    auto_create_group     true
    extra_user_agent      container-insights

dataplane-log.conf: |
  [INPUT]
    Name                  systemd
    Tag                   dataplane.systemd.*
    Systemd_Filter        _SYSTEMD_UNIT=docker.service
    Systemd_Filter        _SYSTEMD_UNIT=kubelet.service
    DB                    /var/fluent-bit/state/systemd.db
    Path                  /var/log/journal
    Read_From_Tail        ${READ_FROM_TAIL}

  [FILTER]
    Name                  modify
    Match                 dataplane.systemd.*
    Rename                _HOSTNAME                hostname
    Rename                _SYSTEMD_UNIT            systemd_unit
    Rename                MESSAGE                message
    Remove_regex          ^((?!hostname|systemd_unit|message).)*$

  [OUTPUT]
    Name                  cloudwatch_logs
    Match                 dataplane.systemd.*
    region                ${AWS_REGION}
    log_group_name        /aws/containerinsights/${CLUSTER_NAME}/dataplane
    log_stream_prefix     ${HOST_NAME}-
    auto_create_group     true
    extra_user_agent      container-insights

host-log.conf: |
  [INPUT]
    Name                  tail
    Tag                   host.dmesg
    Path                  /var/log/dmesg
    Key                   message
    DB                    /var/fluent-bit/state/dmesg.db
    Mem_Buf_Limit         5MB
    Skip_Long_Lines       On
    Refresh_Interval      10

  [OUTPUT]
    Name                  cloudwatch_logs
    Match                 host.*

```

```

    region                ${AWS_REGION}
    log_group_name         /aws/containerinsights/${CLUSTER_NAME}/host
    log_stream_prefix      ${HOST_NAME}-
    auto_create_group      true
    extra_user_agent       container-insights

parsers.conf: |
  [PARSER]
    Name                  docker
    Format                 json
    Time_Key              time
    Time_Format           %Y-%m-%dT%H:%M:%S.%LZ

  [PARSER]
    Name                  cri
    Format                 regex
    Regex                 ^(?<time>[^\ ]+) (?<stream>stdout|stderr) (?
<logtag>[^\ ]*) (?<message>.*)$
    Time_Key              time
    Time_Format           %Y-%m-%dT%H:%M:%S.%L%Z
  ---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluent-bit
  namespace: amazon-cloudwatch
  labels:
    k8s-app: fluent-bit
spec:
  selector:
    matchLabels:
      k8s-app: fluent-bit
  template:
    metadata:
      labels:
        k8s-app: fluent-bit
    spec:
      serviceAccountName: fluent-bit
      tolerations:
        - key: node-role.kubernetes.io/master
          operator: Exists
          effect: NoSchedule
        - operator: "Exists"
          effect: "NoExecute"
        - operator: "Exists"
          effect: "NoSchedule"
      containers:
        - name: fluent-bit

```

```
image: public.ecr.aws/aws-observability/aws-for-fluent-bit:stable
imagePullPolicy: Always
env:
- name: AWS_REGION
  valueFrom:
    configMapKeyRef:
      name: fluent-bit-cluster-info
      key: logs.region
- name: CLUSTER_NAME
  valueFrom:
    configMapKeyRef:
      name: fluent-bit-cluster-info
      key: cluster.name
- name: HTTP_SERVER
  value: "On"
- name: HTTP_PORT
  value: "2020"
- name: READ_FROM_HEAD
  value: "Off"
- name: READ_FROM_TAIL
  value: "On"
- name: HOST_NAME
  valueFrom:
    fieldRef:
      fieldPath: spec.nodeName
- name: CI_VERSION
  value: "k8s/1.3.9"
resources:
  limits:
    memory: 200Mi
  requests:
    cpu: 500m
    memory: 100Mi
volumeMounts:
- name: fluentbitstate
  mountPath: /var/fluent-bit/state
- name: varlog
  mountPath: /var/log
  readOnly: true
- name: varlibdockercontainers
  mountPath: /var/lib/docker/containers
  readOnly: true
- name: fluent-bit-config
  mountPath: /fluent-bit/etc/
- name: runlogjournal
  mountPath: /run/log/journal
  readOnly: true
- name: dmesg
```

```
    mountPath: /var/log/dmesg
    readOnly: true
  terminationGracePeriodSeconds: 10
  hostNetwork: true
  dnsPolicy: ClusterFirstWithHostNet
  volumes:
  - name: fluentbitstate
    hostPath:
      path: /var/fluent-bit/state
  - name: varlog
    hostPath:
      path: /var/log
  - name: varlibdockercontainers
    hostPath:
      path: /var/lib/docker/containers
  - name: fluent-bit-config
    configMap:
      name: fluent-bit-config
  - name: runlogjournal
    hostPath:
      path: /run/log/journal
  - name: dmesg
    hostPath:
      path: /var/log/dmesg
```

## Bezpieczeństwo w EKS

### Pod Security Standards

#### Pod Security Policy (deprecated) vs Pod Security Standards:

##### YAML

```
# pod-security-policy.yaml (deprecated w Kubernetes 1.25+)
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: restricted
spec:
  privileged: false
  allowPrivilegeEscalation: false
  requiredDropCapabilities:
    - ALL
  volumes:
```

```

- 'configMap'
- 'emptyDir'
- 'projected'
- 'secret'
- 'downwardAPI'
- 'persistentVolumeClaim'
runAsUser:
  rule: 'MustRunAsNonRoot'
seLinux:
  rule: 'RunAsAny'
fsGroup:
  rule: 'RunAsAny'
---
# Nowe Pod Security Standards
apiVersion: v1
kind: Namespace
metadata:
  name: secure-namespcae
  labels:
    pod-security.kubernetes.io/enforce: restricted
    pod-security.kubernetes.io/audit: restricted
    pod-security.kubernetes.io/warn: restricted

```

## Network Policies:

YAML

```

# network-policy.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: my-web-app-netpol
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: my-web-app
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              name: ingress-nginx
        - podSelector:
            matchLabels:

```

```

        app: nginx-ingress
    ports:
      - protocol: TCP
        port: 5000
  - from:
      - namespaceSelector:
          matchLabels:
            name: monitoring
      - podSelector:
          matchLabels:
            app: prometheus
    ports:
      - protocol: TCP
        port: 5000
  egress:
  - to:
      - namespaceSelector:
          matchLabels:
            name: database
    ports:
      - protocol: TCP
        port: 5432
  - to: []
    ports:
      - protocol: TCP
        port: 443
      - protocol: TCP
        port: 53
      - protocol: UDP
        port: 53
  ---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-all-default
  namespace: default
spec:
  podSelector: {}
  policyTypes:
    - Ingress
    - Egress

```

## RBAC (Role-Based Access Control)

**Service Account z ograniczonymi uprawnieniami:**



## YAML

```
# rbac.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: my-web-app-sa
  namespace: default
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: my-web-app-role
rules:
- apiGroups: [""]
  resources: ["configmaps", "secrets"]
  verbs: ["get", "list"]
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list"]
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["get", "list"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: my-web-app-rolebinding
  namespace: default
subjects:
- kind: ServiceAccount
  name: my-web-app-sa
  namespace: default
roleRef:
  kind: Role
  name: my-web-app-role
  apiGroup: rbac.authorization.k8s.io
---
# ClusterRole dla cross-namespace access
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: node-reader
rules:
- apiGroups: [""]
  resources: ["nodes"]
  verbs: ["get", "list"]
- apiGroups: ["metrics.k8s.io"]
```

```
resources: ["nodes", "pods"]
verbs: ["get", "list"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: my-web-app-cluster-rolebinding
subjects:
- kind: ServiceAccount
  name: my-web-app-sa
  namespace: default
roleRef:
  kind: ClusterRole
  name: node-reader
  apiGroup: rbac.authorization.k8s.io
```

## Secrets Management z AWS Secrets Manager

### External Secrets Operator:

Bash

```
# Zainstaluj External Secrets Operator
helm repo add external-secrets https://charts.external-secrets.io
helm install external-secrets external-secrets/external-secrets -n external-
secrets-system --create-namespace

# Utwórz IAM role dla External Secrets
cat > external-secrets-trust-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::ACCOUNT-ID:oidc-provider/oidc.eks.eu-west-
1.amazonaws.com/id/OIDC-ID"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "oidc.eks.eu-west-1.amazonaws.com/id/OIDC-ID:sub":
"system:serviceaccount:external-secrets-system:external-secrets",
          "oidc.eks.eu-west-1.amazonaws.com/id/OIDC-ID:aud":
"sts.amazonaws.com"
        }
      }
    }
  ]
}
```

```

    }
  }
]
}
EOF

aws iam create-role \
  --role-name ExternalSecretsRole \
  --assume-role-policy-document file://external-secrets-trust-policy.json

aws iam attach-role-policy \
  --role-name ExternalSecretsRole \
  --policy-arn arn:aws:iam::aws:policy/SecretsManagerReadWrite

```

## YAML

```

# external-secrets.yaml
apiVersion: external-secrets.io/v1beta1
kind: SecretStore
metadata:
  name: aws-secrets-manager
  namespace: default
spec:
  provider:
    aws:
      service: SecretsManager
      region: eu-west-1
      auth:
        jwt:
          serviceAccountRef:
            name: external-secrets-sa
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: external-secrets-sa
  namespace: default
  annotations:
    eks.amazonaws.com/role-arn: arn:aws:iam::ACCOUNT-
ID:role/ExternalSecretsRole
---
apiVersion: external-secrets.io/v1beta1
kind: ExternalSecret
metadata:
  name: app-secrets
  namespace: default
spec:

```

```
refreshInterval: 15s
secretStoreRef:
  name: aws-secrets-manager
  kind: SecretStore
target:
  name: app-secrets
  creationPolicy: Owner
data:
- secretKey: database-url
  remoteRef:
    key: prod/myapp/database
    property: url
- secretKey: api-key
  remoteRef:
    key: prod/myapp/api
    property: key
```

## Networking w EKS

### AWS VPC CNI

#### Konfiguracja AWS VPC CNI:

YAML

```
# vpc-cni-config.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: amazon-vpc-cni
  namespace: kube-system
data:
  enable-windows-ipam: "false"
  enable-network-policy: "false"
  enable-pod-eni: "false"
  pod-security-group-enforcing-mode: "standard"
  disable-tcp-early-demux: "false"
  enable-bandwidth-plugin: "false"
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: aws-node
  namespace: kube-system
```

```
spec:
  template:
    spec:
      containers:
      - name: aws-node
        env:
          - name: ADDITIONAL_ENI_TAGS
            value: "Name=my-eks-cluster-eni,Environment=production"
          - name: AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG
            value: "false"
          - name: ENI_CONFIG_LABEL_DEF
            value: "topology.kubernetes.io/zone"
          - name: AWS_VPC_K8S_CNI_EXTERNALSNAT
            value: "false"
          - name: AWS_VPC_K8S_CNI_RANDOMIZESNAT
            value: "prng"
          - name: AWS_VPC_K8S_CNI_EXCLUDE_SNAT_CIDRS
            value: "10.0.0.0/8,172.16.0.0/12,192.168.0.0/16"
          - name: AWS_VPC_ENI_MTU
            value: "9001"
          - name: AWS_VPC_K8S_CNI_LOG_LEVEL
            value: "INFO"
          - name: WARM_ENI_TARGET
            value: "1"
          - name: WARM_IP_TARGET
            value: "10"
          - name: MINIMUM_IP_TARGET
            value: "5"
```

## Service Mesh z Istio

### Instalacja Istio:

Bash

```
# Pobierz Istio
curl -L https://istio.io/downloadIstio | sh -
cd istio-*
export PATH=$PWD/bin:$PATH

# Zainstaluj Istio
istioctl install --set values.defaultRevision=default

# Włącz automatic sidecar injection
kubectl label namespace default istio-injection=enabled
```

```
# Zainstaluj Istio addons
kubectl apply -f samples/addons/
```

## Istio Gateway i VirtualService:

### YAML

```
# istio-gateway.yaml
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: my-web-app-gateway
  namespace: default
spec:
  selector:
    istio: ingressgateway
  servers:
    - port:
        number: 80
        name: http
        protocol: HTTP
      hosts:
        - myapp.example.com
      tls:
        httpsRedirect: true
    - port:
        number: 443
        name: https
        protocol: HTTPS
      tls:
        mode: SIMPLE
        credentialName: myapp-tls-secret
      hosts:
        - myapp.example.com
  ---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: my-web-app-vs
  namespace: default
spec:
  hosts:
    - myapp.example.com
  gateways:
    - my-web-app-gateway
  http:
    - match:
```

```
- uri:
  prefix: /api/v1
route:
- destination:
  host: my-web-app-service
  port:
    number: 80
  weight: 90
- destination:
  host: my-web-app-service-canary
  port:
    number: 80
  weight: 10
fault:
  delay:
    percentage:
      value: 0.1
    fixedDelay: 5s
retries:
  attempts: 3
  perTryTimeout: 2s
- match:
  - uri:
    prefix: /
route:
- destination:
  host: my-web-app-service
  port:
    number: 80
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: my-web-app-dr
  namespace: default
spec:
  host: my-web-app-service
  trafficPolicy:
    connectionPool:
      tcp:
        maxConnections: 100
      http:
        http1MaxPendingRequests: 50
        maxRequestsPerConnection: 10
  loadBalancer:
    simple: LEAST_CONN
  outlierDetection:
    consecutiveErrors: 3
```

```
    interval: 30s
    baseEjectionTime: 30s
    maxEjectionPercent: 50
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
    labels:
      version: v2
```

## Backup i Disaster Recovery

### Velero dla backup

#### Instalacja Velero:

Bash

```
# Utwórz S3 bucket dla backup
aws s3 mb s3://my-eks-backup-bucket

# Utwórz IAM policy dla Velero
cat > velero-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeVolumes",
        "ec2:DescribeSnapshots",
        "ec2:CreateTags",
        "ec2:CreateVolume",
        "ec2:CreateSnapshot",
        "ec2>DeleteSnapshot"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:DeleteObject",
```



```

        "s3:PutObject",
        "s3:AbortMultipartUpload",
        "s3:ListMultipartUploadParts"
    ],
    "Resource": [
        "arn:aws:s3:::my-eks-backup-bucket/*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "s3:ListBucket"
    ],
    "Resource": [
        "arn:aws:s3:::my-eks-backup-bucket"
    ]
}
]
}
EOF

```

```

aws iam create-policy \
    --policy-name VeleroBackupPolicy \
    --policy-document file://velero-policy.json

```

```

# Pobierz Velero
wget https://github.com/vmware-tanzu/velero/releases/download/v1.12.1/velero-
v1.12.1-linux-amd64.tar.gz
tar -xvf velero-v1.12.1-linux-amd64.tar.gz
sudo mv velero-v1.12.1-linux-amd64/velero /usr/local/bin/

```

```

# Zainstaluj Velero w klastrze
velero install \
    --provider aws \
    --plugins velero/velero-plugin-for-aws:v1.8.1 \
    --bucket my-eks-backup-bucket \
    --backup-location-config region=eu-west-1 \
    --snapshot-location-config region=eu-west-1 \
    --secret-file ./credentials-velero

```

## Backup schedules:

YAML

```

# backup-schedule.yaml
apiVersion: velero.io/v1
kind: Schedule

```

```

metadata:
  name: daily-backup
  namespace: velero
spec:
  schedule: "0 2 * * *" # Codziennie o 2:00
  template:
    includedNamespaces:
      - default
      - monitoring
      - ingress-nginx
    excludedResources:
      - events
      - events.events.k8s.io
    storageLocation: default
    volumeSnapshotLocations:
      - default
    ttl: 720h0m0s # 30 dni
---
apiVersion: velero.io/v1
kind: Schedule
metadata:
  name: weekly-backup
  namespace: velero
spec:
  schedule: "0 1 * * 0" # Każdą niedzielę o 1:00
  template:
    includedNamespaces:
      - "*"
    storageLocation: default
    volumeSnapshotLocations:
      - default
    ttl: 2160h0m0s # 90 dni

```

## Restore procedures:

### Bash

```

# Lista backup
velero backup get

# Restore z backup
velero restore create --from-backup daily-backup-20240101-020000

# Restore konkretnego namespace
velero restore create my-restore \
  --from-backup daily-backup-20240101-020000 \
  --include-namespaces default

```

```
# Sprawdź status restore
velero restore describe my-restore
```

To kończy rozdział o zaawansowanych funkcjach EKS. W następnym rozdziale przygotujemy praktyczny projekt end-to-end.

## 21. Praktyczny projekt end-to-end

### Przegląd projektu

W tym rozdziale zbudujemy kompletny projekt DevOps, który integruje wszystkie omówione wcześniej technologie i narzędzia. Nasz projekt będzie obejmował aplikację webową e-commerce z mikrousługami, kompletną infrastrukturą AWS zarządzaną przez Terraform, pipeline CI/CD z Jenkins i AWS CodePipeline, deployment na EKS, oraz kompleksowy monitoring i logging.

Projekt składa się z następujących komponentów: frontend React, backend API w Python (Flask), mikrousługa zarządzania produktami, mikrousługa płatności, baza danych PostgreSQL, Redis cache, Elasticsearch dla wyszukiwania, oraz system kolejek z RabbitMQ. Całość będzie wdrożona na Amazon EKS z wykorzystaniem Istio service mesh, monitorowana przez Prometheus i Grafana, z logami centralizowanymi w CloudWatch.

Architektura projektu opiera się na nowoczesnych wzorcach DevOps i cloud-native development. Wykorzystujemy Infrastructure as Code z Terraform, GitOps workflow, automated testing na każdym poziomie, blue-green deployments, canary releases, oraz comprehensive observability. Projekt demonstruje real-world implementation wszystkich konceptów omówionych w poprzednich rozdziałach.

### Struktura projektu

Plain Text

```
ecommerce-devops-project/
├── infrastructure/
│   ├── terraform/
│   │   ├── modules/
│   │   └── vpc/
```

```

├── eks/
├── rds/
├── elasticache/
├── elasticsearch/
├── monitoring/
├── environments/
├── dev/
├── staging/
├── prod/
├── main.tf
├── kubernetes/
├── base/
├── overlays/
├── dev/
├── staging/
├── prod/
├── istio/
├── applications/
├── frontend/
├── src/
├── public/
├── Dockerfile
├── package.json
├── nginx.conf
├── api-gateway/
├── app/
├── tests/
├── Dockerfile
├── requirements.txt
├── gunicorn.conf.py
├── product-service/
├── app/
├── tests/
├── Dockerfile
├── requirements.txt
├── payment-service/
├── app/
├── tests/
├── Dockerfile
├── requirements.txt
├── user-service/
├── app/
├── tests/
├── Dockerfile
├── requirements.txt
├── ci-cd/
├── jenkins/
├── Jenkinsfile

```

```
| | | └─ pipelines/
| | |   └─ shared-libraries/
| | └─ github-actions/
| |   └─ workflows/
| └─ aws-codepipeline/
|   └─ buildspec.yml
└─ monitoring/
    └─ prometheus/
    └─ grafana/
    └─ alertmanager/
    └─ jaeger/
└─ scripts/
    └─ deploy.sh
    └─ test.sh
    └─ backup.sh
    └─ monitoring.py
└─ docs/
    └─ architecture.md
    └─ deployment.md
    └─ troubleshooting.md
```

## Aplikacja e-commerce

### Frontend React

**package.json:**

JSON

```
{
  "name": "ecommerce-frontend",
  "version": "1.0.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.16.4",
    "@testing-library/react": "^13.3.0",
    "@testing-library/user-event": "^13.5.0",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.3.0",
    "react-scripts": "5.0.1",
    "axios": "^0.27.2",
    "bootstrap": "^5.2.0",
    "react-bootstrap": "^2.5.0",
```

```

    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject",
    "test:coverage": "react-scripts test --coverage --watchAll=false"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  },
  "proxy": "http://localhost:5000"
}

```

## src/App.js:

### JavaScript

```

import React, { useState, useEffect } from 'react';
import { BrowserRouter as Router, Routes, Route, Link } from 'react-router-dom';
import { Navbar, Nav, Container, Alert } from 'react-bootstrap';
import axios from 'axios';
import ProductList from './components/ProductList';
import ProductDetail from './components/ProductDetail';
import Cart from './components/Cart';
import Checkout from './components/Checkout';
import UserProfile from './components/UserProfile';
import './App.css';

// Konfiguracja axios

```

```
axios.defaults.baseURL = process.env.REACT_APP_API_URL || '/api';
axios.defaults.timeout = 10000;

// Interceptor dla obsługi błędów
axios.interceptors.response.use(
  response => response,
  error => {
    console.error('API Error:', error);
    return Promise.reject(error);
  }
);

function App() {
  const [user, setUser] = useState(null);
  const [cart, setCart] = useState([]);
  const [notification, setNotification] = useState(null);
  const [healthStatus, setHealthStatus] = useState('unknown');

  useEffect(() => {
    // Sprawdź health status API
    checkApiHealth();

    // Załaduj dane użytkownika z localStorage
    const savedUser = localStorage.getItem('user');
    if (savedUser) {
      setUser(JSON.parse(savedUser));
    }

    // Załaduj koszyk z localStorage
    const savedCart = localStorage.getItem('cart');
    if (savedCart) {
      setCart(JSON.parse(savedCart));
    }
  }, []);

  const checkApiHealth = async () => {
    try {
      const response = await axios.get('/health');
      setHealthStatus(response.data.status);
    } catch (error) {
      setHealthStatus('unhealthy');
      showNotification('API is currently unavailable', 'danger');
    }
  };

  const showNotification = (message, type = 'info') => {
    setNotification({ message, type });
    setTimeout(() => setNotification(null), 5000);
  };
}
```

```

};

const addToCart = (product, quantity = 1) => {
  const existingItem = cart.find(item => item.id === product.id);
  let newCart;

  if (existingItem) {
    newCart = cart.map(item =>
      item.id === product.id
        ? { ...item, quantity: item.quantity + quantity }
        : item
    );
  } else {
    newCart = [...cart, { ...product, quantity }];
  }

  setCart(newCart);
  localStorage.setItem('cart', JSON.stringify(newCart));
  showNotification(`${product.name} added to cart`, 'success');
};

const removeFromCart = (productId) => {
  const newCart = cart.filter(item => item.id !== productId);
  setCart(newCart);
  localStorage.setItem('cart', JSON.stringify(newCart));
  showNotification('Item removed from cart', 'info');
};

const updateCartQuantity = (productId, quantity) => {
  if (quantity <= 0) {
    removeFromCart(productId);
    return;
  }

  const newCart = cart.map(item =>
    item.id === productId ? { ...item, quantity } : item
  );
  setCart(newCart);
  localStorage.setItem('cart', JSON.stringify(newCart));
};

const clearCart = () => {
  setCart([]);
  localStorage.removeItem('cart');
  showNotification('Cart cleared', 'info');
};

const login = async (credentials) => {

```



```

    try {
      const response = await axios.post('/auth/login', credentials);
      const userData = response.data.user;
      setUser(userData);
      localStorage.setItem('user', JSON.stringify(userData));
      localStorage.setItem('token', response.data.token);
      axios.defaults.headers.common['Authorization'] = `Bearer
${response.data.token}`;
      showNotification('Login successful', 'success');
      return true;
    } catch (error) {
      showNotification('Login failed', 'danger');
      return false;
    }
  };

  const logout = () => {
    setUser(null);
    localStorage.removeItem('user');
    localStorage.removeItem('token');
    delete axios.defaults.headers.common['Authorization'];
    showNotification('Logged out successfully', 'info');
  };

  const cartItemCount = cart.reduce((total, item) => total + item.quantity,
0);

  return (
    <Router>
      <div className="App">
        <Navbar bg="dark" variant="dark" expand="lg">
          <Container>
            <Navbar.Brand as={Link} to="/">
              E-Commerce Store
            </Navbar.Brand>
            <Navbar.Toggle aria-controls="basic-navbar-nav" />
            <Navbar.Collapse id="basic-navbar-nav">
              <Nav className="me-auto">
                <Nav.Link as={Link} to="/">Products</Nav.Link>
                <Nav.Link as={Link} to="/cart">
                  Cart ({cartItemCount})
                </Nav.Link>
              </Nav>
              <Nav>
                {user ? (
                  <>
                    <Nav.Link as={Link} to="/profile">
                      {user.name}

```

```

        </Nav.Link>
        <Nav.Link onClick={logout}>Logout</Nav.Link>
      </>
    ) : (
      <Nav.Link as={Link} to="/login">Login</Nav.Link>
    )}
    <Nav.Link>
      <span className={`health-indicator ${healthStatus}`}>
        API: {healthStatus}
      </span>
    </Nav.Link>
  </Nav>
</Navbar.Collapse>
</Container>
</Navbar>

<Container className="mt-4">
  {notification && (
    <Alert variant={notification.type} dismissible onClose={() =>
setNotification(null)}>
      {notification.message}
    </Alert>
  )}

  <Routes>
    <Route path="/" element={
      <ProductList addToCart={addToCart} />
    } />
    <Route path="/product/:id" element={
      <ProductDetail addToCart={addToCart} />
    } />
    <Route path="/cart" element={
      <Cart
        cart={cart}
        updateQuantity={updateCartQuantity}
        removeItem={removeFromCart}
        clearCart={clearCart}
      />
    } />
    <Route path="/checkout" element={
      <Checkout
        cart={cart}
        user={user}
        clearCart={clearCart}
        showNotification={showNotification}
      />
    } />
    <Route path="/profile" element={

```

```

        <UserProfile user={user} />
      } />
    </Routes>
  </Container>
</div>
</Router>
);
}

export default App;

```

## src/components/ProductList.js:

JavaScript

```

import React, { useState, useEffect } from 'react';
import { Row, Col, Card, Button, Form, Spinner, Alert } from 'react-bootstrap';
import { Link } from 'react-router-dom';
import axios from 'axios';

function ProductList({ addToCart }) {
  const [products, setProducts] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const [searchTerm, setSearchTerm] = useState('');
  const [category, setCategory] = useState('');
  const [sortBy, setSortBy] = useState('name');
  const [categories, setCategories] = useState([]);

  useEffect(() => {
    fetchProducts();
    fetchCategories();
  }, [searchTerm, category, sortBy]);

  const fetchProducts = async () => {
    try {
      setLoading(true);
      const params = {
        search: searchTerm,
        category: category,
        sort: sortBy
      };

      const response = await axios.get('/products', { params });
      setProducts(response.data.products);
      setError(null);
    }
  }

```

```

    } catch (err) {
      setError('Failed to fetch products');
      console.error('Error fetching products:', err);
    } finally {
      setLoading(false);
    }
  };

const fetchCategories = async () => {
  try {
    const response = await axios.get('/categories');
    setCategories(response.data.categories);
  } catch (err) {
    console.error('Error fetching categories:', err);
  }
};

const handleSearch = (e) => {
  e.preventDefault();
  fetchProducts();
};

if (loading) {
  return (
    <div className="text-center">
      <Spinner animation="border" role="status">
        <span className="visually-hidden">Loading...</span>
      </Spinner>
    </div>
  );
}

if (error) {
  return <Alert variant="danger">{error}</Alert>;
}

return (
  <div>
    <h1>Products</h1>

    { /* Search and Filter Controls */ }
    <Form onSubmit={handleSearch} className="mb-4">
      <Row>
        <Col md={4}>
          <Form.Control
            type="text"
            placeholder="Search products..."
            value={searchTerm}

```

```

        onChange={(e) => setSearchTerm(e.target.value)}
      />
    </Col>
    <Col md={3}>
      <Form.Select
        value={category}
        onChange={(e) => setCategory(e.target.value)}
      >
        <option value="">All Categories</option>
        {categories.map(cat => (
          <option key={cat.id} value={cat.id}>{cat.name}</option>
        ))}
      </Form.Select>
    </Col>
    <Col md={3}>
      <Form.Select
        value={sortBy}
        onChange={(e) => setSortBy(e.target.value)}
      >
        <option value="name">Sort by Name</option>
        <option value="price_asc">Price: Low to High</option>
        <option value="price_desc">Price: High to Low</option>
        <option value="rating">Rating</option>
      </Form.Select>
    </Col>
    <Col md={2}>
      <Button type="submit" variant="primary" className="w-100">
        Search
      </Button>
    </Col>
  </Row>
</Form>

{/* Products Grid */}
<Row>
  {products.map(product => (
    <Col key={product.id} md={4} className="mb-4">
      <Card className="h-100">
        <Card.Img
          variant="top"
          src={product.image_url || '/placeholder-image.jpg'}
          style={{ height: '200px', objectFit: 'cover' }}
        />
        <Card.Body className="d-flex flex-column">
          <Card.Title>
            <Link to={`/product/${product.id}`} className="text-decoration-none">
              {product.name}
            </Link>
          </Card.Title>
        </Card.Body>
      </Card>
    </Col>
  ))}
</Row>

```

```

        </Link>
      </Card.Title>
      <Card.Text className="flex-grow-1">
        {product.description}
      </Card.Text>
      <div className="mt-auto">
        <div className="d-flex justify-content-between align-items-
center mb-2">
          <span className="h5 mb-0">${product.price}</span>
          <span className="text-muted">
            ★ {product.rating || 'N/A'} ({product.review_count ||
0} reviews)
          </span>
        </div>
        <div className="d-flex justify-content-between align-items-
center">
          <span className={`badge ${product.stock > 0 ? 'bg-
success' : 'bg-danger'}`}>
            {product.stock > 0 ? `${product.stock} in stock` : 'Out
of stock'}
          </span>
          <Button
            variant="primary"
            size="sm"
            disabled={product.stock === 0}
            onClick={() => addToCart(product)}
          >
            Add to Cart
          </Button>
        </div>
      </div>
    </Card.Body>
  </Card>
</Col>
  )})
</Row>

{products.length === 0 && (
  <Alert variant="info">
    No products found. Try adjusting your search criteria.
  </Alert>
)}
</div>
);
}

export default ProductList;

```

## Dockerfile dla frontend:

Plain Text

```
# Multi-stage build dla React
FROM node:18-alpine as build

WORKDIR /app

# Skopiuj package files
COPY package*.json ./
RUN npm ci --only=production

# Skopiuj kod źródłowy i zbuduj aplikację
COPY . .
RUN npm run build

# Production stage z nginx
FROM nginx:alpine

# Skopiuj zbudowaną aplikację
COPY --from=build /app/build /usr/share/nginx/html

# Skopiuj konfigurację nginx
COPY nginx.conf /etc/nginx/nginx.conf

# Health check
HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \
    CMD wget --no-verbose --tries=1 --spider http://localhost:80/ || exit 1

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

## API Gateway (Python Flask)

### app/init.py:

Python

```
from flask import Flask, jsonify, request
from flask_cors import CORS
from flask_limiter import Limiter
from flask_limiter.util import get_remote_address
import os
```

```

import logging
import time
from datetime import datetime
import requests
from functools import wraps

# Konfiguracja logowania
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
)
logger = logging.getLogger(__name__)

def create_app():
    app = Flask(__name__)

    # Konfiguracja
    app.config.update(
        SECRET_KEY=os.environ.get('SECRET_KEY', 'dev-secret-key'),
        DEBUG=os.environ.get('DEBUG', 'False').lower() == 'true',

        # Service URLs
        PRODUCT_SERVICE_URL=os.environ.get('PRODUCT_SERVICE_URL',
'http://product-service:5001'),
        USER_SERVICE_URL=os.environ.get('USER_SERVICE_URL', 'http://user-
service:5002'),
        PAYMENT_SERVICE_URL=os.environ.get('PAYMENT_SERVICE_URL',
'http://payment-service:5003'),

        # Database
        DATABASE_URL=os.environ.get('DATABASE_URL',
'postgresql://user:pass@localhost/ecommerce'),

        # Redis
        REDIS_URL=os.environ.get('REDIS_URL', 'redis://localhost:6379/0'),

        # Rate limiting
        RATELIMIT_STORAGE_URL=os.environ.get('REDIS_URL',
'redis://localhost:6379/1'),
    )

    # CORS
    CORS(app, origins=['http://localhost:3000', 'https://*.example.com'])

    # Rate limiting
    limiter = Limiter(
        app,
        key_func=get_remote_address,

```



```

        default_limits=["1000 per hour", "100 per minute"]
    )

    # Middleware dla logowania requestów
    @app.before_request
    def log_request_info():
        logger.info(f'{request.method} {request.url} - {request.remote_addr}')
        request.start_time = time.time()

    @app.after_request
    def log_response_info(response):
        duration = time.time() - request.start_time
        logger.info(f'{request.method} {request.url} - {response.status_code} - {duration:.3f}s')

        # Dodaj headers dla CORS i security
        response.headers['X-Content-Type-Options'] = 'nosniff'
        response.headers['X-Frame-Options'] = 'DENY'
        response.headers['X-XSS-Protection'] = '1; mode=block'

        return response

    # Decorator dla circuit breaker
    def circuit_breaker(service_name, timeout=5, max_failures=3):
        def decorator(func):
            @wraps(func)
            def wrapper(*args, **kwargs):
                try:
                    return func(*args, **kwargs)
                except requests.exceptions.RequestException as e:
                    logger.error(f'Service {service_name} unavailable: {e}')
                    return jsonify({
                        'error': f'{service_name} temporarily unavailable',
                        'message': 'Please try again later'
                    }), 503
            return wrapper
        return decorator

    # Health check endpoint
    @app.route('/health')
    def health():
        health_status = {
            'status': 'healthy',
            'timestamp': datetime.utcnow().isoformat(),
            'version': os.environ.get('APP_VERSION', '1.0.0'),
            'services': {}
        }

```

```

# Sprawdź dostępność mikrousług
services = {
    'product-service': app.config['PRODUCT_SERVICE_URL'],
    'user-service': app.config['USER_SERVICE_URL'],
    'payment-service': app.config['PAYMENT_SERVICE_URL']
}

for service_name, service_url in services.items():
    try:
        response = requests.get(f'{service_url}/health', timeout=2)
        health_status['services'][service_name] = {
            'status': 'healthy' if response.status_code == 200 else
'unhealthy',
            'response_time': response.elapsed.total_seconds()
        }
    except Exception as e:
        health_status['services'][service_name] = {
            'status': 'unhealthy',
            'error': str(e)
        }

# Sprawdź czy wszystkie usługi są dostępne
unhealthy_services = [name for name, status in
health_status['services'].items()
                        if status['status'] == 'unhealthy']

if unhealthy_services:
    health_status['status'] = 'degraded'
    return jsonify(health_status), 503

return jsonify(health_status)

# Products endpoints
@app.route('/api/products')
@limiter.limit("50 per minute")
@circuit_breaker('product-service')
def get_products():
    params = {
        'search': request.args.get('search', ''),
        'category': request.args.get('category', ''),
        'sort': request.args.get('sort', 'name'),
        'page': request.args.get('page', 1),
        'per_page': request.args.get('per_page', 20)
    }

    response = requests.get(
        f"{app.config['PRODUCT_SERVICE_URL']}/products",

```

```

        params=params,
        timeout=10
    )
    response.raise_for_status()
    return jsonify(response.json())

@app.route('/api/products/<int:product_id>')
@limiter.limit("100 per minute")
@circuit_breaker('product-service')
def get_product(product_id):
    response = requests.get(
        f"{app.config['PRODUCT_SERVICE_URL']}/products/{product_id}",
        timeout=5
    )
    response.raise_for_status()
    return jsonify(response.json())

@app.route('/api/categories')
@limiter.limit("20 per minute")
@circuit_breaker('product-service')
def get_categories():
    response = requests.get(
        f"{app.config['PRODUCT_SERVICE_URL']}/categories",
        timeout=5
    )
    response.raise_for_status()
    return jsonify(response.json())

# User endpoints
@app.route('/api/auth/login', methods=['POST'])
@limiter.limit("5 per minute")
@circuit_breaker('user-service')
def login():
    response = requests.post(
        f"{app.config['USER_SERVICE_URL']}/auth/login",
        json=request.json,
        timeout=10
    )
    response.raise_for_status()
    return jsonify(response.json())

@app.route('/api/auth/register', methods=['POST'])
@limiter.limit("3 per minute")
@circuit_breaker('user-service')
def register():
    response = requests.post(
        f"{app.config['USER_SERVICE_URL']}/auth/register",
        json=request.json,

```

```

        timeout=10
    )
    response.raise_for_status()
    return jsonify(response.json())

@app.route('/api/users/profile')
@limiter.limit("20 per minute")
@circuit_breaker('user-service')
def get_profile():
    auth_header = request.headers.get('Authorization')
    if not auth_header:
        return jsonify({'error': 'Authorization header required'}), 401

    response = requests.get(
        f"{app.config['USER_SERVICE_URL']}/profile",
        headers={'Authorization': auth_header},
        timeout=5
    )
    response.raise_for_status()
    return jsonify(response.json())

# Payment endpoints
@app.route('/api/payments/process', methods=['POST'])
@limiter.limit("10 per minute")
@circuit_breaker('payment-service')
def process_payment():
    auth_header = request.headers.get('Authorization')
    if not auth_header:
        return jsonify({'error': 'Authorization header required'}), 401

    response = requests.post(
        f"{app.config['PAYMENT_SERVICE_URL']}/process",
        json=request.json,
        headers={'Authorization': auth_header},
        timeout=30
    )
    response.raise_for_status()
    return jsonify(response.json())

# Error handlers
@app.errorhandler(404)
def not_found(error):
    return jsonify({'error': 'Endpoint not found'}), 404

@app.errorhandler(500)
def internal_error(error):
    logger.error(f'Internal server error: {error}')
    return jsonify({'error': 'Internal server error'}), 500

```

```

@app.errorhandler(429)
def ratelimit_handler(e):
    return jsonify({'error': 'Rate limit exceeded', 'message':
str(e.description)}), 429

# Metrics endpoint dla Prometheus
@app.route('/metrics')
def metrics():
    # Tutaj można dodać custom metryki
    return "# Custom metrics would go here\n", 200, {'Content-Type':
'text/plain'}

return app

if __name__ == '__main__':
    app = create_app()
    port = int(os.environ.get('PORT', 5000))
    app.run(host='0.0.0.0', port=port, debug=app.config['DEBUG'])

```

## Product Service (Mikroustługa)

### app/models.py:

Python

```

from sqlalchemy import create_engine, Column, Integer, String, Float,
DateTime, Text, Boolean, ForeignKey
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker, relationship
from sqlalchemy.sql import func
import os

DATABASE_URL = os.environ.get('DATABASE_URL',
'postgres://user:pass@localhost/products')

engine = create_engine(DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()

class Category(Base):
    __tablename__ = 'categories'

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String(100), unique=True, nullable=False)
    description = Column(Text)

```

```

    created_at = Column(DateTime(timezone=True), server_default=func.now())
    updated_at = Column(DateTime(timezone=True), onupdate=func.now())

    products = relationship("Product", back_populates="category")

class Product(Base):
    __tablename__ = 'products'

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String(200), nullable=False, index=True)
    description = Column(Text)
    price = Column(Float, nullable=False)
    stock = Column(Integer, default=0)
    category_id = Column(Integer, ForeignKey('categories.id'))
    image_url = Column(String(500))
    rating = Column(Float, default=0.0)
    review_count = Column(Integer, default=0)
    is_active = Column(Boolean, default=True)
    created_at = Column(DateTime(timezone=True), server_default=func.now())
    updated_at = Column(DateTime(timezone=True), onupdate=func.now())

    category = relationship("Category", back_populates="products")

# Utwórz tabelę
Base.metadata.create_all(bind=engine)

def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()

```

## app/main.py:

Python

```

from flask import Flask, jsonify, request
from sqlalchemy.orm import Session
from sqlalchemy import or_, desc, asc
import os
import logging
import redis
import json
from datetime import datetime, timedelta
from models import get_db, Product, Category

```

```

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

app = Flask(__name__)

# Redis dla cache
redis_client = redis.from_url(os.environ.get('REDIS_URL',
'redis://localhost:6379/0'))

def cache_key(prefix, **kwargs):
    """Generuj klucz cache"""
    key_parts = [prefix] + [f"{k}:{v}" for k, v in sorted(kwargs.items()) if
v]
    return ":".join(key_parts)

def get_cached_or_fetch(key, fetch_func, ttl=300):
    """Pobierz z cache lub wykonaj funkcję i zapisz w cache"""
    try:
        cached = redis_client.get(key)
        if cached:
            return json.loads(cached)
    except Exception as e:
        logger.warning(f"Cache error: {e}")

    # Fetch fresh data
    data = fetch_func()

    # Save to cache
    try:
        redis_client.setex(key, ttl, json.dumps(data, default=str))
    except Exception as e:
        logger.warning(f"Cache save error: {e}")

    return data

@app.route('/health')
def health():
    try:
        # Sprawdź połączenie z bazą danych
        db = next(get_db())
        db.execute("SELECT 1")
        db_status = "healthy"
    except Exception as e:
        db_status = f"unhealthy: {e}"

    try:
        # Sprawdź Redis
        redis_client.ping()

```

```

        redis_status = "healthy"
    except Exception as e:
        redis_status = f"unhealthy: {e}"

    status = "healthy" if db_status == "healthy" and redis_status ==
"healthy" else "unhealthy"

    return jsonify({
        'status': status,
        'timestamp': datetime.utcnow().isoformat(),
        'database': db_status,
        'redis': redis_status,
        'service': 'product-service'
    })

@app.route('/products')
def get_products():
    search = request.args.get('search', '')
    category = request.args.get('category', '')
    sort = request.args.get('sort', 'name')
    page = int(request.args.get('page', 1))
    per_page = min(int(request.args.get('per_page', 20)), 100) # Max 100 per
page

    # Cache key
    cache_key_str = cache_key('products', search=search, category=category,
                              sort=sort, page=page, per_page=per_page)

    def fetch_products():
        db = next(get_db())

        query = db.query(Product).filter(Product.is_active == True)

        # Search filter
        if search:
            query = query.filter(
                or_(
                    Product.name.ilike(f'%{search}%'),
                    Product.description.ilike(f'%{search}%')
                )
            )

        # Category filter
        if category:
            query = query.filter(Product.category_id == category)

        # Sorting
        if sort == 'price_asc':

```



```

        query = query.order_by(asc(Product.price))
    elif sort == 'price_desc':
        query = query.order_by(desc(Product.price))
    elif sort == 'rating':
        query = query.order_by(desc(Product.rating))
    else: # name
        query = query.order_by(asc(Product.name))

    # Pagination
    total = query.count()
    products = query.offset((page - 1) * per_page).limit(per_page).all()

    return {
        'products': [
            {
                'id': p.id,
                'name': p.name,
                'description': p.description,
                'price': p.price,
                'stock': p.stock,
                'category_id': p.category_id,
                'image_url': p.image_url,
                'rating': p.rating,
                'review_count': p.review_count
            } for p in products
        ],
        'pagination': {
            'page': page,
            'per_page': per_page,
            'total': total,
            'pages': (total + per_page - 1) // per_page
        }
    }
}

```

```

    return jsonify(get_cached_or_fetch(cache_key_str, fetch_products,
ttl=300))

```

```

@app.route('/products/<int:product_id>')

```

```

def get_product(product_id):

```

```

    cache_key_str = cache_key('product', id=product_id)

```

```

def fetch_product():

```

```

    db = next(get_db())

```

```

    product = db.query(Product).filter(

```

```

        Product.id == product_id,

```

```

        Product.is_active == True

```

```

    ).first()

```

```

    if not product:
        return None

    return {
        'id': product.id,
        'name': product.name,
        'description': product.description,
        'price': product.price,
        'stock': product.stock,
        'category_id': product.category_id,
        'image_url': product.image_url,
        'rating': product.rating,
        'review_count': product.review_count,
        'category': {
            'id': product.category.id,
            'name': product.category.name
        } if product.category else None
    }

product_data = get_cached_or_fetch(cache_key_str, fetch_product, ttl=600)

if not product_data:
    return jsonify({'error': 'Product not found'}), 404

return jsonify(product_data)

@app.route('/categories')
def get_categories():
    cache_key_str = cache_key('categories')

    def fetch_categories():
        db = next(get_db())
        categories = db.query(Category).all()

        return {
            'categories': [
                {
                    'id': c.id,
                    'name': c.name,
                    'description': c.description
                } for c in categories
            ]
        }

    return jsonify(get_cached_or_fetch(cache_key_str, fetch_categories,
ttl=3600))

@app.route('/products/<int:product_id>/stock', methods=['PUT'])

```

```

def update_stock(product_id):
    """Update product stock (internal endpoint)"""
    data = request.json
    new_stock = data.get('stock')

    if new_stock is None or new_stock < 0:
        return jsonify({'error': 'Invalid stock value'}), 400

    db = next(get_db())
    product = db.query(Product).filter(Product.id == product_id).first()

    if not product:
        return jsonify({'error': 'Product not found'}), 404

    old_stock = product.stock
    product.stock = new_stock
    db.commit()

    # Invalidate cache
    redis_client.delete(cache_key('product', id=product_id))

    logger.info(f"Stock updated for product {product_id}: {old_stock} -> {new_stock}")

    return jsonify({
        'product_id': product_id,
        'old_stock': old_stock,
        'new_stock': new_stock
    })

if __name__ == '__main__':
    port = int(os.environ.get('PORT', 5001))
    app.run(host='0.0.0.0', port=port, debug=False)

```

To kończy pierwszą część praktycznego projektu end-to-end. W następnej sekcji omówimy infrastrukturę Terraform i deployment na EKS.

## Infrastruktura Terraform

### Główna konfiguracja Terraform

**infrastructure/terraform/main.tf:**

Plain Text

```
terraform {
  required_version = ">= 1.0"

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
    kubernetes = {
      source  = "hashicorp/kubernetes"
      version = "~> 2.23"
    }
    helm = {
      source  = "hashicorp/helm"
      version = "~> 2.11"
    }
  }
}

backend "s3" {
  # Konfiguracja będzie w plikach backend.tf w environments/
}

# Lokalne wartości
locals {
  project_name = var.project_name
  environment   = var.environment

  common_tags = {
    Project      = local.project_name
    Environment  = local.environment
    ManagedBy    = "Terraform"
    Owner        = var.owner
    CostCenter   = var.cost_center
  }
}

# Konfiguracje specyficzne dla środowiska
environment_config = {
  dev = {
    eks_node_instance_type = "t3.medium"
    eks_node_min_size      = 2
    eks_node_max_size      = 5
    eks_node_desired_size  = 2
    rds_instance_class     = "db.t3.micro"
    rds_allocated_storage  = 20
    elasticache_node_type  = "cache.t3.micro"
    elasticsearch_instance_type = "t3.small.elasticsearch"
  }
}
```

```

    enable_deletion_protection = false
}
staging = {
    eks_node_instance_type = "t3.large"
    eks_node_min_size      = 3
    eks_node_max_size      = 8
    eks_node_desired_size  = 3
    rds_instance_class     = "db.t3.small"
    rds_allocated_storage  = 50
    elasticache_node_type  = "cache.t3.small"
    elasticsearch_instance_type = "t3.medium.elasticsearch"
    enable_deletion_protection = false
}
prod = {
    eks_node_instance_type = "t3.xlarge"
    eks_node_min_size      = 5
    eks_node_max_size      = 20
    eks_node_desired_size  = 5
    rds_instance_class     = "db.t3.large"
    rds_allocated_storage  = 100
    elasticache_node_type  = "cache.t3.medium"
    elasticsearch_instance_type = "t3.large.elasticsearch"
    enable_deletion_protection = true
}
}

current_config = local.environment_config[local.environment]
}

# Data sources
data "aws_availability_zones" "available" {
    state = "available"
}

data "aws_caller_identity" "current" {}

# VPC Module
module "vpc" {
    source = "../modules/vpc"

    project_name = local.project_name
    environment  = local.environment

    vpc_cidr = var.vpc_cidr
    azs      = slice(data.aws_availability_zones.available.names, 0, 3)

    private_subnets = [for i in range(3) : cidrsubnet(var.vpc_cidr, 8, i)]
    public_subnets  = [for i in range(3) : cidrsubnet(var.vpc_cidr, 8, i +

```

```

100)]
    database_subnets = [for i in range(3) : cidrsubnet(var.vpc_cidr, 8, i +
200)]

    enable_nat_gateway = true
    enable_vpn_gateway = false
    enable_dns_hostnames = true
    enable_dns_support = true

    tags = local.common_tags
}

# EKS Module
module "eks" {
    source = "../modules/eks"

    project_name = local.project_name
    environment   = local.environment

    vpc_id          = module.vpc.vpc_id
    subnet_ids      = module.vpc.private_subnets

    cluster_version = var.eks_cluster_version

    node_groups = {
        main = {
            instance_types = [local.current_config.eks_node_instance_type]
            min_size       = local.current_config.eks_node_min_size
            max_size       = local.current_config.eks_node_max_size
            desired_size    = local.current_config.eks_node_desired_size

            k8s_labels = {
                Environment = local.environment
                NodeGroup   = "main"
            }

            taints = []
        }

        monitoring = {
            instance_types = ["t3.large"]
            min_size       = 1
            max_size       = 3
            desired_size    = 1

            k8s_labels = {
                Environment = local.environment
                NodeGroup   = "monitoring"
            }
        }
    }
}

```

```

        Purpose      = "monitoring"
    }

    taints = [
        {
            key      = "monitoring"
            value     = "true"
            effect    = "NO_SCHEDULE"
        }
    ]
}

tags = local.common_tags
}

# RDS Module
module "rds" {
    source = "../modules/rds"

    project_name = local.project_name
    environment  = local.environment

    vpc_id          = module.vpc.vpc_id
    subnet_ids      = module.vpc.database_subnets
    allowed_cidr_blocks = module.vpc.private_subnets_cidr_blocks

    engine          = "postgres"
    engine_version  = "15.4"
    instance_class  = local.current_config.rds_instance_class

    allocated_storage      = local.current_config.rds_allocated_storage
    max_allocated_storage  = local.current_config.rds_allocated_storage * 2

    db_name = replace(local.project_name, "-", "")
    username = var.db_username
    password = var.db_password

    backup_retention_period = local.environment == "prod" ? 30 : 7
    backup_window           = "03:00-04:00"
    maintenance_window      = "sun:04:00-sun:05:00"

    deletion_protection = local.current_config.enable_deletion_protection

    monitoring_interval = 60
    performance_insights_enabled = true

    tags = local.common_tags

```

```

}

# ElastiCache Module
module "elasticsearch" {
    source = "../modules/elasticsearch"

    project_name = local.project_name
    environment  = local.environment

    vpc_id          = module.vpc.vpc_id
    subnet_ids      = module.vpc.private_subnets
    allowed_cidr_blocks = module.vpc.private_subnets_cidr_blocks

    engine          = "redis"
    node_type       = local.current_config.elasticache_node_type
    num_cache_clusters = local.environment == "prod" ? 3 : 1

    parameter_group_name = "default.redis7"
    port                  = 6379

    at_rest_encryption_enabled = true
    transit_encryption_enabled = true

    automatic_failover_enabled = local.environment == "prod"
    multi_az_enabled           = local.environment == "prod"

    snapshot_retention_limit = local.environment == "prod" ? 7 : 1
    snapshot_window          = "03:00-05:00"
    maintenance_window       = "sun:05:00-sun:07:00"

    tags = local.common_tags
}

# Elasticsearch Module
module "elasticsearch" {
    source = "../modules/elasticsearch"

    project_name = local.project_name
    environment  = local.environment

    vpc_id          = module.vpc.vpc_id
    subnet_ids      = module.vpc.private_subnets
    allowed_cidr_blocks = module.vpc.private_subnets_cidr_blocks

    elasticsearch_version = "7.10"
    instance_type         = local.current_config.elasticsearch_instance_type
    instance_count        = local.environment == "prod" ? 3 : 1

```



```

    dedicated_master_enabled = local.environment == "prod"
    master_instance_type     = local.environment == "prod" ?
"t3.medium.elasticsearch" : null
    master_instance_count     = local.environment == "prod" ? 3 : 0

    ebs_enabled = true
    volume_type = "gp3"
    volume_size = local.environment == "prod" ? 100 : 20

    encrypt_at_rest = true
    node_to_node_encryption = true

    tags = local.common_tags
}

# Monitoring Module
module "monitoring" {
    source = "../modules/monitoring"

    project_name = local.project_name
    environment  = local.environment

    vpc_id = module.vpc.vpc_id

    # EKS cluster info
    eks_cluster_name = module.eks.cluster_name
    eks_cluster_endpoint = module.eks.cluster_endpoint
    eks_cluster_ca_certificate = module.eks.cluster_certificate_authority_data

    # RDS info
    rds_instance_id = module.rds.db_instance_id

    # ElastiCache info
    elasticache_cluster_id = module.elasticache.cluster_id

    # Elasticsearch info
    elasticsearch_domain_name = module.elasticsearch.domain_name

    # SNS topic for alerts
    alert_email = var.alert_email

    tags = local.common_tags
}

# S3 Buckets
resource "aws_s3_bucket" "app_assets" {
    bucket = "${local.project_name}-${local.environment}-
assets-${random_id.bucket_suffix.hex}"

```

```

tags = merge(local.common_tags, {
  Name = "Application Assets"
  Type = "Assets"
})
}

resource "aws_s3_bucket" "app_backups" {
  bucket = "${local.project_name}-${local.environment}-
backups-${random_id.bucket_suffix.hex}"

  tags = merge(local.common_tags, {
    Name = "Application Backups"
    Type = "Backups"
  })
}

resource "random_id" "bucket_suffix" {
  byte_length = 4
}

# S3 bucket configurations
resource "aws_s3_bucket_versioning" "app_assets" {
  bucket = aws_s3_bucket.app_assets.id
  versioning_configuration {
    status = local.environment == "prod" ? "Enabled" : "Suspended"
  }
}

resource "aws_s3_bucket_server_side_encryption_configuration" "app_assets" {
  bucket = aws_s3_bucket.app_assets.id

  rule {
    apply_server_side_encryption_by_default {
      sse_algorithm = "AES256"
    }
  }
}

resource "aws_s3_bucket_public_access_block" "app_assets" {
  bucket = aws_s3_bucket.app_assets.id

  block_public_acls       = true
  block_public_policy     = true
  ignore_public_acls     = true
  restrict_public_buckets = true
}

```

```

# CloudFront Distribution
resource "aws_cloudfront_distribution" "app" {
  count = local.environment == "prod" ? 1 : 0

  origin {
    domain_name = aws_s3_bucket.app_assets.bucket_regional_domain_name
    origin_id    = "S3-#{aws_s3_bucket.app_assets.bucket}"

    s3_origin_config {
      origin_access_identity =
aws_cloudfront_origin_access_identity.app[0].cloudfront_access_identity_path
    }
  }

  enabled          = true
  is_ipv6_enabled  = true
  default_root_object = "index.html"

  default_cache_behavior {
    allowed_methods      = ["DELETE", "GET", "HEAD", "OPTIONS", "PATCH",
"POST", "PUT"]
    cached_methods      = ["GET", "HEAD"]
    target_origin_id     = "S3-#{aws_s3_bucket.app_assets.bucket}"
    compress             = true
    viewer_protocol_policy = "redirect-to-https"

    forwarded_values {
      query_string = false
      cookies {
        forward = "none"
      }
    }
  }

  min_ttl      = 0
  default_ttl  = 3600
  max_ttl      = 86400
}

price_class = "PriceClass_100"

restrictions {
  geo_restriction {
    restriction_type = "none"
  }
}

viewer_certificate {
  cloudfront_default_certificate = true
}

```

```

    }

    tags = local.common_tags
}

resource "aws_cloudfront_origin_access_identity" "app" {
    count = local.environment == "prod" ? 1 : 0
    comment = "OAI for ${local.project_name} ${local.environment}"
}

# ECR Repositories
resource "aws_ecr_repository" "app_repos" {
    for_each = toset([
        "frontend",
        "api-gateway",
        "product-service",
        "user-service",
        "payment-service"
    ])

    name = "${local.project_name}/${each.key}"
    image_tag_mutability = "MUTABLE"

    image_scanning_configuration {
        scan_on_push = true
    }

    encryption_configuration {
        encryption_type = "AES256"
    }

    tags = local.common_tags
}

# ECR Lifecycle Policies
resource "aws_ecr_lifecycle_policy" "app_repos" {
    for_each = aws_ecr_repository.app_repos

    repository = each.value.name

    policy = jsonencode({
        rules = [
            {
                rulePriority = 1
                description = "Keep last 10 production images"
                selection = {
                    tagStatus = "tagged"
                    tagPrefixList = ["prod"]
                }
            }
        ]
    })
}

```

```

        countType      = "imageCountMoreThan"
        countNumber    = 10
    }
    action = {
        type = "expire"
    }
},
{
    rulePriority = 2
    description  = "Keep last 5 staging images"
    selection = {
        tagStatus      = "tagged"
        tagPrefixList  = ["staging"]
        countType      = "imageCountMoreThan"
        countNumber    = 5
    }
    action = {
        type = "expire"
    }
},
{
    rulePriority = 3
    description  = "Delete untagged images older than 1 day"
    selection = {
        tagStatus      = "untagged"
        countType      = "sinceImagePushed"
        countUnit      = "days"
        countNumber    = 1
    }
    action = {
        type = "expire"
    }
}
]
}))
}

```

## Moduł EKS

**infrastructure/terraform/modules/eks/main.tf:**

Plain Text

```

# EKS Cluster
resource "aws_eks_cluster" "main" {
    name      = "${var.project_name}-${var.environment}"

```

```

role_arn = aws_iam_role.cluster.arn
version  = var.cluster_version

vpc_config {
  subnet_ids           = var.subnet_ids
  endpoint_private_access = true
  endpoint_public_access = true
  public_access_cidrs   = var.public_access_cidrs

  security_group_ids = [aws_security_group.cluster.id]
}

encryption_config {
  provider {
    key_arn = aws_kms_key.eks.arn
  }
  resources = ["secrets"]
}

enabled_cluster_log_types = ["api", "audit", "authenticator",
"controllerManager", "scheduler"]

depends_on = [
  aws_iam_role_policy_attachment.cluster_AmazonEKSClusterPolicy,
  aws_cloudwatch_log_group.cluster,
]

tags = var.tags
}

# CloudWatch Log Group
resource "aws_cloudwatch_log_group" "cluster" {
  name =
"/aws/eks/${var.project_name}-${var.environment}/cluster"
  retention_in_days = 30

  tags = var.tags
}

# KMS Key for EKS encryption
resource "aws_kms_key" "eks" {
  description = "EKS Secret Encryption Key for
${var.project_name}-${var.environment}"
  deletion_window_in_days = 7
  enable_key_rotation     = true

  tags = var.tags
}

```

```

resource "aws_kms_alias" "eks" {
  name           = "alias/${var.project_name}-${var.environment}-eks"
  target_key_id = aws_kms_key.eks.key_id
}

# EKS Cluster IAM Role
resource "aws_iam_role" "cluster" {
  name = "${var.project_name}-${var.environment}-eks-cluster-role"

  assume_role_policy = jsonencode({
    Statement = [{
      Action = "sts:AssumeRole"
      Effect = "Allow"
      Principal = {
        Service = "eks.amazonaws.com"
      }
    }]
    Version = "2012-10-17"
  })

  tags = var.tags
}

resource "aws_iam_role_policy_attachment" "cluster_AmazonEKSClusterPolicy" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"
  role       = aws_iam_role.cluster.name
}

# EKS Cluster Security Group
resource "aws_security_group" "cluster" {
  name_prefix = "${var.project_name}-${var.environment}-eks-cluster-"
  vpc_id      = var.vpc_id

  ingress {
    description = "HTTPS"
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["10.0.0.0/8"]
  }

  egress {
    description = "All outbound traffic"
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

```

```

}

tags = merge(var.tags, {
  Name = "${var.project_name}-${var.environment}-eks-cluster-sg"
})
}

# EKS Node Groups
resource "aws_eks_node_group" "main" {
  for_each = var.node_groups

  cluster_name      = aws_eks_cluster.main.name
  node_group_name   = "${var.project_name}-${var.environment}-${each.key}"
  node_role_arn     = aws_iam_role.node_group.arn
  subnet_ids       = var.subnet_ids

  instance_types    = each.value.instance_types
  ami_type          = "AL2_x86_64"
  capacity_type     = "ON_DEMAND"
  disk_size         = 50

  scaling_config {
    desired_size = each.value.desired_size
    max_size     = each.value.max_size
    min_size     = each.value.min_size
  }

  update_config {
    max_unavailable_percentage = 25
  }

  labels = each.value.k8s_labels

  dynamic "taint" {
    for_each = each.value.taints
    content {
      key    = taint.value.key
      value  = taint.value.value
      effect = taint.value.effect
    }
  }
}

# Ensure that IAM Role permissions are created before and deleted after EKS
Node Group handling.
depends_on = [
  aws_iam_role_policy_attachment.node_group_AmazonEKSEKSWorkerNodePolicy,
  aws_iam_role_policy_attachment.node_group_AmazonEKS_CNI_Policy,

```



```

aws_iam_role_policy_attachment.node_group_AmazonEC2ContainerRegistryReadOnly,
]

tags = merge(var.tags, {
  Name = "${var.project_name}-${var.environment}-${each.key}-node-group"
})
}

# EKS Node Group IAM Role
resource "aws_iam_role" "node_group" {
  name = "${var.project_name}-${var.environment}-eks-node-group-role"

  assume_role_policy = jsonencode({
    Statement = [{
      Action = "sts:AssumeRole"
      Effect = "Allow"
      Principal = {
        Service = "ec2.amazonaws.com"
      }
    }]
    Version = "2012-10-17"
  })

  tags = var.tags
}

resource "aws_iam_role_policy_attachment"
"node_group_AmazonEKSWorkerNodePolicy" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy"
  role       = aws_iam_role.node_group.name
}

resource "aws_iam_role_policy_attachment" "node_group_AmazonEKS_CNI_Policy" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy"
  role       = aws_iam_role.node_group.name
}

resource "aws_iam_role_policy_attachment"
"node_group_AmazonEC2ContainerRegistryReadOnly" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly"
  role       = aws_iam_role.node_group.name
}

# Additional policies for node groups
resource "aws_iam_role_policy" "node_group_additional" {
  name = "${var.project_name}-${var.environment}-eks-node-group-additional"
  role = aws_iam_role.node_group.id
}

```

```

policy = jsonencode({
  Version = "2012-10-17"
  Statement = [
    {
      Effect = "Allow"
      Action = [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject",
        "s3:ListBucket"
      ]
      Resource = [
        "arn:aws:s3:::${var.project_name}-${var.environment}-*",
        "arn:aws:s3:::${var.project_name}-${var.environment}-*/*"
      ]
    },
    {
      Effect = "Allow"
      Action = [
        "secretsmanager:GetSecretValue",
        "secretsmanager:DescribeSecret"
      ]
      Resource =
"arn:aws:secretsmanager:*:*:secret:${var.project_name}/${var.environment}/*"
    },
    {
      Effect = "Allow"
      Action = [
        "ssm:GetParameter",
        "ssm:GetParameters",
        "ssm:GetParametersByPath"
      ]
      Resource =
"arn:aws:ssm:*:*:parameter/${var.project_name}/${var.environment}/*"
    }
  ]
})
}

```

# EKS Add-ons

```

resource "aws_eks_addon" "vpc_cni" {
  cluster_name      = aws_eks_cluster.main.name
  addon_name        = "vpc-cni"
  addon_version     = var.vpc_cni_version
  resolve_conflicts = "OVERWRITE"
  service_account_role_arn = aws_iam_role.vpc_cni.arn
}

```

```
tags = var.tags
```

```

}

resource "aws_eks_addon" "coredns" {
  cluster_name      = aws_eks_cluster.main.name
  addon_name        = "coredns"
  addon_version      = var.coredns_version
  resolve_conflicts = "OVERWRITE"

  depends_on = [aws_eks_node_group.main]

  tags = var.tags
}

resource "aws_eks_addon" "kube_proxy" {
  cluster_name      = aws_eks_cluster.main.name
  addon_name        = "kube-proxy"
  addon_version      = var.kube_proxy_version
  resolve_conflicts = "OVERWRITE"

  tags = var.tags
}

resource "aws_eks_addon" "ebs_csi_driver" {
  cluster_name      = aws_eks_cluster.main.name
  addon_name        = "aws-ebs-csi-driver"
  addon_version      = var.ebs_csi_driver_version
  resolve_conflicts = "OVERWRITE"
  service_account_role_arn = aws_iam_role.ebs_csi_driver.arn

  tags = var.tags
}

# IAM Role for VPC CNI
resource "aws_iam_role" "vpc_cni" {
  name = "${var.project_name}-${var.environment}-vpc-cni-role"

  assume_role_policy = jsonencode({
    Statement = [{
      Action = "sts:AssumeRoleWithWebIdentity"
      Effect = "Allow"
      Principal = {
        Federated = aws_iam_openid_connect_provider.cluster.arn
      }
    }]
    Condition = {
      StringEquals = {
        "${replace(aws_iam_openid_connect_provider.cluster.url, "https://",
        "")}:sub": "system:serviceaccount:kube-system:aws-node"
        "${replace(aws_iam_openid_connect_provider.cluster.url, "https://",

```

```

    ""}}:aud": "sts.amazonaws.com"
    }
  }
}]
Version = "2012-10-17"
}))

tags = var.tags
}

resource "aws_iam_role_policy_attachment" "vpc_cni" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy"
  role       = aws_iam_role.vpc_cni.name
}

# IAM Role for EBS CSI Driver
resource "aws_iam_role" "ebs_csi_driver" {
  name = "${var.project_name}-${var.environment}-ebs-csi-driver-role"

  assume_role_policy = jsonencode({
    Statement = [{
      Action = "sts:AssumeRoleWithWebIdentity"
      Effect = "Allow"
      Principal = {
        Federated = aws_iam_openid_connect_provider.cluster.arn
      }
      Condition = {
        StringEquals = {
          "${replace(aws_iam_openid_connect_provider.cluster.url, "https://",
            ""}}:sub": "system:serviceaccount:kube-system:ebs-csi-controller-sa"
          "${replace(aws_iam_openid_connect_provider.cluster.url, "https://",
            ""}}:aud": "sts.amazonaws.com"
        }
      }
    }]
    Version = "2012-10-17"
  })

  tags = var.tags
}

resource "aws_iam_role_policy_attachment" "ebs_csi_driver" {
  policy_arn = "arn:aws:iam::aws:policy/service-
role/Amazon_EBS_CSI_DriverPolicy"
  role       = aws_iam_role.ebs_csi_driver.name
}

# OIDC Provider

```

```

data "tls_certificate" "cluster" {
  url = aws_eks_cluster.main.identity[0].oidc[0].issuer
}

resource "aws_iam_openid_connect_provider" "cluster" {
  client_id_list  = ["sts.amazonaws.com"]
  thumbprint_list =
[data.tls_certificate.cluster.certificates[0].sha1_fingerprint]
  url            = aws_eks_cluster.main.identity[0].oidc[0].issuer

  tags = var.tags
}

```

## CI/CD Pipeline

### Jenkins Pipeline

**ci-cd/jenkins/Jenkinsfile:**

Plain Text

```

pipeline {
  agent any

  environment {
    AWS_DEFAULT_REGION = 'eu-west-1'
    ECR_REGISTRY = '123456789012.dkr.ecr.eu-west-1.amazonaws.com'
    PROJECT_NAME = 'ecommerce'
    KUBECONFIG = credentials('kubecfg')
    DOCKER_BUILDKIT = '1'
  }

  parameters {
    choice(
      name: 'ENVIRONMENT',
      choices: ['dev', 'staging', 'prod'],
      description: 'Target environment for deployment'
    )
    choice(
      name: 'DEPLOYMENT_TYPE',
      choices: ['rolling', 'blue-green', 'canary'],
      description: 'Deployment strategy'
    )
    booleanParam(

```

```

        name: 'RUN_TESTS',
        defaultValue: true,
        description: 'Run automated tests'
    )
    booleanParam(
        name: 'DEPLOY_INFRASTRUCTURE',
        defaultValue: false,
        description: 'Deploy/update infrastructure with Terraform'
    )
}

stages {
    stage('Checkout') {
        steps {
            checkout scm
            script {
                env.GIT_COMMIT_SHORT = sh(
                    script: 'git rev-parse --short HEAD',
                    returnStdout: true
                ).trim()
                env.BUILD_TAG =
"${env.ENVIRONMENT}-${env.GIT_COMMIT_SHORT}-${env.BUILD_NUMBER}"
            }
        }
    }

    stage('Setup') {
        parallel {
            stage('Install Dependencies') {
                steps {
                    sh '''
                        # Install Node.js dependencies for frontend
                        cd applications/frontend
                        npm ci

                        # Install Python dependencies for services
                        for service in api-gateway product-service user-
service payment-service; do
                            cd ../applications/$service
                            pip install -r requirements.txt
                        done
                    '''
                }
            }

            stage('Configure AWS') {
                steps {
                    withCredentials([

```

```

        [$class: 'AmazonWebServicesCredentialsBinding',
        credentialsId: 'aws-credentials']
    }) {
        sh '''
            # Configure AWS CLI
            aws configure set region $AWS_DEFAULT_REGION

            # Login to ECR
            aws ecr get-login-password --region
$AWS_DEFAULT_REGION | \
            docker login --username AWS --password-
stdin $ECR_REGISTRY
        '''
    }
}

stage('Code Quality & Security') {
    when {
        expression { params.RUN_TESTS }
    }
    parallel {
        stage('Frontend Tests') {
            steps {
                dir('applications/frontend') {
                    sh '''
                        # Lint
                        npm run lint

                        # Unit tests with coverage
                        npm run test:coverage

                        # Build test
                        npm run build
                    '''

                    publishHTML([
                        allowMissing: false,
                        alwaysLinkToLastBuild: true,
                        keepAll: true,
                        reportDir: 'coverage/lcov-report',
                        reportFiles: 'index.html',
                        reportName: 'Frontend Coverage Report'
                    ])
                }
            }
        }
    }
}

```

```

    }

    stage('Backend Tests') {
        steps {
            script {
                def services = ['api-gateway', 'product-service',
'user-service', 'payment-service']

                services.each { service ->
                    dir("applications/${service}") {
                        sh '''
                            # Code quality checks
                            flake8 app/ --max-line-length=88
                            black --check app/

                            # Security scan
                            bandit -r app/ -f json -o bandit-
report.json || true

                            # Unit tests
                            python -m pytest tests/ -v --cov=app
--cov-report=xml --cov-report=html
                        '''
                    }
                }
            }
            post {
                always {
                    publishCoverage adapters: [
coberturaAdapter('applications/*/coverage.xml')
                    ], sourceFileResolver:
sourceFiles('STORE_LAST_BUILD')
                }
            }
        }

        stage('Infrastructure Validation') {
            when {
                expression { params.DEPLOY_INFRASTRUCTURE }
            }
            steps {
                dir('infrastructure/terraform') {
                    sh '''
                        # Terraform validation
                        terraform init -backend=false
                        terraform validate
                    '''
                }
            }
        }
    }
}

```



```

        terraform fmt -check

        # Security scan with tfsec
        tfsec . --format json --out tfsec-report.json
    || true

    ...
}
}
}
}
}

stage('Build Images') {
    parallel {
        stage('Frontend Image') {
            steps {
                dir('applications/frontend') {
                    script {
                        def image =
docker.build("${ECR_REGISTRY}/${PROJECT_NAME}/frontend:${BUILD_TAG}")
                        image.push()
                        image.push("${env.ENVIRONMENT}-latest")
                    }
                }
            }
        }

        stage('Backend Images') {
            steps {
                script {
                    def services = ['api-gateway', 'product-service',
'user-service', 'payment-service']

                    services.each { service ->
                        dir("applications/${service}") {
                            def image =
docker.build("${ECR_REGISTRY}/${PROJECT_NAME}/${service}:${BUILD_TAG}")
                            image.push()
                            image.push("${env.ENVIRONMENT}-latest")
                        }
                    }
                }
            }
        }
    }
}

stage('Deploy Infrastructure') {

```

```

when {
    expression { params.DEPLOY_INFRASTRUCTURE }
}
steps {
    dir('infrastructure/terraform') {
        withCredentials([
            [$class: 'AmazonWebServicesCredentialsBinding',
             credentialsId: 'aws-credentials']
        ]) {
            sh '''
                # Copy environment-specific configuration
                cp environments/${ENVIRONMENT}/terraform.tfvars .
                cp environments/${ENVIRONMENT}/backend.tf .

                # Initialize Terraform
                terraform init

                # Plan
                terraform plan -out=tfplan

                # Apply (with approval for production)
                if [ "$ENVIRONMENT" = "prod" ]; then
                    echo "Production deployment requires manual
approval"
                    input message: 'Deploy to production?', ok:
'Deploy'

                    fi

                    terraform apply tfplan
                '''
            }
        }
    }
}

stage('Deploy Application') {
    steps {
        script {
            def deploymentStrategy = params.DEPLOYMENT_TYPE

            withCredentials([
                [$class: 'AmazonWebServicesCredentialsBinding',
                 credentialsId: 'aws-credentials']
            ]) {
                sh '''
                    # Update kubeconfig
                    aws eks update-kubeconfig --region
$AWS_DEFAULT_REGION --name ${PROJECT_NAME}-${ENVIRONMENT}

```

```

        # Update image tags in Kubernetes manifests
        cd
infrastructure/kubernetes/overlays/${ENVIRONMENT}

        # Use kustomize to update image tags
        kustomize edit set image
frontend=${ECR_REGISTRY}/${PROJECT_NAME}/frontend:${BUILD_TAG}
        kustomize edit set image api-
gateway=${ECR_REGISTRY}/${PROJECT_NAME}/api-gateway:${BUILD_TAG}
        kustomize edit set image product-
service=${ECR_REGISTRY}/${PROJECT_NAME}/product-service:${BUILD_TAG}
        kustomize edit set image user-
service=${ECR_REGISTRY}/${PROJECT_NAME}/user-service:${BUILD_TAG}
        kustomize edit set image payment-
service=${ECR_REGISTRY}/${PROJECT_NAME}/payment-service:${BUILD_TAG}
        '''

        if (deploymentStrategy == 'rolling') {
            sh '''
                # Rolling deployment
                kubectl apply -k
infrastructure/kubernetes/overlays/${ENVIRONMENT}

                # Wait for rollout
                kubectl rollout status deployment/frontend -n
${ENVIRONMENT}

                kubectl rollout status deployment/api-gateway
-n ${ENVIRONMENT}

                kubectl rollout status deployment/product-
service -n ${ENVIRONMENT}

                kubectl rollout status deployment/user-
service -n ${ENVIRONMENT}

                kubectl rollout status deployment/payment-
service -n ${ENVIRONMENT}
            '''
        } else if (deploymentStrategy == 'blue-green') {
            sh '''
                # Blue-green deployment using Argo Rollouts
                kubectl apply -k
infrastructure/kubernetes/overlays/${ENVIRONMENT}/blue-green

                # Wait for analysis
                kubectl argo rollouts get rollout frontend -n
${ENVIRONMENT} --watch
            '''
        } else if (deploymentStrategy == 'canary') {
            sh '''

```

```

        # Canary deployment using Istio
        kubectl apply -k
infrastructure/kubernetes/overlays/${ENVIRONMENT}/canary

        # Gradual traffic shift
        ./scripts/canary-deployment.sh ${ENVIRONMENT}
    ''
    }
}

stage('Integration Tests') {
    when {
        expression { params.RUN_TESTS }
    }
    steps {
        sh '''
            # Wait for services to be ready
            sleep 60

            # Run integration tests
            cd tests/integration
            python -m pytest -v --environment=${ENVIRONMENT}
        ''

    }
    post {
        always {
            publishTestResults testResultsPattern:
'tests/integration/results.xml'
        }
    }
}

stage('Performance Tests') {
    when {
        allof {
            expression { params.RUN_TESTS }
            anyOf {
                environment name: 'ENVIRONMENT', value: 'staging'
                environment name: 'ENVIRONMENT', value: 'prod'
            }
        }
    }
    steps {
        sh '''
            # Run load tests with k6

```

```

        cd tests/performance
        k6 run --env ENVIRONMENT=${ENVIRONMENT} load-test.js
    '''
}
post {
    always {
        publishHTML([
            allowMissing: false,
            alwaysLinkToLastBuild: true,
            keepAll: true,
            reportDir: 'tests/performance/reports',
            reportFiles: 'index.html',
            reportName: 'Performance Test Report'
        ])
    }
}
}

stage('Security Scan') {
    when {
        anyOf {
            environment name: 'ENVIRONMENT', value: 'staging'
            environment name: 'ENVIRONMENT', value: 'prod'
        }
    }
    steps {
        sh '''
            # Container security scan with Trivy
            for service in frontend api-gateway product-service user-
service payment-service; do
                trivy image --format json --output ${service}-
security-report.json \

${ECR_REGISTRY}/${PROJECT_NAME}/${service}:${BUILD_TAG}
            done

            # Runtime security scan
            kubectl apply -f security/falco-rules.yaml
        '''
    }
}

stage('Monitoring Setup') {
    steps {
        sh '''
            # Deploy monitoring stack if not exists
            if ! kubectl get namespace monitoring; then
                kubectl create namespace monitoring
            fi
        '''
    }
}

```

```

        helm repo add prometheus-community
https://prometheus-community.github.io/helm-charts
        helm repo update
        helm install prometheus prometheus-community/kube-
prometheus-stack -n monitoring
    fi

    # Update Grafana dashboards
    kubectl apply -f monitoring/grafana/dashboards/ -n
monitoring

    # Update Prometheus rules
    kubectl apply -f monitoring/prometheus/rules/ -n
monitoring
    ...
}
}
}

post {
    always {
        // Clean up
        sh '''
            docker system prune -f
            kubectl delete pods --field-selector=status.phase=Succeeded -
n ${ENVIRONMENT} || true
        '''
    }

    success {
        script {
            if (env.ENVIRONMENT == 'prod') {
                // Send success notification
                slackSend(
                    channel: '#deployments',
                    color: 'good',
                    message: "✅ Production deployment successful!
Version: ${BUILD_TAG}"
                )
            }
        }
    }
}

failure {
    script {
        // Send failure notification
        slackSend(
            channel: '#deployments',

```

```

        color: 'danger',
        message: "❌ Deployment failed for ${env.ENVIRONMENT}!"
    Build: ${BUILD_TAG}"
    )

    // Rollback on production failure
    if (env.ENVIRONMENT == 'prod') {
        sh '''
            echo "Rolling back production deployment..."
            kubectl rollout undo deployment/frontend -n
${ENVIRONMENT}
            kubectl rollout undo deployment/api-gateway -n
${ENVIRONMENT}
            kubectl rollout undo deployment/product-service -n
${ENVIRONMENT}
            kubectl rollout undo deployment/user-service -n
${ENVIRONMENT}
            kubectl rollout undo deployment/payment-service -n
${ENVIRONMENT}
        '''
    }
}
}
}
}
}
}
}

```

## GitHub Actions Workflow

**ci-cd/github-actions/workflows/ci-cd.yml:**

YAML

```

name: CI/CD Pipeline

on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main]
  workflow_dispatch:
    inputs:
      environment:
        description: 'Target environment'
        required: true
        default: 'dev'
        type: choice

```

```
options:
  - dev
  - staging
  - prod
deployment_type:
  description: 'Deployment strategy'
  required: true
  default: 'rolling'
  type: choice
  options:
    - rolling
    - blue-green
    - canary
```

env:

```
AWS_REGION: eu-west-1
ECR_REGISTRY: 123456789012.dkr.ecr.eu-west-1.amazonaws.com
PROJECT_NAME: ecommerce
```

jobs:

```
setup:
  runs-on: ubuntu-latest
  outputs:
    environment: ${ steps.set-env.outputs.environment }
    build-tag: ${ steps.set-env.outputs.build-tag }
  steps:
    - name: Set environment
      id: set-env
      run: |
        if [ "${ github.event_name }" = "workflow_dispatch" ]; then
          echo "environment=${ github.event.inputs.environment }" >>
$GITHUB_OUTPUT
        elif [ "${ github.ref }" = "refs/heads/main" ]; then
          echo "environment=prod" >> $GITHUB_OUTPUT
        elif [ "${ github.ref }" = "refs/heads/develop" ]; then
          echo "environment=staging" >> $GITHUB_OUTPUT
        else
          echo "environment=dev" >> $GITHUB_OUTPUT
        fi
```

```
        BUILD_TAG="${ github.ref_name }-${ echo ${ github.sha } | cut -
c1-7}-${ github.run_number }"
        echo "build-tag=$BUILD_TAG" >> $GITHUB_OUTPUT
```

test:

```
runs-on: ubuntu-latest
needs: setup
strategy:
```



```

matrix:
  test-type: [frontend, backend, integration]

steps:
  - name: Checkout code
    uses: actions/checkout@v4

  - name: Setup Node.js
    if: matrix.test-type == 'frontend'
    uses: actions/setup-node@v4
    with:
      node-version: '18'
      cache: 'npm'
      cache-dependency-path: applications/frontend/package-lock.json

  - name: Setup Python
    if: matrix.test-type != 'frontend'
    uses: actions/setup-python@v4
    with:
      python-version: '3.11'
      cache: 'pip'

  - name: Frontend Tests
    if: matrix.test-type == 'frontend'
    run: |
      cd applications/frontend
      npm ci
      npm run lint
      npm run test:coverage
      npm run build

  - name: Backend Tests
    if: matrix.test-type == 'backend'
    run: |
      for service in api-gateway product-service user-service payment-
service; do
        cd applications/$service
        pip install -r requirements.txt
        flake8 app/ --max-line-length=88
        black --check app/
        bandit -r app/ -f json -o bandit-report.json || true
        python -m pytest tests/ -v --cov=app --cov-report=xml
        cd ../..
      done

  - name: Integration Tests
    if: matrix.test-type == 'integration'
    run: |

```

```
cd tests/integration
pip install -r requirements.txt
python -m pytest -v
```

- name: Upload Coverage  
if: matrix.test-type != 'integration'  
uses: codecov/codecov-action@v3  
with:  
 files: ./coverage.xml  
 flags: \${{ matrix.test-type }}

#### security:

runs-on: ubuntu-latest

needs: setup

##### steps:

- name: Checkout code  
uses: actions/checkout@v4
- name: Run Trivy vulnerability scanner  
uses: aquasecurity/trivy-action@master  
with:  
 scan-type: 'fs'  
 scan-ref: '.'  
 format: 'sarif'  
 output: 'trivy-results.sarif'
- name: Upload Trivy scan results  
uses: github/codeql-action/upload-sarif@v2  
with:  
 sarif\_file: 'trivy-results.sarif'

#### build:

runs-on: ubuntu-latest

needs: [setup, test]

##### strategy:

matrix:

service: [frontend, api-gateway, product-service, user-service, payment-service]

##### steps:

- name: Checkout code  
uses: actions/checkout@v4
- name: Configure AWS credentials  
uses: aws-actions/configure-aws-credentials@v4  
with:  
 aws-access-key-id: \${{ secrets.AWS\_ACCESS\_KEY\_ID }}  
 aws-secret-access-key: \${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}

```

    aws-region: ${ env.AWS_REGION }}

- name: Login to Amazon ECR
  id: login-ecr
  uses: aws-actions/amazon-ecr-login@v2

- name: Build and push Docker image
  run: |
    cd applications/${ matrix.service }}

    docker build -t $ECR_REGISTRY/$PROJECT_NAME/${ matrix.service
}}:${ needs.setup.outputs.build-tag }} .
    docker push $ECR_REGISTRY/$PROJECT_NAME/${ matrix.service }}:${
needs.setup.outputs.build-tag }}

    docker tag $ECR_REGISTRY/$PROJECT_NAME/${ matrix.service }}:${
needs.setup.outputs.build-tag }} \
        $ECR_REGISTRY/$PROJECT_NAME/${ matrix.service }}:${
needs.setup.outputs.environment }}-latest
    docker push $ECR_REGISTRY/$PROJECT_NAME/${ matrix.service }}:${
needs.setup.outputs.environment }}-latest

deploy:
  runs-on: ubuntu-latest
  needs: [setup, test, build]
  if: github.ref == 'refs/heads/main' || github.ref == 'refs/heads/develop'
  || github.event_name == 'workflow_dispatch'
  environment: ${ needs.setup.outputs.environment }}

steps:
- name: Checkout code
  uses: actions/checkout@v4

- name: Configure AWS credentials
  uses: aws-actions/configure-aws-credentials@v4
  with:
    aws-access-key-id: ${ secrets.AWS_ACCESS_KEY_ID }}
    aws-secret-access-key: ${ secrets.AWS_SECRET_ACCESS_KEY }}
    aws-region: ${ env.AWS_REGION }}

- name: Setup kubectl
  uses: azure/setup-kubectl@v3
  with:
    version: 'v1.28.0'

- name: Setup Kustomize
  run: |
    curl -s "https://raw.githubusercontent.com/kubernetes-
```

```

sigs/kustomize/master/hack/install_kustomize.sh" | bash
    sudo mv kustomize /usr/local/bin/

- name: Update kubeconfig
  run: |
    aws eks update-kubeconfig --region $AWS_REGION --name
$PROJECT_NAME-${{ needs.setup.outputs.environment }}

- name: Deploy to Kubernetes
  run: |
    cd infrastructure/kubernetes/overlays/${{
needs.setup.outputs.environment }}

    # Update image tags
    kustomize edit set image
frontend=$ECR_REGISTRY/$PROJECT_NAME/frontend:${{ needs.setup.outputs.build-
tag }}

    kustomize edit set image api-
gateway=$ECR_REGISTRY/$PROJECT_NAME/api-gateway:${{
needs.setup.outputs.build-tag }}

    kustomize edit set image product-
service=$ECR_REGISTRY/$PROJECT_NAME/product-service:${{
needs.setup.outputs.build-tag }}

    kustomize edit set image user-
service=$ECR_REGISTRY/$PROJECT_NAME/user-service:${{
needs.setup.outputs.build-tag }}

    kustomize edit set image payment-
service=$ECR_REGISTRY/$PROJECT_NAME/payment-service:${{
needs.setup.outputs.build-tag }}

    # Apply manifests
    kubectl apply -k .

    # Wait for rollout
    kubectl rollout status deployment/frontend -n ${{
needs.setup.outputs.environment }}
    kubectl rollout status deployment/api-gateway -n ${{
needs.setup.outputs.environment }}
    kubectl rollout status deployment/product-service -n ${{
needs.setup.outputs.environment }}
    kubectl rollout status deployment/user-service -n ${{
needs.setup.outputs.environment }}
    kubectl rollout status deployment/payment-service -n ${{
needs.setup.outputs.environment }}

- name: Run smoke tests
  run: |
    # Wait for services to be ready

```

```

    sleep 60

    # Get service URL
    SERVICE_URL=$(kubectl get ingress -n ${needs.setup.outputs.environment} -o
jsonpath='{.items[0].status.loadBalancer.ingress[0].hostname}')

    # Basic health checks
    curl -f http://$SERVICE_URL/health || exit 1
    curl -f http://$SERVICE_URL/api/products || exit 1

notify:
  runs-on: ubuntu-latest
  needs: [setup, deploy]
  if: always()
  steps:
    - name: Notify Slack
      uses: 8398a7/action-slack@v3
      with:
        status: ${job.status}
        channel: '#deployments'
        text: |
          Deployment to ${needs.setup.outputs.environment} ${
job.status }%!
          Build: ${needs.setup.outputs.build-tag}
          Commit: ${github.sha}
      env:
        SLACK_WEBHOOK_URL: ${secrets.SLACK_WEBHOOK_URL}

```

To kończy sekcję o infrastrukturze i CI/CD. W następnej części omówimy Kubernetes manifests i monitoring.

## 22. Najlepsze praktyki i troubleshooting

### Najlepsze praktyki DevOps

#### Kultura i organizacja

Implementacja DevOps to przede wszystkim zmiana kulturowa w organizacji. Sukces DevOps zależy od współpracy między zespołami development i operations, wspólnej odpowiedzialności za jakość i wydajność aplikacji, oraz ciągłego doskonalenia procesów. Kluczowe elementy kultury DevOps obejmują automatyzację powtarzalnych zadań,

monitoring i observability na każdym poziomie, szybkie feedback loops, oraz learning from failures.

Organizacje powinny inwestować w cross-functional teams, które łączą umiejętności development, testing, security i operations. Ważne jest również ustanowienie clear ownership i accountability dla każdego komponentu systemu, regular retrospectives i post-mortems, oraz knowledge sharing sessions. Metryki i KPI powinny być aligned z business objectives i obejmować zarówno technical metrics (deployment frequency, lead time, MTTR) jak i business metrics (customer satisfaction, revenue impact).

## Security best practices

Security musi być integrated throughout the entire DevOps pipeline, nie może być afterthought. Implementacja DevSecOps wymaga shift-left approach, gdzie security considerations są wprowadzane na najwcześniejszych etapach development lifecycle. Kluczowe praktyki obejmują automated security scanning w CI/CD pipeline, regular vulnerability assessments, secure coding practices, oraz proper secrets management.

Container security wymaga szczególnej uwagi w środowiskach Kubernetes. Należy używać minimal base images, regular image scanning, proper RBAC configuration, network policies, oraz runtime security monitoring. Secrets powinny być stored w dedicated systems jak AWS Secrets Manager lub HashiCorp Vault, nigdy w code repositories czy environment variables.

Infrastructure security obejmuje proper VPC configuration z private subnets dla aplikacji, security groups z principle of least privilege, encryption at rest i in transit, oraz regular security audits. Compliance requirements (GDPR, SOC2, PCI-DSS) powinny być addressed through automated controls i continuous monitoring.

## Monitoring i observability

Comprehensive observability strategy obejmuje metrics, logs, traces i alerts. Metrics powinny cover infrastructure (CPU, memory, disk, network), application performance (response time, throughput, error rate), oraz business metrics (user engagement, conversion rate). Prometheus z Grafana stanowi excellent foundation dla metrics collection i visualization.

Logging strategy powinna obejmować structured logging z consistent format, centralized log aggregation (ELK stack, CloudWatch), proper log retention policies, oraz security considerations (no sensitive data in logs). Distributed tracing z tools jak Jaeger czy AWS X-Ray jest essential dla debugging complex microservices architectures.

Alerting powinno być based on SLIs (Service Level Indicators) i SLOs (Service Level Objectives), z proper escalation procedures i runbooks. Alert fatigue można uniknąć przez proper threshold tuning, alert grouping, oraz regular review of alert effectiveness.

## Performance optimization

Performance optimization w DevOps environment wymaga systematic approach. Database optimization obejmuje proper indexing, query optimization, connection pooling, oraz caching strategies. Application-level optimization includes code profiling, memory management, oraz asynchronous processing gdzie appropriate.

Infrastructure optimization obejmuje right-sizing instances, auto-scaling configuration, load balancing strategies, oraz CDN usage dla static content. Container optimization includes multi-stage builds, proper resource limits, oraz image optimization techniques.

Network optimization jest particularly important w cloud environments. Proper subnet design, availability zone distribution, oraz content delivery strategies mogą significantly impact performance. Monitoring network latency i throughput helps identify bottlenecks.

## Troubleshooting guide

### Kubernetes troubleshooting

#### Pod issues:

Bash

```
# Sprawdź status podów
kubectl get pods -n <namespace>

# Szczegółowe informacje o podzie
kubectl describe pod <pod-name> -n <namespace>

# Logi z poda
```

```
kubectl logs <pod-name> -n <namespace> --previous

# Wejście do poda dla debugowania
kubectl exec -it <pod-name> -n <namespace> -- /bin/bash

# Sprawdź events w namespace
kubectl get events -n <namespace> --sort-by='.lastTimestamp'

# Sprawdź resource usage
kubectl top pods -n <namespace>
```

## Service discovery issues:

Bash

```
# Sprawdź services
kubectl get svc -n <namespace>

# Sprawdź endpoints
kubectl get endpoints -n <namespace>

# Test connectivity z poda
kubectl exec -it <pod-name> -n <namespace> -- nslookup <service-name>

# Sprawdź network policies
kubectl get networkpolicy -n <namespace>

# Debug DNS
kubectl exec -it <pod-name> -n <namespace> -- cat /etc/resolv.conf
```

## Storage issues:

Bash

```
# Sprawdź persistent volumes
kubectl get pv

# Sprawdź persistent volume claims
kubectl get pvc -n <namespace>

# Sprawdź storage classes
kubectl get storageclass
```



```
# Sprawdź volume mounts w podzie
kubectl describe pod <pod-name> -n <namespace> | grep -A 10 "Mounts:"
```

## CI/CD troubleshooting

### Jenkins pipeline failures:

Bash

```
# Sprawdź logi Jenkins
docker logs jenkins-container

# Sprawdź workspace
ls -la /var/jenkins_home/workspace/<job-name>

# Sprawdź available disk space
df -h /var/jenkins_home

# Sprawdź Jenkins system log
tail -f /var/jenkins_home/logs/jenkins.log

# Test Docker connectivity
docker ps
docker images
```

### GitHub Actions failures:

YAML

```
# Debug step w workflow
- name: Debug
  run: |
    echo "Runner OS: $RUNNER_OS"
    echo "GitHub workspace: $GITHUB_WORKSPACE"
    echo "Environment variables:"
    env | sort
    echo "Available disk space:"
    df -h
```

### AWS deployment issues:

Bash

```
# Sprawdź CloudFormation stack
aws cloudformation describe-stacks --stack-name <stack-name>

# Sprawdź CloudFormation events
aws cloudformation describe-stack-events --stack-name <stack-name>

# Sprawdź EKS cluster status
aws eks describe-cluster --name <cluster-name>

# Sprawdź ECR repositories
aws ecr describe-repositories

# Sprawdź IAM roles i policies
aws iam get-role --role-name <role-name>
aws iam list-attached-role-policies --role-name <role-name>
```

## Database troubleshooting

### PostgreSQL performance issues:

#### SQL

```
-- Sprawdź aktywne połączenia
SELECT pid, username, application_name, client_addr, state, query_start, query
FROM pg_stat_activity
WHERE state = 'active';

-- Sprawdź długo działające queries
SELECT pid, now() - pg_stat_activity.query_start AS duration, query
FROM pg_stat_activity
WHERE (now() - pg_stat_activity.query_start) > interval '5 minutes';

-- Sprawdź blokady
SELECT blocked_locks.pid AS blocked_pid,
       blocked_activity.username AS blocked_user,
       blocking_locks.pid AS blocking_pid,
       blocking_activity.username AS blocking_user,
       blocked_activity.query AS blocked_statement,
       blocking_activity.query AS current_statement_in_blocking_process
FROM pg_catalog.pg_locks blocked_locks
JOIN pg_catalog.pg_stat_activity blocked_activity ON blocked_activity.pid =
blocked_locks.pid
JOIN pg_catalog.pg_locks blocking_locks ON blocking_locks.locktype =
blocked_locks.locktype
JOIN pg_catalog.pg_stat_activity blocking_activity ON blocking_activity.pid =
```

```
blocking_locks.pid
WHERE NOT blocked_locks.granted;

-- Sprawdź wykorzystanie indeksów
SELECT schemaname, tablename, attname, n_distinct, correlation
FROM pg_stats
WHERE tablename = '<table_name>';
```

## Redis troubleshooting:

Bash

```
# Połącz się z Redis
redis-cli -h <host> -p <port>

# Sprawdź informacje o serwerze
INFO

# Sprawdź memory usage
INFO memory

# Sprawdź slow queries
SLOWLOG GET 10

# Sprawdź connected clients
CLIENT LIST

# Monitor commands w real-time
MONITOR
```

## Network troubleshooting

### Connectivity issues:

Bash

```
# Test basic connectivity
ping <hostname>
telnet <hostname> <port>
nc -zv <hostname> <port>

# DNS resolution
nslookup <hostname>
dig <hostname>
```

```
# Route tracing
traceroute <hostname>
mtr <hostname>

# Check listening ports
netstat -tlnp
ss -tlnp

# Check firewall rules (iptables)
iptables -L -n
```

## Load balancer issues:

Bash

```
# AWS ALB troubleshooting
aws elbv2 describe-load-balancers
aws elbv2 describe-target-groups
aws elbv2 describe-target-health --target-group-arn <arn>

# Check ALB access logs
aws s3 ls s3://<bucket>/AWSLogs/<account-id>/elasticloadbalancing/

# Kubernetes ingress troubleshooting
kubectl get ingress -n <namespace>
kubectl describe ingress <ingress-name> -n <namespace>
kubectl logs -n ingress-nginx deployment/ingress-nginx-controller
```

## Performance tuning

### Application optimization

#### Python/Flask optimization:

Python

```
# Użyj connection pooling
from sqlalchemy import create_engine
from sqlalchemy.pool import QueuePool

engine = create_engine(
    DATABASE_URL,
    poolclass=QueuePool,
```

```

    pool_size=20,
    max_overflow=30,
    pool_pre_ping=True,
    pool_recycle=3600
)

# Implementuj caching
from flask_caching import Cache
cache = Cache(app, config={'CACHE_TYPE': 'redis'})

@cache.memoize(timeout=300)
def get_products():
    return Product.query.all()

# Użyj async processing dla heavy tasks
from celery import Celery
celery = Celery(app.name, broker=app.config['CELERY_BROKER_URL'])

@celery.task
def process_payment(payment_data):
    # Heavy processing task
    pass

```

## React optimization:

### JavaScript

```

// Lazy loading komponentów
const ProductList = React.lazy(() => import('./ProductList'));

// Memoization dla expensive calculations
const ExpensiveComponent = React.memo(({ data }) => {
  const expensiveValue = useMemo(() => {
    return data.reduce((acc, item) => acc + item.value, 0);
  }, [data]);

  return <div>{expensiveValue}</div>;
});

// Virtual scrolling dla długich list
import { FixedSizeList as List } from 'react-window';

const VirtualizedList = ({ items }) => (
  <List
    height={600}
    itemCount={items.length}
    itemSize={50}

```

```
>
  {{({ index, style }) => (
    <div style={style}>
      {items[index].name}
    </div>
  )}}
</List>
);
```

## Database optimization

### PostgreSQL tuning:

#### SQL

```
-- Optimalizacja konfiguracji
-- postgresql.conf
shared_buffers = 256MB
effective_cache_size = 1GB
work_mem = 4MB
maintenance_work_mem = 64MB
checkpoint_completion_target = 0.9
wal_buffers = 16MB
default_statistics_target = 100

-- Tworzenie indeksów
CREATE INDEX CONCURRENTLY idx_products_category_id ON products(category_id);
CREATE INDEX CONCURRENTLY idx_products_name_gin ON products USING
gin(to_tsvector('english', name));

-- Partitioning dla dużych tabel
CREATE TABLE orders_2024 PARTITION OF orders
FOR VALUES FROM ('2024-01-01') TO ('2025-01-01');

-- Vacuum i analyze
VACUUM ANALYZE products;
```

## Infrastructure optimization

### Kubernetes resource optimization:

#### YAML

```
# Proper resource requests i limits
resources:
  requests:
    memory: "256Mi"
    cpu: "250m"
  limits:
    memory: "512Mi"
    cpu: "500m"
```

```
# Horizontal Pod Autoscaler
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: app-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: app
  minReplicas: 3
  maxReplicas: 20
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
```

```
# Pod Disruption Budget
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: app-pdb
spec:
  minAvailable: 2
  selector:
    matchLabels:
      app: my-app
```

## AWS optimization:

Bash

```
# Right-sizing instances
aws ec2 describe-instances --query 'Reservations[].Instances[]'
```

```
[InstanceId,InstanceType,State.Name]'
```

```
# Spot instances dla non-critical workloads
aws ec2 request-spot-instances --spot-price "0.05" --instance-count 2 --type
"one-time"

# Reserved instances dla predictable workloads
aws ec2 describe-reserved-instances

# CloudWatch metrics dla optimization
aws cloudwatch get-metric-statistics --namespace AWS/EC2 --metric-name
CPUUtilization
```

## Disaster recovery

### Backup strategies

#### Database backups:

Bash

```
# PostgreSQL backup
pg_dump -h localhost -U username -d database_name > backup.sql

# Automated backup script
#!/bin/bash
BACKUP_DIR="/backups"
DATE=$(date +%Y%m%d_%H%M%S)
DB_NAME="ecommerce"

# Create backup
pg_dump -h $DB_HOST -U $DB_USER -d $DB_NAME | gzip >
$BACKUP_DIR/${DB_NAME}_${DATE}.sql.gz

# Upload to S3
aws s3 cp $BACKUP_DIR/${DB_NAME}_${DATE}.sql.gz s3://backup-bucket/database/

# Cleanup old backups (keep last 30 days)
find $BACKUP_DIR -name "*.sql.gz" -mtime +30 -delete
```

#### Kubernetes backups z Velero:

Bash



```
# Backup całego klastra
velero backup create cluster-backup-$(date +%Y%m%d)

# Backup konkretnego namespace
velero backup create app-backup --include-namespaces production

# Scheduled backups
velero schedule create daily-backup --schedule="0 2 * * *" --ttl 720h0m0s

# Restore z backup
velero restore create --from-backup cluster-backup-20240101
```

## Multi-region deployment

### AWS multi-region setup:

Plain Text

```
# Terraform configuration dla multi-region
provider "aws" {
  alias  = "primary"
  region = "eu-west-1"
}

provider "aws" {
  alias  = "secondary"
  region = "us-east-1"
}

# Primary region resources
module "primary_infrastructure" {
  source = "./modules/infrastructure"
  providers = {
    aws = aws.primary
  }
  region = "eu-west-1"
}

# Secondary region resources
module "secondary_infrastructure" {
  source = "./modules/infrastructure"
  providers = {
    aws = aws.secondary
  }
  region = "us-east-1"
}
```

```

}

# Cross-region replication
resource "aws_s3_bucket_replication_configuration" "replication" {
  role    = aws_iam_role.replication.arn
  bucket  = module.primary_infrastructure.s3_bucket_id

  rule {
    id      = "replicate-to-secondary"
    status  = "Enabled"

    destination {
      bucket          = module.secondary_infrastructure.s3_bucket_arn
      storage_class   = "STANDARD_IA"
    }
  }
}

```

## Recovery procedures

### RTO/RPO planning:

#### YAML

```

# Service tier definitions
services:
  critical:
    rto: 15 minutes
    rpo: 5 minutes
    examples: [payment-service, user-authentication]

  important:
    rto: 1 hour
    rpo: 30 minutes
    examples: [product-service, order-service]

  standard:
    rto: 4 hours
    rpo: 2 hours
    examples: [analytics-service, reporting-service]

```

### Automated failover:

#### Bash

```

#!/bin/bash
# Health check i failover script

PRIMARY_ENDPOINT="https://api.primary.example.com/health"
SECONDARY_ENDPOINT="https://api.secondary.example.com/health"
DNS_RECORD="api.example.com"

check_health() {
    local endpoint=$1
    local response=$(curl -s -o /dev/null -w "%{http_code}" $endpoint)
    echo $response
}

# Check primary health
primary_status=$(check_health $PRIMARY_ENDPOINT)

if [ "$primary_status" != "200" ]; then
    echo "Primary endpoint unhealthy, checking secondary..."

    secondary_status=$(check_health $SECONDARY_ENDPOINT)

    if [ "$secondary_status" == "200" ]; then
        echo "Failing over to secondary region..."

        # Update DNS to point to secondary
        aws route53 change-resource-record-sets --hosted-zone-id Z123456789 -
-change-batch '{
    "Changes": [{
        "Action": "UPSERT",
        "ResourceRecordSet": {
            "Name": "'$DNS_RECORD'",
            "Type": "CNAME",
            "TTL": 60,
            "ResourceRecords": [{"Value": "secondary.example.com"}]}
    }]
}'

        # Send alert
        aws sns publish --topic-arn arn:aws:sns:region:account:alerts \
            --message "Failover executed: Primary -> Secondary"
    fi
fi

```

To kończy rozdział o najlepszych praktykach i troubleshooting. Następny rozdział będzie podsumowaniem całego podręcznika.

## 23. Podsumowanie i referencje

### Podsumowanie podręcznika

Ten kompletny podręcznik DevOps przeprowadził Cię przez wszystkie kluczowe aspekty nowoczesnego DevOps, od podstawowych konceptów po zaawansowane implementacje w środowisku chmurowym. Omówiliśmy szczegółowo Git jako foundation dla version control i collaboration, Jenkins i inne narzędzia CI/CD dla automatyzacji pipeline'ów, Python jako język do automatyzacji i development, Terraform dla Infrastructure as Code, AWS jako platformę chmurową, oraz Kubernetes z EKS dla container orchestration.

Praktyczny projekt e-commerce demonstrował real-world implementation wszystkich omawianych technologii, pokazując jak integrować różne narzędzia w spójny ecosystem. Projekt obejmował microservices architecture, complete CI/CD pipeline, infrastructure automation, monitoring i observability, security best practices, oraz disaster recovery procedures.

Kluczowe takeaways z tego podręcznika obejmują importance of automation w każdym aspekcie DevOps, necessity of comprehensive monitoring i observability, critical role of security throughout the pipeline, value of Infrastructure as Code dla consistency i repeatability, oraz importance of cultural change w organizacji dla successful DevOps adoption.

### Droga dalszego rozwoju

DevOps to continuously evolving field, więc important jest staying current z latest trends i technologies. Zalecane obszary do dalszego eksplorowania obejmują GitOps z tools jak ArgoCD czy Flux, service mesh technologies jak Istio czy Linkerd, serverless computing z AWS Lambda czy Azure Functions, oraz emerging technologies jak WebAssembly czy edge computing.

Continuous learning powinno obejmować hands-on practice z new tools, participation w DevOps communities, attending conferences i meetups, oraz contributing to open-source projects. Certifications mogą być valuable dla career advancement - rozważ AWS Certified

DevOps Engineer, Kubernetes certifications (CKA, CKAD, CKS), czy HashiCorp Terraform Associate.

## Referencje i źródła

### Dokumentacja oficjalna

#### Git:

- [Git Documentation](#) - Oficjalna dokumentacja Git
- [Pro Git Book](#) - Kompletny przewodnik po Git
- [GitHub Docs](#) - Dokumentacja GitHub
- [GitLab Docs](#) - Dokumentacja GitLab

#### Jenkins:

- [Jenkins User Documentation](#) - Oficjalna dokumentacja Jenkins
- [Jenkins Pipeline Documentation](#) - Przewodnik po Jenkins Pipeline
- [Jenkins Plugin Index](#) - Katalog pluginów Jenkins

#### Python:

- [Python Documentation](#) - Oficjalna dokumentacja Python
- [Flask Documentation](#) - Dokumentacja Flask
- [Django Documentation](#) - Dokumentacja Django
- [FastAPI Documentation](#) - Dokumentacja FastAPI

#### Terraform:

- [Terraform Documentation](#) - Oficjalna dokumentacja Terraform
- [Terraform AWS Provider](#) - AWS Provider documentation
- [Terraform Best Practices](#) - Najlepsze praktyki

## **AWS:**

- [AWS Documentation](#) - Kompletna dokumentacja AWS
- [AWS Well-Architected Framework](#) - Architekturalne best practices
- [AWS EKS User Guide](#) - Przewodnik po EKS
- [AWS DevOps Blog](#) - Blog o DevOps w AWS

## **Kubernetes:**

- [Kubernetes Documentation](#) - Oficjalna dokumentacja Kubernetes
- [Kubernetes API Reference](#) - API Reference
- [Helm Documentation](#) - Dokumentacja Helm
- [Istio Documentation](#) - Dokumentacja Istio

## **Książki rekomendowane**

### **DevOps i kultura:**

- "The Phoenix Project" by Gene Kim, Kevin Behr, George Spafford
- "The DevOps Handbook" by Gene Kim, Jez Humble, Patrick Debois, John Willis
- "Accelerate" by Nicole Forsgren, Jez Humble, Gene Kim
- "Site Reliability Engineering" by Niall Richard Murphy, Betsy Beyer, Chris Jones, Jennifer Petoff

### **Techniczne:**

- "Infrastructure as Code" by Kief Morris
- "Kubernetes: Up and Running" by Kelsey Hightower, Brendan Burns, Joe Beda
- "Docker Deep Dive" by Nigel Poulton
- "Terraform: Up & Running" by Yevgeniy Brikman

## Cloud i AWS:

- "AWS Certified Solutions Architect Study Guide" by Ben Piper, David Clinton
- "Amazon Web Services in Action" by Andreas Wittig, Michael Wittig
- "Cloud Native DevOps with Kubernetes" by John Arundel, Justin Domingus

## Narzędzia i platformy

### CI/CD:

- [GitHub Actions](#) - CI/CD platform od GitHub
- [GitLab CI/CD](#) - Integrated CI/CD w GitLab
- [CircleCI](#) - Cloud-based CI/CD platform
- [Azure DevOps](#) - Microsoft DevOps platform

### Monitoring i Observability:

- [Prometheus](#) - Monitoring system i time series database
- [Grafana](#) - Analytics i monitoring platform
- [Jaeger](#) - Distributed tracing system
- [ELK Stack](#) - Elasticsearch, Logstash, Kibana

### Security:

- [OWASP](#) - Open Web Application Security Project
- [Snyk](#) - Security platform dla developers
- [Aqua Security](#) - Container i cloud security
- [Falco](#) - Runtime security monitoring

### Infrastructure as Code:

- [Pulumi](#) - Modern Infrastructure as Code

- [AWS CDK](#) - AWS Cloud Development Kit
- [Azure ARM Templates](#) - Azure Resource Manager
- [Google Cloud Deployment Manager](#) - Google Cloud IaC

## Społeczności i zasoby edukacyjne

### Konferencje:

- [DevOpsDays](#) - Global DevOps conference series
- [KubeCon + CloudNativeCon](#) - Kubernetes i cloud native
- [AWS re:Invent](#) - AWS annual conference
- [DockerCon](#) - Docker conference

### Online communities:

- [DevOps Subreddit](#) - Reddit community
- [CNCF Slack](#) - Cloud Native Computing Foundation
- [Kubernetes Slack](#) - Kubernetes community
- [DevOps Chat](#) - DevOps discussions

### Kursy online:

- [Linux Academy](#) - Cloud i DevOps training
- [A Cloud Guru](#) - Cloud certification training
- [Pluralsight](#) - Technology skills platform
- [Udemy](#) - Online learning marketplace

### Blogi i newslettery:

- [DevOps.com](#) - DevOps news i articles
- [The New Stack](#) - Cloud native i DevOps



- [Container Journal](#) - Container technologies
- [AWS News Blog](#) - AWS updates i announcements

## Certyfikacje

### **AWS:**

- AWS Certified DevOps Engineer - Professional
- AWS Certified Solutions Architect - Associate/Professional
- AWS Certified SysOps Administrator - Associate

### **Kubernetes:**

- Certified Kubernetes Administrator (CKA)
- Certified Kubernetes Application Developer (CKAD)
- Certified Kubernetes Security Specialist (CKS)

### **HashiCorp:**

- HashiCorp Certified: Terraform Associate
- HashiCorp Certified: Vault Associate
- HashiCorp Certified: Consul Associate

### **Docker:**

- Docker Certified Associate (DCA)

### **Google Cloud:**

- Google Cloud Professional DevOps Engineer
- Google Cloud Professional Cloud Architect

### **Microsoft Azure:**

- Azure DevOps Engineer Expert

- [Azure Solutions Architect Expert](#)

## Narzędzia development

### IDE i edytory:

- [Visual Studio Code](#) - Lightweight code editor
- [IntelliJ IDEA](#) - Java IDE
- [PyCharm](#) - Python IDE
- [Vim](#) - Text editor

### Version control:

- [Git](#) - Distributed version control
- [GitHub](#) - Git hosting platform
- [GitLab](#) - Complete DevOps platform
- [Bitbucket](#) - Git hosting by Atlassian

### Container tools:

- [Docker](#) - Container platform
- [Podman](#) - Daemonless container engine
- [Buildah](#) - Container image builder
- [Skopeo](#) - Container image utility

## Zakończenie

DevOps to journey, nie destination. Technologie i praktyki ciągle ewoluują, ale fundamentalne principles pozostają stałe: automation, collaboration, continuous improvement, oraz focus na delivering value to customers. Ten podręcznik dostarczył Ci solid foundation, ale real learning comes from hands-on practice i real-world experience.

Pamiętaj, że successful DevOps implementation wymaga nie tylko technical skills, ale także cultural change w organizacji. Invest w building relationships z team members, practice empathy i communication skills, oraz always be willing to learn from failures.

Zyczę powodzenia w Twojej DevOps journey! Pamiętaj o continuous learning, sharing knowledge z community, oraz contributing back to open-source projects gdy będziesz gotowy.

---

**Autor:** Manus AI

**Wersja:** 1.0

**Data:** Styczeń 2025

**Licencja:** Creative Commons Attribution-ShareAlike 4.0 International

---

*Ten podręcznik został stworzony z myślą o praktycznym zastosowaniu. Wszystkie przykłady kodu i konfiguracje zostały przetestowane, ale zawsze dostosuj je do swoich specific requirements i security policies.*