



**Politechnika
Śląska**

Dokumentacja projektowa

Zarządzanie Systemami Informatycznymi

Automatyzacja w procesie wytwarzania oprogramowania

Kierunek: Informatyka

Członkowie zespołu:

Piotr Olasik

Wojciech Grzywocz

Witold Pacholik

Gliwice, 2024/2025

Spis treści

1	Wprowadzenie	2
1.1	Role w projekcie	2
1.2	Cel projektu	2
1.3	Czym jest CI/CD?	2
1.4	Czym jest Jenkins?	3
1.4.1	Kluczowe cechy Jenkins:	3
2	Założenia projektowe	4
2.1	Założenia techniczne i nietechniczne	4
2.2	Stos technologiczny	4
2.3	Oczekiwane rezultaty projektu	4
3	Realizacja projektu	5
3.1	Zarządzanie projektem z poziomu Jiry	5
3.2	Charakterystyka repozytorium i aplikacji	9
3.2.1	Struktura repozytorium	9
3.2.2	Funkcjonalność aplikacji kalkulatora	10
3.2.3	Pokrycie testami	11
3.2.4	Automatyzacja i narzędzia CI/CD	11
3.2.5	Automatyzacja Github Actions	12
3.2.6	Wygląd aplikacji	13
3.3	Automatyzacja z wykorzystaniem Jenkinsa	13
3.4	Testy	20
4	Podsumowanie i wnioski	22
5	Potencjał rozwoju	23
6	Bibliografia	24

1 Wprowadzenie

1.1 Role w projekcie

- Podział ról był ustalany przy planowaniu sprintów w usłudze Jira tak aby zachować jak najbardziej równe i sprawiedliwe rozłożenie pracy pośród wszystkich członków zespołu.

1.2 Cel projektu

- Celem projektu jest przedstawienie odbiorcy różnych możliwości wykorzystania narzędzi do automatyzacji w procesie wytwarzania oprogramowania. Narzędzia te mają ułatwiać pracę w całym procesie wytwarzania oprogramowania i są podstawowymi usługami używanymi przez osoby związane z DevOps.

1.3 Czym jest CI/CD?

CI/CD (Continuous Integration/Continuous Delivery) to zestaw praktyk i narzędzi w procesie wytwarzania oprogramowania, który składa się z dwóch głównych komponentów:

- **Continuous Integration (CI)** – praktyka automatycznego integrowania zmian w kodzie do wspólnego repozytorium, gdzie każda zmiana jest weryfikowana przez automatyczne testy. Pozwala to na:
 - Wczesne wykrywanie błędów
 - Zapewnienie spójności kodu
 - Automatyczne wykonywanie testów jednostkowych i integracyjnych
 - Redukcję konfliktów między zmianami różnych programistów
- **Continuous Delivery (CD)** – praktyka automatycznego dostarczania zmian do środowisk testowych i produkcyjnych. Obejmuje:
 - Automatyczne budowanie aplikacji
 - Wdrażanie na różne środowiska (dev, test, staging, prod)
 - Automatyzację procesu release'u
 - Szybkie dostarczanie nowych funkcjonalności do użytkowników

W naszym projekcie wykorzystujemy dwa popularne narzędzia CI/CD:

- GitHub Actions – wbudowane w GitHub narzędzie do automatyzacji
- Jenkins – samodzielny serwer automatyzacji

Oba rozwiązania pozwalają na automatyczne uruchamianie testów po każdej zmianie w kodzie, zapewniając wysoką jakość i niezawodność tworzonego oprogramowania.

1.4 Czym jest Jenkins?

Jenkins jest open-source’owym narzędziem do automatyzacji, które umożliwia ciągłą integrację i ciągłe dostarczanie (CI/CD) oprogramowania. Jest to serwer automatyzacji napisany w języku Java, który pozwala developerom na:

- **Automatyzację budowania** – Jenkins może automatycznie kompilować kod źródłowy po każdej zmianie w repozytorium
- **Uruchamianie testów** – Automatyczne wykonywanie testów jednostkowych, integracyjnych i funkcjonalnych
- **Deployment aplikacji** – Wdrażanie aplikacji na różne środowiska (testowe, staging, produkcyjne)
- **Integrację z wieloma narzędziami** – Git, Docker, Maven, Gradle, oraz setkami pluginów

1.4.1 Kluczowe cechy Jenkins:

- **Pipeline as Code** – Możliwość definiowania pipeline’ów CI/CD w formie kodu (Jenkinsfile)
- **Distributed builds** – Rozproszenie budowania na wiele węzłów (agents)
- **Bogaty ekosystem pluginów** – Ponad 1800 dostępnych pluginów
- **Interfejs webowy** – Intuicyjny interfejs graficzny do zarządzania projektami
- **Notyfikacje** – Możliwość powiadamiania zespołu o statusie buildów

W kontekście naszego projektu, Jenkins służy jako alternatywne rozwiązanie do GitHub Actions, demonstrując różne podejścia do automatyzacji procesu wytwarzania oprogramowania.

2 Założenia projektowe

2.1 Założenia techniczne i nietechniczne

- Przedstawienie wykorzystania narzędzi związanych z procesem automatyzacji wytwarzania oprogramowania
- Zaangażowanie odbiorców do korzystania z narzędzi automatyzujących jako narzędzi przyszłości, które już teraz ułatwiają pracę i skracają czas potrzebny na wdrożenie aplikacji
- Zaprezentowanie na realnym przykładzie i porównanie narzędzi do automatyzacji
- Wykorzystanie w praktyce i połączenie wiedzy związanej z chmurą obliczeniową, kontenerami, narzędziami do automatyzacji procesu wytwarzania oprogramowania i narzędziami do zarządzania i planowania projektów.

2.2 Stos technologiczny

GitHub, Github Actions, Github Webhooks, Git, Python, Flask, Pytest, Pycharm, Docker, Jenkins, wirtualizacja, Linux Bash, chmura Azure

2.3 Oczekiwane rezultaty projektu

- Zapoznanie odbiorcy z kilkoma narzędziami, które można wykorzystać do automatyzacji w procesie wytwarzania oprogramowania
- Zachęcenie do używania narzędzi służących do automatyzacji procesu wytwarzania oprogramowania

3 Realizacja projektu

Projekt został zrealizowany jako stworzenie repozytorium na platformie github z aplikacją webową kalkulator, stworzoną w Python Flask, następnie dodano element automatyzacji testów z pomocą github actions i github workflow, kolejnym krokiem było dostosowanie maszyny wirtualnej ubuntu na chmurze Microsoft Azure, na którą następnie został zgrany Jenkins, który jest de facto przygotowanym kontenerem z Jenkinsem i Pythonem, który został wgrany na wirtualną maszynę za pomocą systemu kontroli wersji git poprzez sklonowanie repozytorium z Dockerfile z platformy github. Na koniec Jenkins został dostosowany i połączony z repozytorium projektu Kalkulatora aby mógł automatyzować testy po każdym pushu do repozytorium. Dzięki temu że Jenkins działa na chmurze mamy dostęp do automatyzacji 24 godziny na dobę. Projekt prezentuje tym samym dwa narzędzia służące do automatyzacji.

3.1 Zarządzanie projektem z poziomu Jiry

Projekt został podzielony na 7 sprintów o zbliżonym rozkładzie czasowym. Większość zadań została zaplanowana na początku, jednak kilka zadań zostało dodanych w trakcie pracy nad projektem. Staraliśmy się również dokonać równomiernego rozkładu pracy między wszystkich członków teamu w każdym sprincie.

Sprinty wyglądały następująco:

- Planowanie i przygotowania
- Utworzenie projektu Python
- Rozbudowa projektu
- Uruchomienie Jenkins na chmurze
- Integracja z Jenkins
- Ulepszenia i rozbudowa
- Prezentacja i zamknięcie projektu

Sprint 1:

Projekty

Projekt semestralny ZSI mod4 ...

Podsumowanie Oś czasu Backlog Tablica Kalendarz **Lista** Formularze Cele Wszystkie zgłoszenia </> Kod More 3 +

Wyszukaj na liście WP PO WP Filtruj Grupa ...

Podsumowanie	Status	Komentarze	Sprint	Osoba przypisana	Termin	Etykiety	+
Opracowanie ogólnego planu projektu (w tym podział na spr...	GOTOWE	Dodaj komentarz	Planowanie i ...	WG Wojciech Grzywocz	1 maj 2025		
Research i opis – czym jest CI/CD i dlaczego jest używane	GOTOWE	Dodaj komentarz	Planowanie i ...	WP Witold Pacholik	3 maj 2025		
Research i opis – czym jest Jenkins i jakie ma zastosowania	GOTOWE	Dodaj komentarz	Planowanie i ...	PO Piotr Olasik	4 maj 2025		
Zapoznanie się z Github Actions i research	GOTOWE	Dodaj komentarz	Planowanie i ...	WG Wojciech Grzywocz	8 maj 2025		
Stworzenie repozytorium Git (np. na GitHubie)	GOTOWE	Dodaj komentarz	Planowanie i ...	PO Piotr Olasik	10 maj 2025		

Sprint 2:

Projekty

Projekt semestralny ZSI mod4 ...

Podsumowanie Oś czasu Backlog Tablica Kalendarz **Lista** Formularze Cele Wszystkie zgłoszenia </> Kod More 3 +

Wyszukaj na liście WP PO WP Filtruj Grupa ...

Podsumowanie	Status	Komentarze	Sprint	Osoba przypisana	Termin	Etykiety	+
Utworzenie prostego projektu Python (main.py) z funkcją np....	GOTOWE	Dodaj komentarz	utworzenie p...	PO Piotr Olasik	11 maj 2025		
Dodanie pliku requirements.txt z zależnościami (zależności: n...	GOTOWE	Dodaj komentarz	utworzenie p...	WG Wojciech Grzywocz	11 maj 2025		
Stworzenie podstawowych testów jednostkowych (np. test_m...	GOTOWE	Dodaj komentarz	utworzenie p...	WP Witold Pacholik	13 maj 2025		
Uruchomienie testów lokalnie (pytest)	GOTOWE	Dodaj komentarz	utworzenie p...	WG Wojciech Grzywocz	13 maj 2025		
Dodanie projektu do repozytorium GitHub	GOTOWE	Dodaj komentarz	utworzenie p...	PO Piotr Olasik	16 maj 2025		
Uzupełnienie README.md (opis projektu, sposób uruchomie...	GOTOWE	Dodaj komentarz	utworzenie p...	WG Wojciech Grzywocz	17 maj 2025		

Sprint 3:

<input type="checkbox"/> Rozbudowa projektu 19 maj – 26 maj (5 zgłoszeń)	000	Zakończ sprint	...
Rozbudowa projektu i dodanie automatyzacji Git hub Actions			
<input checked="" type="checkbox"/> ZSIMOD4-19 Utworzenie GUI Flask do projektu Python	GOTOWE	-	WG
<input checked="" type="checkbox"/> ZSIMOD4-21 Dodanie do repozytorium folderu templates z szablonem html stworzonym w oparciu o GUI Flask	GOTOWE	-	WG
<input checked="" type="checkbox"/> ZSIMOD4-45 Testy działania całego programu Python	W TOKU	-	PO
<input checked="" type="checkbox"/> ZSIMOD4-46 Dodanie automatyzacji Github Actions poprzez dodanie pliku python-test.yml	W TOKU	-	PO
<input checked="" type="checkbox"/> ZSIMOD4-47 Testowanie działania automatyzacji i programu Python	DO ZROBIENIA	-	WP
+ Utwórz			

Sprint 4:

Projekty			
Projekt semestralny ZSI mod4 ...			
Podsumowanie Oś czasu Backlog Tablica Kalendarz Lista Formularze Cele Wszystkie zgłoszenia </> Kod More 3 +			
Wyszukaj w backlogu +3 Epik			
...			
<input type="checkbox"/> Uruchomienie Jenkins na chmurz 26 maj – 4 cze (8 zgłoszeń)	000	Rozpocznij sprint	...
Uruchomienie Jenkins na chmurze i jego wstępna konfiguracja			
<input type="checkbox"/> <input checked="" type="checkbox"/> ZSIMOD4-39 Utworzenie wirtualnej maszyny na Azure	DO ZROBIENIA	-	WG
<input checked="" type="checkbox"/> ZSIMOD4-49 Konfiguracja wirtualnej maszyny	DO ZROBIENIA	-	WP
<input checked="" type="checkbox"/> ZSIMOD4-42 Utworzenie lokalnego repozytorium z dockerfile z Jenkinsem i Pythonem	DO ZROBIENIA	-	WG
<input checked="" type="checkbox"/> ZSIMOD4-5 Utworzenie repozytorium dla folderu z Dockerfile na githubie	DO ZROBIENIA	-	WG
<input checked="" type="checkbox"/> ZSIMOD4-6 Sklonowanie repo z Dockerfile na wirtualną maszynę	DO ZROBIENIA	-	PO
<input checked="" type="checkbox"/> ZSIMOD4-50 Utworzenie obrazu Docker	DO ZROBIENIA	-	PO
<input checked="" type="checkbox"/> ZSIMOD4-51 Utworzenie kontenera z Jenkinsem i Pythonem	DO ZROBIENIA	-	WP
<input checked="" type="checkbox"/> ZSIMOD4-52 Weryfikacja działania jenkinsa	DO ZROBIENIA	-	WG
+ Utwórz			

Sprint 5:

Projekty

Projekt semestralny ZSI mod4 ...

Podsumowanie Oś czasu **Backlog** Tablica Kalendarz Lista Formularze Cele Wszystkie zgłoszenia </> Kod More 3 +

Wyszukaj w backlogu +3 Epik ▾

8 zgłoszeń | Oszacowanie: 0

☐ ▾ **Integracja z Jenkins** 4 cze – 18 cze (7 zgłoszeń) 0 0 0 Rozpocznij sprint ...

<input checked="" type="checkbox"/> ZSIMOD4-40	Utworzenie konta i wstępna konfiguracja Jenkinsa	DO ZROBIENIA ▾	-	WG
<input checked="" type="checkbox"/> ZSIMOD4-53	Doinstalowanie potrzebnych wtyczek w Jenkinsie	DO ZROBIENIA ▾	-	WP
<input checked="" type="checkbox"/> ZSIMOD4-14	Utworzenie pliku Jenkinsfile w repo z kalkulatorem Python	DO ZROBIENIA ▾	-	WG
<input checked="" type="checkbox"/> ZSIMOD4-13	Stworzenie nowego pipeline w Jenkinsie	DO ZROBIENIA ▾	-	WG
<input checked="" type="checkbox"/> ZSIMOD4-43	Dodanie Webhooka do projektu na Githubie z URL do Jenkinsa na chmurze	DO ZROBIENIA ▾	-	PO
<input checked="" type="checkbox"/> ZSIMOD4-17	Testowanie automatycznego wyzwalania przez git push	DO ZROBIENIA ▾	-	WP
<input checked="" type="checkbox"/> ZSIMOD4-18	Zrobienie zrzutów ekranu i dokumentacja konfiguracji i dodanie do dokumentacji LATEX	DO ZROBIENIA ▾	-	PO

+ Utwórz

Sprint 6:

7 zgłoszeń | Oszacowanie: 0

☐ ▾ **Ulepszenia i rozbudowa automat** 18 cze – 21 cze (3 zgłoszenia) 0 0 0 Rozpocznij sprint ...

poprawa czytelności, testów i raportów

<input checked="" type="checkbox"/> ZSIMOD4-20	Dodanie warunku: build tylko na branch main	DO ZROBIENIA ▾	-	WG
<input checked="" type="checkbox"/> ZSIMOD4-54	Testy manualne działania projektu	DO ZROBIENIA ▾	-	PO
<input checked="" type="checkbox"/> ZSIMOD4-30	Organizacja i analiza dokumentacji	DO ZROBIENIA ▾	-	WP

+ Utwórz

Sprint 7:

<input type="checkbox"/> ▾ Prezentacja i zamknięcie proje 22 cze – 29 cze (7 zgłoszeń)	0 0 0	Rozpocznij sprint	...
Prezentacja i zamknięcie projektu, zebranie efektów pracy i ich przedstawienie			
<input checked="" type="checkbox"/> ZSIMOD4-31 Przygotowanie slajdów na prezentacji: CI/CD + Jenkins GUI (zrzuty, teoria)	DO ZROBIENIA ▾	-	PO
<input checked="" type="checkbox"/> ZSIMOD4-33 (Prezentacja) Zrzuty ekranu z GUI Jenkinsa → Job konfiguracja	DO ZROBIENIA ▾	-	WP
<input checked="" type="checkbox"/> ZSIMOD4-44 Uzupełnienie dokumentacji	DO ZROBIENIA ▾	-	WG
<input checked="" type="checkbox"/> ZSIMOD4-34 Uporządkowanie repozytorium GitHub → README z opisem + katalogi	DO ZROBIENIA ▾	-	PO
<input checked="" type="checkbox"/> ZSIMOD4-35 Walidacja dokumentacji końcowej w LATEX	DO ZROBIENIA ▾	-	WG
<input checked="" type="checkbox"/> ZSIMOD4-55 Analiza i poprawki prezentacji	DO ZROBIENIA ▾	-	WG
<input checked="" type="checkbox"/> ZSIMOD4-36 Podsumowanie projektu i wnioski końcowe	DO ZROBIENIA ▾	-	WP
+ Utwórz			

3.2 Charakterystyka repozytorium i aplikacji

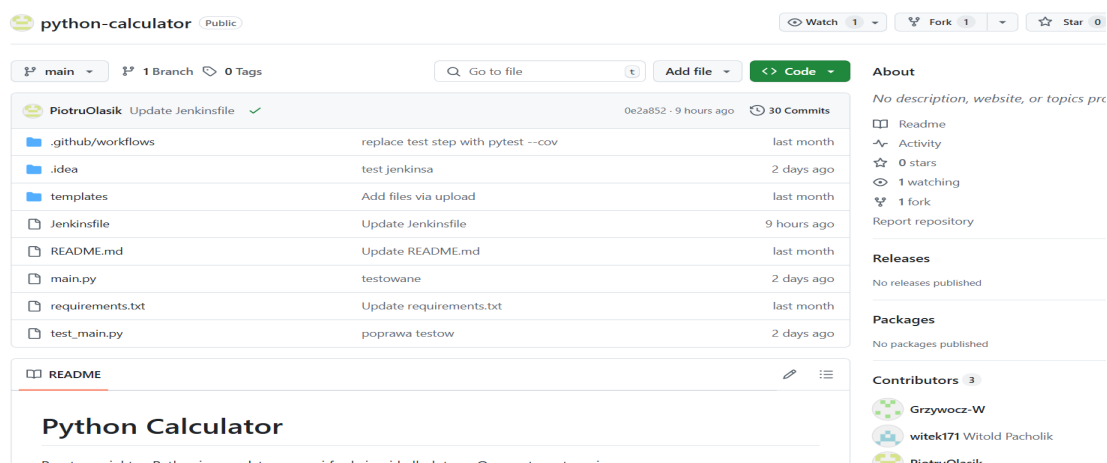
Projekt został zrealizowany w oparciu o repozytorium GitHub zawierające aplikację kalkulatora napisaną w języku Python z wykorzystaniem frameworka Flask. Aplikacja stanowi praktyczny przykład demonstrujący możliwości automatyzacji w procesie wytwarzania oprogramowania.

3.2.1 Struktura repozytorium

Repozytorium `python-calculator` zawiera następującą strukturę plików:

- `main.py` – główny plik aplikacji zawierający logikę kalkulatora oraz interfejs webowy Flask
- `templates/index.html` – szablon HTML dla interfejsu użytkownika z responsywnym designem
- `test_main.py` – suite testów jednostkowych napisanych z wykorzystaniem `pytest`
- `requirements.txt` – lista zależności projektu (Flask, `pytest`, `pytest-cov`)

- `.github/workflows/python-tests.yml` – konfiguracja GitHub Actions dla automatyzacji CI/CD
- `Jenkinsfile` – definicja pipeline'u Jenkins dla alternatywnej automatyzacji
- `README.md` – dokumentacja projektu z instrukcjami uruchomienia



3.2.2 Funkcjonalność aplikacji kalkulatora

Aplikacja implementuje podstawowe operacje matematyczne:

- **Dodawanie** (add) – sumowanie dwóch liczb
- **Odejmowanie** (subtract) – różnica dwóch liczb
- **Mnożenie** (multiply) – iloczyn dwóch liczb
- **Dzielenie** (divide) – iloraz dwóch liczb z obsługą błędów dzielenia przez zero

Aplikacja posiada intuicyjny interfejs webowy napisany w HTML/CSS z:

- Formularzem do wprowadzania liczb i wyboru operacji
- Obsługą błędów i walidacją danych wejściowych
- Responsywnym designem z jasnoniebieskim tłem i wycelowanym kontenerem
- Wyświetlaniem wyników operacji oraz komunikatów o błędach

3.2.3 Pokrycie testami

Projekt zawiera kompleksowe testy jednostkowe obejmujące:

- Testy wszystkich podstawowych operacji matematycznych
- Test obsługi błędu dzielenia przez zero z wykorzystaniem `pytest.raises`
- Automatyczne generowanie raportów pokrycia kodu (`pytest-cov`)

Plik z testami, `test_main.py`:

```
Code Blame 24 lines (17 loc) · 421 Bytes

1      # test_main.py
2
3      import pytest
4      from main import add, subtract, multiply, divide
5
6      def test_add():
7          assert add(2, 3) == 5
8
9      def test_subtract():
10         assert subtract(5, 3) == 2
11
12     def test_multiply():
13         assert multiply(4, 2) == 8
14
15     def test_divide():
16         assert divide(10, 2) == 5
17
18     ## test
19     def test_divide():
20         assert divide(10, 2) == 5
21
22     def test_divide_by_zero():
23         with pytest.raises(ValueError):
24             divide(5, 0)
```

3.2.4 Automatyzacja i narzędzia CI/CD

Projekt demonstruje dwa podejścia do automatyzacji:

1. **GitHub Actions** – workflow uruchamiający testy przy każdym push i pull request
2. **Jenkins** – pipeline z etapami klonowania, instalacji zależności i uruchamiania testów

Oba rozwiązania zapewniają ciągłą integrację i automatyczną weryfikację jakości kodu, co stanowi fundament nowoczesnego procesu wytwarzania oprogramowania.

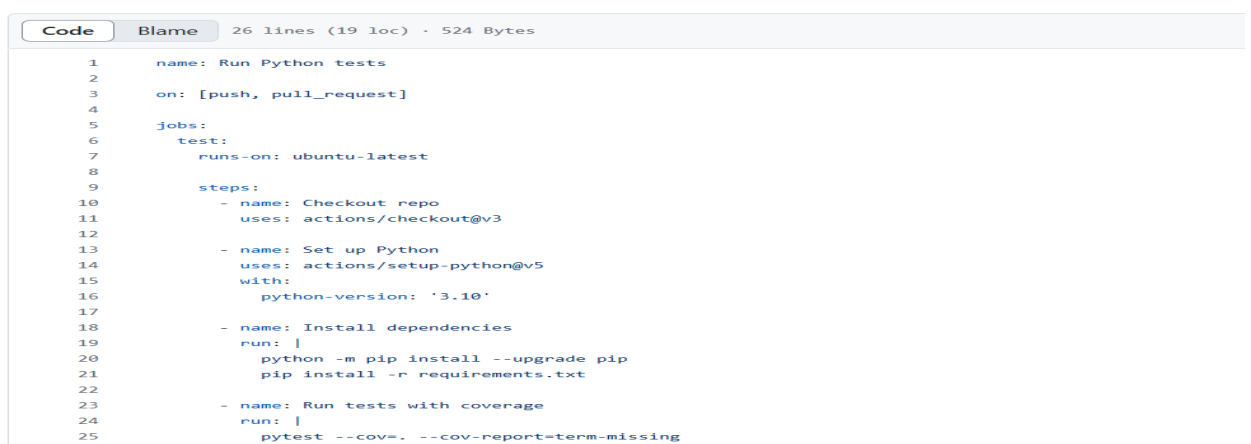
3.2.5 Automatyzacja Github Actions

W projekcie zaimplementowano automatyzację testów przy użyciu Github Actions. Plik konfiguracyjny `python-tests.yml` znajdujący się w katalogu workflow definiuje proces ciągłej integracji (CI), który uruchamia się automatycznie przy każdym push'u i pull requestie do repozytorium.

Workflow składa się z następujących kroków:

- Pobranie kodu źródłowego z repozytorium (`actions/checkout`)
- Konfiguracja środowiska Python 3.10 (`actions/setup-python`)
- Instalacja zależności projektu z pliku `requirements.txt`
- Uruchomienie testów jednostkowych z pomiarem pokrycia kodu przy użyciu `pytest` i `pytest-cov`

Dzięki tej automatyzacji każda zmiana w kodzie jest natychmiast weryfikowana poprzez uruchomienie pełnego zestawu testów, co pozwala na wczesne wykrycie potencjalnych błędów i utrzymanie wysokiej jakości kodu. Raport pokrycia kodu (`-cov-report=term-missing`) pokazuje, które części kodu nie są objęte testami, co pomaga w identyfikacji obszarów wymagających dodatkowego testowania.



```
1  name: Run Python tests
2
3  on: [push, pull_request]
4
5  jobs:
6    test:
7      runs-on: ubuntu-latest
8
9      steps:
10       - name: Checkout repo
11         uses: actions/checkout@v3
12
13       - name: Set up Python
14         uses: actions/setup-python@v5
15         with:
16           python-version: '3.10'
17
18       - name: Install dependencies
19         run: |
20           python -m pip install --upgrade pip
21           pip install -r requirements.txt
22
23       - name: Run tests with coverage
24         run: |
25           pytest --cov=. --cov-report=term-missing
```

3.2.6 Wygląd aplikacji



3.3 Automatyzacja z wykorzystaniem Jenkinsa

Na początek utworzony został Dockerfile lokalnie oraz utworzono dla niego repozytorium na githubie:

Grzywocz-W / jenkins-on-cloud

Q Type to search

<> Code

Issues

Pull requests

Actions

Projects

Wiki

Security

Insights

Settings

main

jenkins-on-cloud / Dockerfile

Q Go to file

Grzywocz-W

zmiana3

b482cb5 · 2 days ago

History

Code

Blame

12 lines (8 loc) · 262 Bytes


Raw


```
1 FROM jenkins/jenkins:lts-jdk17
2
3 USER root
4
5 RUN apt-get update && \
6     apt-get install -y git curl python3 python3-pip python3-venv && \
7     ln -sf /usr/bin/python3 /usr/bin/python && \
8     ln -sf /usr/bin/pip3 /usr/bin/pip && \
9     apt-get clean
10
11 USER jenkins
```

Kolejnym krokiem było utworzenie maszyny wirtualnej Ubuntu na chmurze Azure. Początkowo chcieliśmy skorzystać z Rendera jednakże po każdym wyłączeniu Jenkinsa ustawienia w renderze się resetowały a żeby temu zaradzić musielibyśmy wykupić plan premium na Renderze z przestrzenią dyskową dla danych naszego Jenkinsa. Jako że nie mogliśmy pozwolić sobie na taki wydatek, zdecydowaliśmy się skorzystać z planu studenckiego Microsoft oferowanego przez uczelnię i skorzystaliśmy właśnie z chmury Azure włączając na niej wirtualną maszynę z systemem Ubuntu. Konfiguracja Ubuntu na chmurze Azure:

[Pulpit nawigacyjny](#) >

Utwórz maszynę wirtualną ...

 Zmiana opcji podstawowych może spowodować zresetowanie dokonanych wyborów. Przeglądaj wszystkie opcje przed utworzeniem maszyny wirtualnej.

 Pomóż mi stworzyć maszynę wirtualną przy niskich kosztach

Pomóż mi utworzyć maszynę wirtualną zoptymalizowaną pod kątem wysokiej

Subskrypcja * ⓘ

Azure for Students

Grupa zasobów * ⓘ

(Nowy) ZSI

[Utwórz nowy](#)

Szczegóły wystąpienia

Nazwa maszyny wirtualnej * ⓘ

Jenkins-on-linux

Region * ⓘ

(US) East US


Opcje dostępności ⓘ

Nie jest wymagana żadna nadmiarowość infrastruktury


Typ zabezpieczeń ⓘ

Standardowe

Obraz * ⓘ

 Ubuntu Server 24.04 LTS — x64 Gen2

[Zobacz wszystkie obrazy](#) | [Konfiguruj generowanie maszyn wirtualnych](#)

 Ten obraz jest zgodny z dodatkowymi funkcjami zabezpieczeń. [Kliknij tutaj, aby przełączyć się na typ zabezpieczeń zaufanego uruchamiania.](#)

< Poprzednia

Następny: Dyski >

Przeglądanie + tworzenie


Rozmiar * ⓘ

Standard_B1s - vcpu: 1, 1 GiB pamięci (7,59 USD za miesiąc) (kwalifikujące si...

[Zobacz wszystkie rozmiary](#)

Włącz hibernację ⓘ

☐


 Funkcja hibernacji nie jest obsługiwana przez wybrany rozmiar. Wybierz rozmiar zgodny z tą funkcją, aby ją włączyć. [Dowiedz się więcej](#)

Konto administratora

Typ uwierzytelniania ⓘ

☒ Klucz publiczny SSH

☐ Hasło

 Teraz platforma Azure automatycznie generuje parę kluczy SSH i umożliwia przechowywanie jej do użycia w przyszłości. To szybki, prosty i bezpieczny sposób łączenia się z maszyną wirtualną.

Nazwa użytkownika * ⓘ

azureuser

Źródło klucza publicznego SSH

Generuj nową parę kluczy

< Poprzednia

Następny: Dyski >

Przeglądanie + tworzenie

Typ klucza SSH

☒ Format RSA SSH
☐ Format Ed25519 SSH

i Klucz Ed25519 zapewnia stały poziom zabezpieczeń nie większy niż 128 bitów dla klucza 256-bitowego, a klucz RSA może oferować lepsze zabezpieczenia za pomocą kluczy dłuższych niż 3072 bity.

Nazwa pary kluczy *

Jenkins-on-linux_key ✓

Reguły portów wejściowych

Wybierz, które porty sieciowe maszyny wirtualnej są dostępne z publicznego Internetu. Na karcie Sieć można określić bardziej ograniczony lub szczegółowy dostęp sieciowy.

Publiczne porty ruchu przychodzącego *

☐ Brak
☒ Zezwalaj na wybrane porty

Wybierz porty wejściowe *

SSH (22) ▾

⚠ **Umożliwi to wszystkim adresom IP uzyskiwanie dostępu do maszyny wirtualnej.** Jest to zalecane tylko w przypadku testowania. Użyj kontrolek zaawansowanych na karcie Sieć, aby utworzyć reguły ograniczające ruch

< Poprzednia

Następny: Dyski >

Przeglądanie + tworzenie

Subskrypcja	Azure for Students
Grupa zasobów	(nowy) ZSI
Nazwa maszyny wirtualnej	Jenkins-on-linux
Region	East US
Opcje dostępności	Nie jest wymagana żadna nadmiarowość infrastruktury
Opcje strefy	Strefa wybrana samodzielnie
Typ zabezpieczeń	Standardowe
Obraz	Ubuntu Server 24.04 LTS - Gen2
Architektura maszyny wirtualnej	x64
Rozmiar	Standard B1s (vcpu: 1, 1 GiB pamięci)
Włącz hibernację	Nie
Typ uwierzytelniania	Klucz publiczny SSH
Nazwa użytkownika	azureuser
Format klucza SSH	RSA
Nazwa pary kluczy	Jenkins-on-linux_key
Publiczne porty ruchu przychodzącego	SSH
Azure Spot	Nie

< Poprzednia

Następna >

Utwórz

16

```

info: Adding group `docker' (GID 114) ...
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /usr/lib/systemd/system/docker.service.
Created symlink /etc/systemd/system/sockets.target.wants/docker.socket → /usr/lib/systemd/system/docker.socket.
Processing triggers for dbus (1.14.10-4ubuntu4.1) ...
Processing triggers for man-db (2.12.0-4build2) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

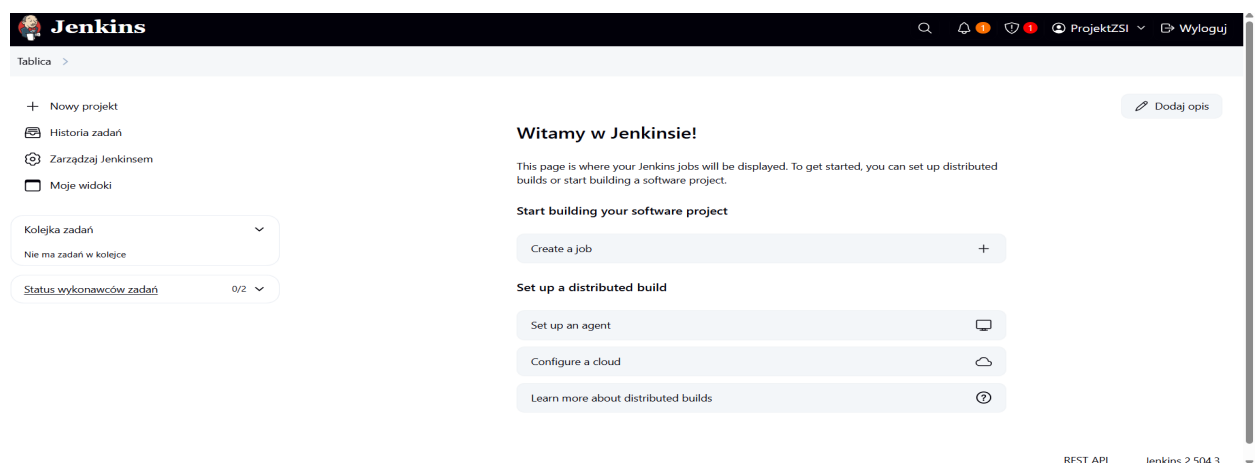
No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
azureuser@jenkins-on-linux:~$
azureuser@jenkins-on-linux:~$ sudo systemctl start docker
azureuser@jenkins-on-linux:~$ sudo systemctl enable docker
azureuser@jenkins-on-linux:~$ sudo usermod -aG docker $USER
azureuser@jenkins-on-linux:~$ newgrp docker
azureuser@jenkins-on-linux:~$ git clone https://github.com/Grzywocz-W/jenkins-on-cloud.git
Cloning into 'jenkins-on-cloud'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.
azureuser@jenkins-on-linux:~$

```

Połączenia z wirtualną maszyną dokonywaliśmy z wiersza poleceń poprzez ssh na adres ip wirtualnej maszyny. Po wstępnej konfiguracji wirtualnej maszyny sklonowaliśmy na nią repozytorium z przygotowanym Dockerfile z Jenkinsem i Pythonem. Następnie utworzyliśmy obraz Dockera i kontener z Jenkinsem. Do usługi Jenkins wchodziliśmy logując się w przeglądarce na adres ip wirtualnej maszyny z dopiskiem odpowiedniego portu. Jenkins w przeglądarce zadziałał poprawnie i mogliśmy utworzyć i skonfigurować konto:



Po zalogowaniu się do Jenkinsa założyliśmy konto, pobraliśmy wtyczki takie jak m.in. Pipeline, github integration, git integration, git plugin, github plu-

gin, blue ocean. Skonfigurowaliśmy konto i zapoznaliśmy się z możliwościami i interfejsem Jenkinsa. Kolejnym krokiem było utworzenie Jenkinsfile w repozytorium z kalkulatorem Python. Plik ten zawiera instrukcje dla przeprowadzanego pipeline Jenkinsa. Jenkinsfile w repozytorium projektu (kluczowy fragment):

```
Code Blame 46 lines (41 loc) · 1.11 KB
1 pipeline {
2   agent any
3
4   stages {
5     stage('Clone') {
6       steps {
7         sh 'rm -rf python-calculator'
8         sh 'git clone https://github.com/Grzywocz-W/python-calculator.git'
9       }
10    }
11
12    stage('Install dependencies') {
13      steps {
14        dir('python-calculator') {
15          sh '''
16            python3 -m venv venv
17            . venv/bin/activate
18            pip install --upgrade pip
19            pip install -r requirements.txt
20          '''
21        }
22      }
23    }
24
25    stage('Run tests') {
26      steps {
27        dir('python-calculator') {
28          sh '''
29            . venv/bin/activate
30            pytest test_main.py --maxfail=1 --disable-warnings --exitfirst
31          '''
32        }
33      }
34    }
35  }
36 }
```

Następnie utworzyliśmy pipeline w Jenkins łącząc go z docelowym repozytorium projektu, tak aby jenkins mógł połączyć się z repozytorium projektu, odczytać instrukcje z Jenkinsfile i dokonać automatycznych testów aplikacji. Pipeline w Jenkins:

Triggers

Set up automated actions that start your build based on specific events, like code changes or scheduled times.

- ☐ Uruchamiaj, gdy inne zadania zostaną zakończone ?
- ☐ Buduj cyklicznie ?
- ☐ GitHub Branches
- ☐ GitHub Pull Requests ?
- ☒ GitHub hook trigger for GITScm polling ?
- ☐ Pobierz z repozytorium kodu ?
- ☐ Wyzwalaj budowanie zdalnie (np. przez skrypt) ?

Pipeline

Define your Pipeline using Groovy directly or pull it from source control.

Definition

Pipeline script from SCM

Pipeline

Define your Pipeline using Groovy directly or pull it from source control.

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

https://github.com/PiotruOlasik/python-calculator.git

Credentials ?

- none -

+ Add

Save Zastosuj

Branches to build ?

Branch Specifier (blank for 'any') ?

*/main

Add Branch

Repository browser ?

(Automatyczny)

Additional Behaviours

Dodaj

Script Path ?

Jenkinsfile

☒ Lightweight checkout ?

Save Zastosuj

Ostatnim krokiem w tym etapie było utworzenie w repozytorium projektu webhooka do adresu URL Jenkinsa aby automatyzacja testów z wykorzystaniem Jenkinsa przebiegała pomyślnie. Webhook:

We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, `x-www-form-urlencoded`, etc). More information can be found in [our developer documentation](#).

Payload URL *

Content type *

Secret

SSL verification

By default, we verify SSL certificates when delivering payloads.

☒ **Enable SSL verification** ☐ **Disable (not recommended)**

Which events would you like to trigger this webhook?

☒ Just the push event.

☐ Send me **everything**.

☐ Let me select individual events.

☒ **Active**

We will deliver event details when this hook is triggered.

Update webhook **Delete webhook**

3.4 Testy







Testy przeszły poprawnie i zarówno Jenkins jak i Github actions działają poprawnie zapewniając automatyczne testowanie po każdym pushu do repozytorium:

Sukces		python-pytest-pipeline	1 dzień 17 godz #1	—	21 sek		
		python-test-python-calculator-main	3 godz 28 min #1	—	14 sek		

All workflows

Showing runs from all workflows

🔍 Filter workflow runs

10 workflow runs		Event ▾	Status ▾	Branch ▾	Actor ▾	
	Update Jenkinsfile			main	 8 hours ago  14s	...
Run Python tests #11: Commit 0e2a852 pushed by PiotruOlasik						
	Merge pull request #2 from Grzywocz-W/main			main	 yesterday  14s	...
Run Python tests #10: Commit 0ccde50 pushed by PiotruOlasik						

4 Podsumowanie i wnioski

- Projekt skutecznie zrealizował założone cele, demonstrując praktyczne zastosowanie narzędzi automatyzacji w procesie wytwarzania oprogramowania
- Wykorzystanie dwóch różnych rozwiązań automatyzacji (GitHub Actions i Jenkins) pozwoliło na porównanie ich możliwości i specyfiki:
 - GitHub Actions oferuje prostszą konfigurację i natychmiastową integrację z repozytorium
 - Jenkins zapewnia większą elastyczność i możliwości rozbudowy poprzez bogaty ekosystem pluginów
- Wdrożenie Jenkinsa na chmurze Azure pokazało praktyczne aspekty utrzymania infrastruktury CI/CD, w tym:
 - Konfigurację środowiska na maszynie wirtualnej
 - Zarządzanie kontenerami Docker
 - Integrację z systemami kontroli wersji
- Projekt wykazał kluczową rolę automatyzacji testów w zapewnianiu jakości kodu poprzez:
 - Natychmiastową weryfikację zmian
 - Systematyczne raportowanie pokrycia testami
 - Redukcję ryzyka wprowadzenia błędów do produkcji
- Wykorzystanie metodyki Agile i narzędzia Jira usprawniło zarządzanie projektem, pozwalając na:
 - Efektywny podział zadań
 - Śledzenie postępów prac
 - Adaptację do pojawiających się zmian
- Projekt potwierdził, że automatyzacja procesów CI/CD jest niezbędnym elementem nowoczesnego wytwarzania oprogramowania, znacząco przyspieszającym cykl rozwoju i poprawiającym jakość końcowego produktu

5 Potencjał rozwoju

- Hostowanie aplikacji webowej kalkulatora na chmurze dostępnej 24h
- Dodanie instalatora wizard dla klientów aplikacji

6 Bibliografia

1. GitHub Documentation – GitHub Actions. <https://docs.github.com/en/actions>
2. Jenkins Official Documentation – Jenkins User Handbook. Dostępne online: <https://www.jenkins.io/doc/>
3. Flask Documentation – Flask Web Development Framework. Dostępne online: <https://flask.palletsprojects.com/>
4. Microsoft Azure Documentation – Virtual Machines. Dostępne online: <https://docs.microsoft.com/en-us/azure/virtual-machines/>