Gitea Actions: runner i pierwszy workflow

Poziom: podstawy **Dotyczy: Gitea 1.27, Gitea Runner 1.0.0** **Czas: ~30 min**

Gitea Actions uruchamia automatyczne kroki (budowanie, testy, wdrożenie) przy każdym zdarzeniu w repozytorium, na przykład przy każdym pushu. Sama Gitea tylko zleca pracę, wykonuje ją osobny program: runner. Tu instalujemy runnera i uruchamiamy pierwszy, najprostszy workflow.

Skróty użyte na tej stronie (4)

Skrót	Co znaczy
CD	Continuous Delivery/Deployment: automatyczne wdrażanie zbudowanego kodu
CI	Continuous Integration: automatyczne budowanie i testowanie kodu po każdej zmianie; w ITIL także configuration item, czyli element konfiguracji: każdy składnik usługi zapisany w CMDB
CLI	Command Line Interface: praca przez wpisywanie komend w terminalu
YAML	YAML Ain't Markup Language: format plików konfiguracyjnych oparty na wcięciach

Fiszka

KLIKAMY W INTERFEJSIE

Włączenie Actions dla repozytorium: ustawienia repozytorium

```
Ustawienia repozytorium → Actions → General → Enable Repository Actions
```

Od Gitea 1.21 mechanizm Actions jest włączony w całej instancji domyślnie, ale dla każdego repozytorium z osobna trzeba go jeszcze zaznaczyć w ustawieniach, zanim workflow z tego repozytorium w ogóle zacznie się uruchamiać.

TYLKO ODCZYT

Wygenerowanie tokenu rejestracji runnera z poziomu CLI: gitea actions generate-runner-token

```
docker exec -u git gitea gitea actions generate-runner-token
```

Token służy do jednorazowej rejestracji runnera, nie do bieżącego uwierzytelniania. Ten sam token da się pobrać bez CLI w interfejsie, patrz niżej.

Przykład:

```
9f1c2b7e4a6d8f0c3b5a7e9d1f3c5b7a9e1d3f5c
```

CLI = Command Line Interface: praca przez wpisywanie komend w terminalu · **docker** = narzędzie do kontenerów

ZMIENIA SYSTEM

Uruchomienie runnera w Dockerze: docker compose up -d

```
docker compose up -d
```

Wykonujemy w `/docker/compose/gitea-runner/`, po uzupełnieniu pliku `.env` adresem instancji i tokenem rejestracji. Runner sam rejestruje się w Gitei przy pierwszym starcie i zapisuje potwierdzenie w pliku `.runner` w katalogu danych.

Przykład:

```
[+] Running 1/1
 ✔ Container gitea-runner  Started
```

docker = narzędzie do kontenerów

KLIKAMY W INTERFEJSIE

Podgląd zarejestrowanych runnerów: Site Administration → Actions → Runners

```
Site Administration → Actions → Runners
```

Ten sam ekran, na poziomie organizacji albo repozytorium zamiast całej instancji, pokazuje runnery widoczne tylko dla tego zakresu.

TYLKO ODCZYT

Wygenerowanie pliku konfiguracyjnego runnera: gitea-runner generate-config

```
docker exec gitea-runner gitea-runner generate-config
```

Wypisuje domyślną konfigurację (limit równoległych zadań, etykiety, ścieżka do cache) do skopiowania i dostosowania w pliku `config.yaml`, zamiast trzymać wszystko w zmiennych środowiskowych.

docker = narzędzie do kontenerów

TYLKO ODCZYT

Sprawdzenie logu runnera: docker compose logs -f

```
docker compose logs -f gitea-runner
```

Pierwsze miejsce diagnozy, gdy workflow nie startuje albo runner nie łączy się z instancją.

Przykład:

```
gitea-runner  | INFO Runner registered successfully
gitea-runner  | INFO Listening for tasks...
```

docker = narzędzie do kontenerów

Z życia wzięte

Z ŻYCIA WZIĘTE

Ola, wtorek, 10:15. Włącza Actions dla repozytorium `cennik`, ale workflow po pierwszym pushu wisi w statusie „Waiting" bez końca. W logu runnera pusto, bo runner jeszcze nie istnieje: samo włączenie Actions w repozytorium nie uruchamia żadnego runnera, trzeba go dopiero postawić i zarejestrować.

Jak to działa

Rozwiń: jak to działa pod spodem

Pojęcie: CI. Skrót od ciągłej integracji: automatyczne budowanie i testowanie kodu przy każdej zmianie w repozytorium, zamiast robienia tego ręcznie od czasu do czasu. Im częściej sprawdzamy, tym szybciej łapiemy błąd, zanim narośnie na niego kolejny kod.

Pojęcie: CD. Skrót od ciągłego dostarczania (albo wdrażania): automatyczne przekazanie zbudowanej i przetestowanej wersji dalej, aż do działającego środowiska, znów bez ręcznego klikania. CI sprawdza, że kod działa, CD dba, żeby ta sprawdzona wersja trafiła tam, gdzie ma działać.

Analogia: linia montażowa. Wyobraźmy sobie fabrykę. Zdarzenie na hali (dostawa nowej partii materiału) uruchamia linię: kolejne stanowiska wykonują po kolei swoje zadania, a na końcu gotowy produkt trafia na magazyn. Gitea Actions działa tak samo, tylko materiałem jest kod, a stanowiskami komputer, który go buduje i sprawdza.

Pojęcie: workflow. Cały przepis na linię montażową dla jednego repozytorium, zapisany w jednym pliku YAML w katalogu `.gitea/workflows/`. Jedno repozytorium może mieć kilka workflow, każdy uruchamiany przez inne zdarzenie albo robiący coś innego.

Pojęcie: zdarzenie (trigger). To, co uruchamia workflow: push do gałęzi, otwarcie pull requesta, utworzenie tagu, a nawet ręczne kliknięcie w interfejsie albo harmonogram czasowy. Bez zdarzenia workflow leży bezczynnie, nawet jeśli jest poprawnie zapisany w repozytorium.

Pojęcie: zadanie (job). Jeden odcinek linii montażowej: zestaw kroków wykonywanych razem, w jednym kontenerze roboczym. Workflow może mieć kilka zadań, domyślnie uruchamianych równolegle, chyba że jawnie każemy jednemu czekać na drugie.

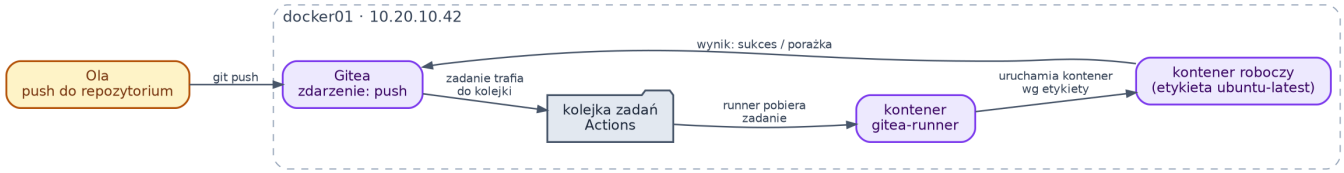
Pojęcie: krok (step). Pojedyncza czynność w zadaniu: uruchomienie komendy, pobranie kodu, zbudowanie obrazu. Kroki w jednym zadaniu wykonują się po kolei, w tym samym kontenerze roboczym, więc plik zapisany w jednym kroku widzi kolejny krok tego samego zadania.

Pojęcie: runner. Program, który stoi i czeka na nowe zadania z Gitei, a gdy jakieś się pojawi, sam pobiera je z kolejki i wykonuje w osobnym, jednorazowym kontenerze roboczym. To runner, nie sama Gitea, robi całą pracę: buduje, testuje, wdraża. Bez uruchomionego i zarejestrowanego runnera workflow czeka w kolejce bezczynnie.

Pojęcie: etykieta runnera. Nazwa (na przykład `ubuntu-latest`), którą zadanie w polu `runs-on` wskazuje, jakiego środowiska potrzebuje. Runner ma zdefiniowany zestaw etykiet, którym odpowiada obraz kontenera do uruchomienia (`ubuntu-latest:docker://node:20-bullseye`): zadanie czeka w kolejce, dopóki żaden dostępny runner nie obsługuje pasującej etykiety.

Pojęcie: sekret. Wartość (hasło, token, klucz) zapisana w ustawieniach repozytorium, dostępna w workflow przez `${{ secrets.NAZWA }}`, ale nigdy niewidoczna w samym pliku ani w logu builda. Odwrotność sekretu to zmienna repozytorium: wartość jawna, przydatna do rzeczy niewrażliwych jak nazwa środowiska.

Droga jednego zdarzenia od pushu do wyniku wygląda tak:



Push trafia do kolejki zadań, runner pobiera zadanie i uruchamia kontener roboczy, wynik wraca do Gitei (kliknięcie otwiera pełny rozmiar)

Nazwa się zmieniła: act_runner to dziś Gitea Runner

> ⚠ Do 5 maja 2026 ten sam program nazywał się `act_runner`, z obrazem `gitea/act_runner` i binarką `act_runner`. Od wersji 1.0.0 projekt przeniósł się do repozytorium `gitea.com/gitea/runner`, binarka nazywa się `gitea-runner`, a

> obraz `gitea/runner`. Instrukcje ze starszych poradników w internecie wciąż używają starych nazw: `act_runner` czytamy jako `gitea-runner`, `GITEA_INSTANCE_URL` i `GITEA_RUNNER_REGISTRATION_TOKEN` zostały bez zmian, więc te dwie zmienne środowiskowe da się przepisać bez modyfikacji.

Instalacja runnera w Dockerze

> □□ W naszym środowisku dane aplikacji trzymamy w `/docker/apps/<nazwa>`, a plik Compose w `/docker/compose/<nazwa>/compose.yaml`. To konwencja, nie wymóg Dockera: można ją zmienić, byle w całej firmie była jedna. Katalog `/docker` to osobny wolumen `vgDocker/lvDocker`, żeby pełny dysk aplikacji nie zatrzymał systemu.

`/docker/compose/gitea-runner/compose.yaml`:

```
services:
  gitea-runner:
    image: gitea/runner:1.0.0
    container_name: gitea-runner
    restart: unless-stopped
    environment:
      GITEA_INSTANCE_URL: ${GITEA_INSTANCE_URL}
      GITEA_RUNNER_REGISTRATION_TOKEN: ${GITEA_RUNNER_REGISTRATION_TOKEN}
      GITEA_RUNNER_NAME: docker01-runner
    volumes:
      - /docker/apps/gitea-runner:/data
      - /var/run/docker.sock:/var/run/docker.sock
```

`/docker/compose/gitea-runner/.env`:

```
GITEA_INSTANCE_URL=https://git.zielona.example
GITEA_RUNNER_REGISTRATION_TOKEN=WstawTokenZRejestracji
```

>

> ⚠ Wolumen `/var/run/docker.sock` daje kontenerowi runnera pełną władzę nad demonem Dockera hosta docker01, a więc w praktyce nad wszystkimi pozostałymi kontenerami na tym hoście i pośrednio nad samym hostem. Każdy workflow, który ten runner wykona, mógłby (celowo albo przez pomyłkę w cudzym kodzie z otwartego repozytorium) zatrzymać, podmienić albo odczytać dane dowolnego innego kontenera na docker01. Runnera z dostępem do gniazda Dockera podłączamy tylko do repozytoriów, którym ufamy, nigdy do publicznie dostępnej instancji z anonimowymi zgłoszeniami.

Rejestracja jest jednorazowa: przy pierwszym starcie kontener zapisuje potwierdzenie w pliku `.runner` w `/docker/apps/gitea-runner`. Kolejne `docker compose up -d` (na przykład po aktualizacji obrazu) nie rejestruje runnera ponownie, dopóki ten plik istnieje.

Instalacja runnera bez Dockera: binarka i usługa systemd

Pobieramy binarkę i rejestrujemy runnera z uprawnieniami zwykłego konta systemowego, nie roota:

```
wget -O /usr/local/bin/gitea-runner https://dl.gitea.com/gitea-runner/1.0.0/gitea-runner-1.0.0-linux-amd64
chmod +x /usr/local/bin/gitea-runner
```

```
sudo -u nazwa_usera gitea-runner register --no-interactive \
  --instance https://git.zielona.example \
  --token WstawTokenZRejestracji \
  --name NazwaRunnera
```

Jednostka systemd:

```
[Unit]
Description=Gitea Runner
After=network.target docker.service

[Service]
User=nazwa_usera
WorkingDirectory=/opt/gitea-runner
```

```
ExecStart=/usr/local/bin/gitea-runner daemon
Restart=always

[Install]
WantedBy=multi-user.target
```

```
sudo systemctl enable --now gitea-runner
```

Ten wariant nadal potrzebuje dostępu do Dockera (grupa `docker` dla konta usługi), jeśli zadania mają uruchamiać kontenery robocze: runner sam w sobie jest tylko programem sterującym, wykonawcą kroków pozostaje kontener.

Pierwszy workflow: „Witaj, Zielona"

`.gitea/workflows/witaj.yaml`:

```
name: Witaj, Zielona
on: [push]

jobs:
  powitanie:
    runs-on: ubuntu-latest
    steps:
      - name: Pobranie kodu
        uses: actions/checkout@v4
      - name: Powitanie
        run: echo "Czesc z Gitea Actions, Zielona!"
```

Zapisujemy plik w repozytorium i wykonujemy zwykły push: samo dodanie pliku w `.gitea/workflows/` jest już zdarzeniem `push`, które go uruchamia.

> □□ Zrzut do uzupełnienia: zakładka „Actions" repozytorium `zielona/cennik` po pierwszym pushu z workflow „Witaj, Zielona", widoczny zielony status ukończonego zadania „powitanie" i rozwinięty log kroku „Powitanie" z linią „Czesc z Gitea Actions, Zielona!".

Sekrety i zmienne repozytorium

```
Ustawienia repozytorium → Actions → Secrets → Add Secret
Ustawienia repozytorium → Actions → Variables → Add Variable
```

Sekret wpisany raz nie da się już podejrzeć w interfejsie, tylko nadpisać nowym, dokładnie jak hasło w większości systemów. W workflow sięgamy po niego przez `${{ secrets.NAZWA_SEKRETU }}`, po zmienną przez `${{ vars.NAZWA_ZMIENNEJ }}`.

Gitea Actions kontra Jenkins kontra TeamCity

	Gitea Actions	Jenkins	TeamCity
Gdzie żyje definicja	plik YAML w repozytorium, `.gitea/workflows/`	osobny serwer, `Jenkinsfile` w repozytorium (Pipeline) albo konfiguracja w UI	osobny serwer, konfiguracja głównie w UI, opcjonalnie zapisana jako kod w Kotlinie
Instalacja	jeden kontener obok Gitei, runner rejestruje się sam	osobny serwer kontrolera plus agenci, więcej ruchomych części	osobny serwer, zwykle cięższy proces wdrożenia niż Jenkins
Kiedy sięgamy	mały zespół, już ma Gitea, potrzebuje prostego CI/CD bez uczenia się nowego narzędzia	duże, złożone pipeline'y, mnóstwo wtyczek, wieloletnia baza zadań do utrzymania	zespół ceniący gotowe raporty i UI bez ręcznego pisania YAML/Groovy
Składnia workflow	zgodna z GitHub Actions, YAML	Groovy (deklaratywny albo skryptowy), patrz Pipeline: Jenkinsfile, etapy, artefakty	głównie konfiguracja przez UI, mniej tekstu do utrzymania w repozytorium
Słaba strona	młodszy projekt, mniejszy ekosystem gotowych akcji niż GitHub Actions	najwięcej ruchomych części do utrzymania spośród trzech	licencja komercyjna powyżej darmowego progu liczby konfiguracji budowania

Częste błędy

Objaw	Przyczyna	Co robimy

Workflow w ogóle się nie uruchamia po pushu	Actions nie włączone dla tego repozytorium	Ustawienia repozytorium → Actions → General → Enable Repository Actions
Zadanie wisi w statusie „Waiting" bez końca	brak zarejestrowanego runnera z pasującą etykietą	sprawdzamy Site Administration → Actions → Runners, stawiamy albo rejestrujemy runnera
Runner rejestruje się przy każdym starcie kontenera od nowa	brak trwałego wolumenu na `/data`, plik `.runner` znika razem z kontenerem	montujemy `/docker/apps/gitea-runner:/data` w compose.yaml
Krok `docker build` w workflow kończy się błędem braku dostępu do Dockera	kontener runnera bez zamontowanego `/var/run/docker.sock`	dokładamy wolumen z gniazdem Dockera, ze świadomością ryzyka opisanego wyżej

□□ Ćwiczenia

> Zadania sprawdzające.

1. Wyjaśniamy różnicę między zdarzeniem, zadaniem i krokiem, na przykładzie workflow „Witaj, Zielona".

Odpowiedź

Zdarzenie to `push` do repozytorium, ono uruchamia cały workflow. Zadanie to `powitanie`, jeden zestaw kroków wykonywany w jednym kontenerze roboczym. Kroki to `Pobranie kodu` i `Powitanie`, wykonywane po kolei wewnątrz tego zadania.

1. Workflow nowego repozytorium wisi w statusie „Waiting" od dziesięciu minut. Wskazujemy dwa najbardziej prawdopodobne powody i jak je sprawdzić.

Odpowiedź

Po pierwsze: Actions może nie być włączone dla tego repozytorium (Ustawienia repozytorium → Actions → General). Po drugie: żaden zarejestrowany runner nie obsługuje etykiety, której żąda pole `runs-on` (sprawdzamy Site Administration → Actions → Runners i porównujemy etykiety).

Powiązane: Gitea: Git z filiżanką herbaty · Instalacja Gitea w Dockerze · Pierwsze repozytorium: od pustego projektu do pierwszego pusha · Docker Compose · Pipeline: Jenkinsfile, etapy, artefakty

Źródła: docs.gitea.com, rozdział „Usage → Actions" · blog.gitea.com, „Release of Runner 1.0.0" · `gitea-runner --help`

Revision #33
Created 2026-09-27 10:32:10 UTC by Tomasz Konieczny
Updated 2026-10-26 14:17:34 UTC by Tomasz Konieczny