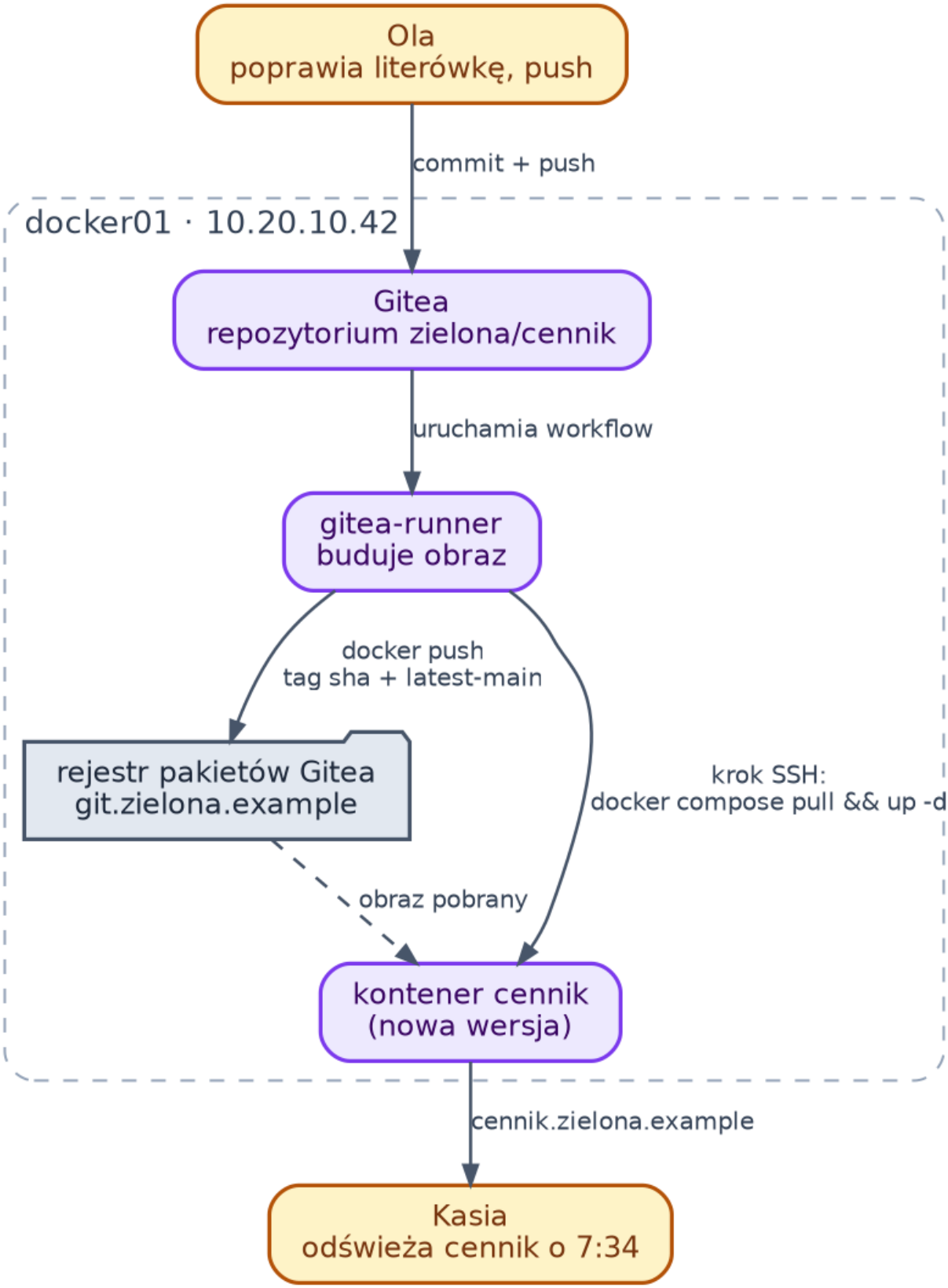
Pierwszy kontener: obraz budowany w Gitea i wdrożony na docker01

Poziom: podstawy **Dotyczy: Gitea 1.27, Gitea Runner 1.0.0, Docker na docker01**

Czas: ~45 min

Doprowadzamy drogę od pushu do działającego kontenera: repozytorium `zielona/cennik` dostaje workflow, który sam buduje obraz, wypycha go do wbudowanego rejestru Gitei i wdraża na serwerze docker01. Efekt: zmiana w kodzie trafia na produkcję bez ręcznego `docker build` i `scp`.

Cała droga od commitu do działającego kontenera, zanim rozbijemy ją na kroki:



Commit trafia do Gitei, workflow buduje obraz i wypycha go do rejestru, krok SSH wdraża go na docker01, Kasia widzi nową wersję (kliknięcie otwiera pełny rozmiar)

Skróty użyte na tej stronie (2)

Skrót	Co znaczy
HTTP	Hypertext Transfer Protocol: protokół stron WWW
SSH	Secure Shell: szyfrowane zdalne logowanie do serwera

Zanim zaczniemy

> **Z ŻYCIA WZIĘTE**
>
> Poniedziałek, 7:30. Kasia, w terenie, otwiera cennik na telefonie przed spotkaniem z klientem i widzi wczorajszą cenę na jeden z produktów, literówkę w opisie. Dzwoni do Oli. Ola poprawia jedną linijkę w repozytorium `zielona/cennik`, robi push z laptopa i nie robi nic więcej: reszta dzieje się sama.

Wymaganie	Uwaga
Runner z tego rozdziału, zarejestrowany i uruchomiony	patrz Gitea Actions: runner i pierwszy workflow
Organizacja `zielona` w Gitei, z repozytorium `cennik`	zespół Oli trzyma tu małą aplikację z cennikiem dla handlowców
Docker na docker01, katalogi `/docker/apps` i `/docker/compose` gotowe	patrz Instalacja Gitea w Dockerze po konwencję ścieżek
Klucz SSH do konta wdrożeniowego na docker01	konto z prawem uruchomić `docker compose` w katalogu `/docker/compose/cennik`, nic więcej

Uważaj: krok wdrożenia w tej procedurze łączy się po SSH z kontenera roboczego runnera na sam host docker01. Prywatny klucz do tego połączenia trzymamy wyłącznie jako sekret repozytorium, nigdy w treści workflow ani w logu.

Przebieg

1. Repozytorium z minimalną aplikacją

Zakładamy repozytorium `zielona/cennik` w organizacji `zielona`, tak jak opisuje strona Pierwsze repozytorium: od pustego projektu do pierwszego pusha. Aplikacja na start to statyczna strona

serwowana przez `nginx:alpine`, żeby cały przykład zmieścił się w dwóch plikach.

`cennik.html`:

```
<!doctype html>
<html lang="pl">
<head><meta charset="utf-8"><title>Cennik Zielona</title></head>
<body>
  <h1>Cennik Zielona</h1>
  <p>Kubek miodu: 24,50 zł</p>
</body>
</html>
```

`Dockerfile`:

```
FROM nginx:1.27-alpine
COPY cennik.html /usr/share/nginx/html/index.html
```

> **Pojęcie: obraz.** Gotowy, złożony z warstw zestaw plików, z którego Docker uruchamia kontener: system plików aplikacji, jej zależności i sposób jej startu, zapisane raz i uruchamiane wielokrotnie, identycznie na każdym hoście.

> **Pojęcie: rejestr.** Magazyn obrazów, z którego Docker je pobiera (`docker pull`) i do którego je wypycha (`docker push`), odpowiednik repozytorium Git, tylko dla gotowych obrazów zamiast kodu źródłowego. Gitea ma wbudowany rejestr pakietów, więc nie potrzebujemy osobnej instalacji Docker Registry czy Harbor obok.

> **Pojęcie: tag.** Etykieta doklejona do nazwy obrazu po dwukropku (`cennik:1.4.2`, `cennik:latest-main`), rozróżniająca kolejne wersje tego samego obrazu w rejestrze. Jeden i ten sam obraz może mieć naraz kilka tagów, tak jak w tej procedurze: stały tag z numeru commitu i zmienny `latest-main` wskazujący zawsze na najnowszą wersję z gałęzi `main`.

2. Token do rejestru pakietów i docker login

Token dostępu osobistego z uprawnieniem do pakietów zakładamy w profilu konta, którym będzie logował się workflow: `Ustawienia → Aplikacje → Wygeneruj nowy token`, z zaznaczonym zakresem

`write:package`.

```
docker login git.zielona.example -u nazwa_usera
```

Przykład:

```
Password:
Login Succeeded
```

W workflow logowanie dzieje się automatycznie, tokenem z sekretu repozytorium, nie ręcznie z linii poleceń: powyższe polecenie służy tylko do sprawdzenia z własnej stacji, że token w ogóle działa, zanim wpiszemy go do sekretu.

3. Sekret z tokenem rejestru

```
Ustawienia repozytorium → Actions → Secrets → Add Secret
Nazwa: REJESTR_TOKEN
```

4. Workflow budujący obraz i wypychający go do rejestru

`.gitea/workflows/build-i-publikacja.yaml`:

```
name: Budowa i publikacja obrazu
on:
  push:
    branches: [main]

jobs:
  buduj:
    runs-on: ubuntu-latest
    steps:
      - name: Pobranie kodu
        uses: actions/checkout@v4

      - name: Logowanie do rejestru Gitea
        run: echo "${{ secrets.REJESTR_TOKEN }}" | docker login git.zielona.example -u
ola.wisniewska --password-stdin
```

```
      - name: Budowa obrazu
        run: |
          docker build \
            -t git.zielona.example/zielona/cennik:${{ gitea.sha }} \
            -t git.zielona.example/zielona/cennik:latest-main \
            .

      - name: Wypchnięcie obrazu do rejestru
        run: |
          docker push git.zielona.example/zielona/cennik:${{ gitea.sha }}
          docker push git.zielona.example/zielona/cennik:latest-main
```

`${{ gitea.sha }}` to skrócony identyfikator commitu, który wywołał workflow: dzięki niemu każdy zbudowany obraz ma jeden niepowtarzalny, niezmienny tag, niezależnie od tego, że `latest-main` przesuwa się z każdym pushem na nowy obraz.

5. Compose na docker01 korzystający z obrazu z rejestru

> 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/cennik/compose.yaml`:

```
services:
  cennik:
    image: git.zielona.example/zielona/cennik:${TAG_OBRAZU}
    container_name: cennik
    restart: unless-stopped
    ports:
      - "8090:80"
```

`/docker/compose/cennik/.env`:

```
TAG_OBRAZU=latest-main
```

Zmienna `TAG_OBRAZU` w `.env` wskazuje tag, po który sięga `docker compose pull`: na co dzień `latest-main`, przy wycofaniu zmian w kroku 8 podmieniamy ją na konkretny, niezmienny tag z numeru commitu.

6. Krok wdrożenia: SSH do docker01

```
      - name: Wdrożenie na docker01
        run: |
          mkdir -p ~/.ssh
          echo "${{ secrets.KLUCZ_WDROZENIOWY }}" > ~/.ssh/id_ed25519
          chmod 600 ~/.ssh/id_ed25519
          ssh-keyscan -H 10.20.10.42 >> ~/.ssh/known_hosts
          ssh -i ~/.ssh/id_ed25519 nazwa_usera@10.20.10.42 \
            'cd /docker/compose/cennik && docker compose pull && docker compose up -d'
```

Uważaj: ten krok dokłada do repozytorium prywatny klucz SSH jako sekret `KLUCZ_WDROZENIOWY`, ważny na koncie z prawem uruchomić dokładnie jedną komendę w jednym katalogu na docker01, nic ponadto. Kompromitacja tego sekretu (na przykład przez `echo` kroku wypisany w logu przez pomyłkę) daje napastnikowi tyle, na ile pozwala to konto, nie całego hosta.

> Alternatywa bez klucza SSH: skoro runner i tak działa jako kontener na docker01, z zamontowanym gniazdem Dockera (patrz Gitea Actions: runner i pierwszy workflow), krok wdrożenia mógłby wywołać `docker compose` bezpośrednio przez to gniazdo, bez wychodzenia przez sieć. Unikamy wtedy trzymania klucza SSH jako osobnego sekretu, kosztem tego, że każdy workflow tego runnera dostaje tę samą, pełną władzę nad hostem, o której ostrzegaliśmy przy montowaniu gniazda. To nie jest wybór bezpieczniejszy, tylko inny rozkład tego samego ryzyka: SSH ogranicza je do jednego konta i jednego katalogu, gniazdo Dockera nie ogranicza niczego.

Weryfikacja

```
curl -s -o /dev/null -w "%{http_code}\n" https://cennik.zielona.example
```

Przykład:

```
200
```

```
ssh nazwa_usera@10.20.10.42 'docker compose -f /docker/compose/cennik/compose.yaml images'
```

Przykład:

```
CONTAINER   REPOSITORY                              TAG           IMAGE ID       SIZE
cennik      git.zielona.example/zielona/cennik   latest-main   4f6a2b1c9e3d   23.4MB
```

Otwieramy `https://cennik.zielona.example` w przeglądarce i sprawdzamy, czy poprawiona treść jest widoczna: to jest ostateczne potwierdzenie, kod odpowiedzi HTTP mówi tylko, że serwer w ogóle odpowiada.

Wycofanie zmian

Jeśli nowa wersja psuje coś na produkcji, wracamy do poprzedniego, znanego dobrego obrazu przez podmianę tagu w `.env`, nie przez cofanie commitów w repozytorium:

```
ssh nazwa_usera@10.20.10.42
```

```
sed -i 's/TAG_OBRAZU=.*/TAG_OBRAZU=PoprzedniTagZCommita/' /docker/compose/cennik/.env
docker compose -f /docker/compose/cennik/compose.yaml pull
docker compose -f /docker/compose/cennik/compose.yaml up -d
```

Poprzedni tag odczytujemy z listy pakietów organizacji `zielona` w Gitei (`Organizacja → Pakiety`) albo z historii workflow w zakładce Actions, gdzie każdy udany bieg pokazuje, jaki `gitea.sha` wtedy wypchnął. Tag `latest-main` nie nadaje się do wycofania, bo zawsze wskazuje na najnowszy obraz, nigdy na konkretną, wcześniejszą wersję.

Gdy coś nie zadziała

Objaw	Przyczyna	Co robimy
Krok `docker login` w workflow kończy się błędem `unauthorized`	token w sekrecie bez uprawnienia `write:package`	generujemy nowy token z zaznaczonym zakresem pakietów, aktualizujemy sekret
`docker build` działa, `docker push` odrzucony	nazwa obrazu nie zaczyna się od nazwy organizacji zgodnej z repozytorium (`git.zielona.example/zielona/...`)	poprawiamy nazwę obrazu w kroku budowy tak, żeby pasowała do organizacji `zielona`
Krok SSH kończy się `Permission denied (publickey)`	klucz w sekrecie niekompletny (brakujący nowy wiersz na końcu) albo klucz publiczny nie dodany na docker01	porównujemy klucz publiczny w `~/.ssh/authorized_keys` konta wdrożeniowego z kluczem prywatnym w sekrecie

Objaw	Przyczyna	Co robimy
Kontener `cennik` wstaje ze starą treścią mimo udanego pusha	`docker compose pull` pobrał obraz, ale `up -d` nie wymusił ponownego utworzenia kontenera, bo Compose uznał konfigurację za niezmienioną	`docker compose up -d --force-recreate` na docker01, a potem sprawdzenie tagu w `.env`

Powiązane: Gitea Actions: runner i pierwszy workflow · Instalacja Gitea w Dockerze · Pierwsze repozytorium: od pustego projektu do pierwszego pusha · Docker Compose

Źródła: docs.gitea.com, rozdział „Packages → Container" · `docker help build` · `docker help push`

Revision #33
Created 2026-09-27 10:32:12 UTC by Tomasz Konieczny
Updated 2026-10-26 14:17:35 UTC by Tomasz Konieczny