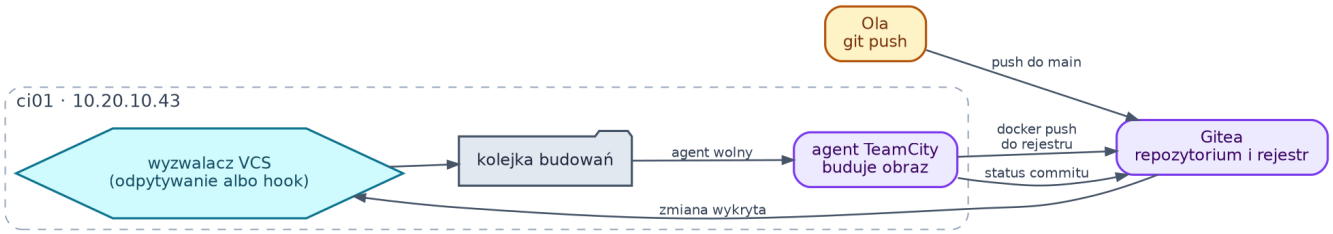
Pierwszy projekt: cennik z Gitea budowany w TeamCity

Typ: procedura **Dotyczy: TeamCity 2026.2, Gitea 1.27, repozytorium zielona/cennik**

Czas: ~45 min

Doprowadzamy tę samą drogę, co w książce o Gitea, tylko silnikiem budowania jest teraz TeamCity: push do repozytorium `zielona/cennik` uruchamia build, który sam buduje obraz, wypycha go do rejestru Gitei i zostawia w Gitea widoczny status commitu.

Cała droga od pushu do statusu w Gitea, zanim rozbijemy ją na kroki:



Push w Gitea uruchamia wyzwalacz VCS, kolejkę, agenta budującego obraz i status commitu z powrotem w Gitea (kliknięcie otwiera pełny rozmiar)

Skróty użyte na tej stronie (5)

Skrót	Co znaczy
API	Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą
DSL	Domain Specific Language: język programowania zaprojektowany wąsko, do jednego zadania, tu do zapisania konfiguracji TeamCity jako kod w Kotlinie
HTTPS	HTTP zabezpieczony szyfrowaniem TLS
REST	sposób budowania API przez zwykłe żądania HTTP (GET, POST) zamiast osobnego protokołu
VCS	Version Control System: system kontroli wersji, program taki jak Git, śledzący zmiany w plikach i ich historię

Zanim zaczniemy

Wymaganie	Uwaga
Serwer TeamCity z autoryzowanym agentem	patrz Instalacja TeamCity w Dockerze i Agenci TeamCity: dodanie, autoryzacja, pule
Repozytorium `zielona/cennik` w organizacji `zielona`	to samo repozytorium co w książce o Gitea
Token dostępu osobistego w Gitea, zakres `read:repository` i `write:package`	Ustawienia → Aplikacje → Wygeneruj nowy token, w profilu konta Oli
Agent z zamontowanym gniazdem Dockera	patrz uwaga o ryzyku w Instalacja TeamCity w Dockerze

Uważaj: token z zakresem `write:package` trzymamy wyłącznie jako parametr typu hasło w konfiguracji TeamCity, nigdy jawnie w krokach budowania ani w logu. Konfiguracja parametrów zamaskowanych w logu działa w TeamCity tak samo jak `credentials()` w Jenkinsfile, opisane na stronie Pipeline: Jenkinsfile, etapy, artefakty.

Przebieg

1. Utworzenie projektu i korzenia VCS

Zakładamy nowy projekt wskazujący repozytorium Gitea:

```
Administracja → Projects → Create project → From URL
```

> **Pojęcie: korzeń VCS (VCS root).** Wskazanie w TeamCity, gdzie leży repozytorium i jak się do niego dostać: adres, gałąź domyślna i sposób uwierzytelnienia. Jeden korzeń VCS może być współdzielony przez wiele konfiguracji budowania w tym samym projekcie, zamiast powtarzać te same dane w każdej z nich.

Pole korzenia VCS	Wartość w Zielonej	Uwaga
Type of VCS	Git	jedyny typ, którego TeamCity potrzebuje dla Gitea, bez osobnej wtyczki
Fetch URL	`https://git.zielona.example/zielona/cennik.git`	wariant HTTPS z tokenem, wariant SSH niżej
Default branch	`refs/heads/main`	zgodnie z gałęzią chronioną z książki o Gitea

Pole korzenia VCS	Wartość w Zielonej	Uwaga
Authentication method	Password	w polu hasła wklejamy token dostępu osobistego, nie hasło konta

Wariant SSH, gdy wolimy uniknąć tokenu HTTPS: `Fetch URL` w postaci `ssh://git@git.zielona.example:2222/zielona/cennik.git`, `Authentication method` ustawione na `Uploaded key`, z prywatnym kluczem wgranym w `Administracja → SSH Keys`, tym samym, którego publiczny odpowiednik dodaliśmy już w Gitea przy pierwszym repozytorium.

2. Konfiguracja budowania: krok budowy i publikacji obrazu

Zakładamy konfigurację budowania wewnątrz projektu, z dwoma krokami typu `Docker`:

Krok	Runner	Co robi
Budowa obrazu	Docker: Build Image	buduje obraz z `Dockerfile` w korzeniu repozytorium, tag `git.zielona.example/zielona/cennik:%build.vcs.number%`
Publikacja obrazu	Docker: Push Image	wypycha zbudowany obraz do wbudowanego rejestru pakietów Gitei

Rejestr Gitei wymaga zalogowania przed publikacją: zakładamy parametr konfiguracji `docker.registry.password`, typu `Password`, z wartością tokenu `write:package`, i wskazujemy go w ustawieniach połączenia z rejestrem Dockera zamiast wpisywać token wprost.

Uważaj: `%build.vcs.number%` to zmienna TeamCity ze skróconym identyfikatorem commitu, który wywołał build, odpowiednik `${{ gitea.sha }}` z workflow Gitea Actions: dzięki niej każdy zbudowany obraz ma osobny, niezmienny tag, niezależny od tagu `latest-main`.

3. Wyzwalacz: build po każdym pushu

```
Konfiguracja budowania → Triggers → Add new trigger → VCS Trigger
```

Domyślny wyzwalacz VCS odpytuje repozytorium co kilkanaście minut i uruchamia build, gdy zauważy nowy commit na obserwowanej gałęzi. Odpytywanie samo w sobie działa od razu, bez dodatkowej konfiguracji po stronie Gitea, kosztem opóźnienia rzędu kilkunastu minut między pushem a startem builda.

> Przyspieszenie bez wtyczki dedykowanej Gitei: hook w Gitea (`Ustawienia repozytorium → Webhooks → Add Webhook → Gitea`) może wywołać ten sam efekt co push, wołając REST API TeamCity komendą `curl` zaraz po pushu, zamiast czekać na najbliższe odpytywanie. TeamCity nie ma osobnej wtyczki webhooków dla Gitei, tak jak ma dla GitHuba, ale generyczne powiadomienie REST działa niezależnie od typu serwera Git po drugiej stronie.

4. Artefakty

```
Konfiguracja budowania → General Settings → Artifact paths
```

```
Dockerfile => artefakty/
cennik.html => artefakty/
```

> **Pojęcie: artefakt.** Plik zachowany po zakończonym buildzie, dostępny do pobrania z historii, nawet po wyczyszczeniu workspace agenta, odpowiednik `archiveArtifacts` z Jenkinsfile. Dla samego obrazu Dockera artefaktem w praktyce jest wpis w rejestrze Gitei, nie plik w TeamCity: obraz zajmowałby niepotrzebnie miejsce zdublowane w dwóch miejscach naraz.

5. Status commitu z powrotem w Gitea

```
Konfiguracja budowania → Build Features → Add build feature → Commit status publisher
```

Pole	Wartość w Zielonej	Uwaga
VCS root	korzeń VCS z kroku 1	ten sam, co dla checkoutu kodu
Publisher	GitHub	interfejs Gitei odpowiada wystarczająco na żądania w stylu GitHuba, żeby ten publisher zadziałał wprost
GitHub URL	`https://git.zielona.example/api/v1`	adres API Gitei zamiast publicznego GitHuba
Access Token	token z zakresu `read:repository` i `write:repository`	osobny od tokenu do rejestru pakietów

Od tego momentu każdy build zostawia w Gitea, przy odpowiednim commicie, zielony albo czerwony znacznik statusu, dokładnie tak jak zrobiłby to workflow Gitea Actions.

◫ ZRZUT EKRANU DO UZUPEŁNIENIA

Zrzut do uzupełnienia: widok commitu w Gitea, zakładka repozytorium `zielona/cennik`, z widocznym zielonym znacznikiem statusu „ci/teamcity" obok wiadomości commitu.

6. Kotlin DSL: konfiguracja jako kod

> **Pojęcie: Kotlin DSL.** Sposób zapisania całej konfiguracji projektu, korzenia VCS, konfiguracji budowania, wyzwalaczy, jako plików Kotlin w repozytorium, zamiast wyłącznie w bazie danych serwera. TeamCity potrafi wygenerować taki zapis automatycznie z konfiguracji już złożonej w interfejsie: `Konfiguracja projektu → Versioned Settings → Generate Kotlin DSL`.

Włączenie **ustawień wersjonowanych** (Versioned Settings) wskazuje repozytorium, w którym TeamCity będzie odtąd trzymał ten wygenerowany kod, najczęściej osobny katalog `.teamcity/` w tym samym repozytorium `zielona/cennik` albo osobne repozytorium ustawień, współdzielone przez wiele projektów. Od tego momentu zmiana w interfejsie i zmiana w pliku Kotlin robią dokładnie to samo, a historia zmian konfiguracji przechodzi przez ten sam przegląd kodu i pull request, co reszta repozytorium, tak jak Jenkinsfile w Jenkinsie.

> Zespół Oli włącza ustawienia wersjonowane dopiero po tym, jak pierwsza konfiguracja zaczyna działać stabilnie klikana w interfejsie: przepisanie działającej konfiguracji na kod, gdy już wiadomo, że działa, jest bezpieczniejsze niż pisanie Kotlina od zera bez wcześniejszego sprawdzenia w interfejsie, czy build w ogóle przechodzi.

Weryfikacja

```
git -C cennik commit --allow-empty -m "Test wyzwalacza TeamCity"
git -C cennik push origin main
```

Sprawdzamy w interfejsie, że build wystartował:

```
Projekty → cennik → Build Configuration → historia buildów
```

Sprawdzamy obraz w rejestrze pakietów Gitei:

```
Organizacja zielona → Pakiety
```

Przykład:

```
cennik  git.zielona.example/zielona/cennik  tag: a1b2c3d
```

Wycofanie zmian

Usunięcie konfiguracji budowania bez usuwania repozytorium ani rejestru:

```
Konfiguracja budowania → General Settings → Remove build configuration
```

Odłączenie samego wyzwalacza, gdy chcemy tymczasowo zatrzymać automatyczne buildy bez usuwania reszty konfiguracji:

```
Konfiguracja budowania → Triggers → usuwamy VCS Trigger
```

Gdy coś nie zadziała

Objaw	Przyczyna	Co robimy
Build w ogóle się nie uruchamia po pushu	wyzwalacz VCS jeszcze nie odpytał repozytorium (domyślny odstęp)	czekamy na najbliższe odpytywanie albo dodajemy przyspieszenie z kroku 3
Krok `Docker: Push Image` kończy się błędem `unauthorized`	parametr z tokenem rejestru pusty albo token bez zakresu `write:package`	generujemy nowy token z zaznaczonym zakresem pakietów, aktualizujemy parametr konfiguracji
Status commitu w Gitea się nie pojawia mimo udanego builda	zły adres API w polu „GitHub URL" albo token bez zakresu `repository`	sprawdzamy dokładny adres `https://git.zielona.example/api/v1` i zakres tokenu w Commit Status Publisher
Build kończy się bez błędu, ale obraz w rejestrze ma starą treść	tag `%build.vcs.number%` wskazuje na wcześniejszy, zbuforowany commit	sprawdzamy, czy korzeń VCS w ogóle pobrał najnowszy commit przed krokiem budowy obrazu

Powiązane: Pierwszy kontener: obraz budowany w Gitea i wdrożony na docker01 · Agenci TeamCity: dodanie, autoryzacja, pule · Pipeline: Jenkinsfile, etapy, artefakty

Źródła: jetbrains.com/help/teamcity, rozdział „Commit Status Publisher" · jetbrains.com/help/teamcity, rozdział „Kotlin DSL" · blog.bitbebop.com, „Gitea Status Checks with TeamCity" · jetbrains.com/help/teamcity, rozdział „Configuring VCS Post-Commit Hooks for TeamCity"

Revision #31
Created 2026-09-27 11:19:37 UTC by Tomasz Konieczny
Updated 2026-10-26 14:17:49 UTC by Tomasz Konieczny