Utrzymanie

Kopia zapasowa, aktualizacja i częste problemy.

- Kopia zapasowa i aktualizacja TeamCity
- TeamCity: częste problemy

Kopia zapasowa i aktualizacja TeamCity

Poziom: operacyjny **Dotyczy: TeamCity 2026.2 w Dockerze, PostgreSQL 16 na pg01**

Czas: ~35 min

W przeciwieństwie do Gitei, gdzie kopia repozytoriów i kopia bazy to dwa osobne polecenia, TeamCity robi obie rzeczy naraz jednym narzędziem: `maintainDB`, łączące się z bazą, wewnętrzną albo zewnętrzną, tym samym mechanizmem co sam serwer. Aktualizacja to podmiana tagu obrazu, a migracja schematu bazy dzieje się sama przy pierwszym starcie nowej wersji.

Skróty użyte na tej stronie (3)

Skrót	Co znaczy
API	Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą
JDBC	Java Database Connectivity: standardowy sposób, w jaki program w Javie łączy się z bazą danych
REST	sposób budowania API przez zwykłe żądania HTTP (GET, POST) zamiast osobnego protokołu

Zanim zaczniemy

Wymaganie	Uwaga
Dostęp do `docker exec` na kontenerze `teamcity-server` na `ci01`	konto z prawem uruchamiać kontenery Dockera
Miejsce na `bkp01` na regularne kopie	patrz Świat przykładów: Zielona po adres i rolę serwera
Znajomość obecnego tagu obrazu	`docker compose images` w `/docker/compose/teamcity/`
Chwila przestoju na kopię pełną (opcjonalnie)	`maintainDB` daje najbardziej spójny wynik przy zatrzymanym serwerze

Uważaj: aktualizacja TeamCity migruje schemat bazy danych przy pierwszym starcie nowej wersji, tak samo jak Gitea. Migracja jest jednokierunkowa: powrót do starego tagu obrazu po tym, jak migracja już przeszła, zostawia stary TeamCity wobec nowego schematu bazy, którego nie

rozumie. Jedyna pewna droga wycofania po migracji to odtworzenie z kopii sprzed aktualizacji.

Przebieg

1. Kopia regularna z interfejsu, serwer działa

```
Administracja → Backup → Include in backup: configs, database, build history
Administracja → Backup → Start Backup
```

> **Pojęcie: kopia z interfejsu.** Metoda zalecana do regularnych, cyklicznych kopii: działa, gdy serwer obsługuje ruch, i można ją wywołać też przez REST API z poziomu skryptu harmonogramu, patrz krok 4. Nie obejmuje kolejki buildów w trakcie wykonywania ani samych artefaktów, ze względu na ich rozmiar.

2. Kopia pełna z CLI: maintainDB backup

```
docker exec -u teamcity teamcity-server /opt/teamcity/bin/maintainDB.sh backup -C -D -L -U -F kopia-teamcity -M
```

Przykład:

```
Backup file created: /data/teamcity_server/datadir/backup/kopia-teamcity_20260927_030001.zip
```

Flaga	Co dołącza
`-C`	konfiguracje projektów i konfiguracji budowania
`-D`	dane bazy, niezależnie od tego, czy to wbudowana HSQLDB, czy zewnętrzny PostgreSQL na `pg01`
`-L`	logi buildów
`-U`	dane wtyczek i historię wyzwalaczy
`-F`	nazwa pliku wynikowego
`-M`	dopisanie znacznika czasu do nazwy pliku

> **Pojęcie: maintainDB.** Narzędzie z linii poleceń, łączące się z bazą tym samym połączeniem JDBC, którego używa sam serwer: w przeciwieństwie do `gitea dump`, opisanego w książce o Gitea, obejmuje całą bazę wprost, bez potrzeby osobnego `pg_dump` na `pg01`. Pełna spójność wymaga zatrzymanego serwera na czas kopii, ale narzędzie działa też przy uruchomionym serwerze, kosztem mniejszej pewności co do stanu danych w trakcie zapisu.

3. Skopiowanie zrzutu z kontenera na bkp01

```
docker cp teamcity-server:/data/teamcity_server/datadir/backup/kopia-
teamcity_20260927_030001.zip /docker/apps/teamcity/kopie/
```

```
scp /docker/apps/teamcity/kopie/kopia-teamcity_20260927_030001.zip
nazwa_usera@bkp01:/backup/teamcity/
```

Nazwę pliku wklejamy dokładnie taką, jaką pokazało polecenie z kroku 2: znacznik czasu w nazwie jest unikalny dla każdego uruchomienia.

4. Harmonogram: cron albo timer systemd

Wybieramy jeden z dwóch mechanizmów opisanych osobno, cron: zadania cykliczne albo systemd timery: OnCalendar zamiast crona, i spinamy nim kroki 2 i 3 w jeden skrypt uruchamiany co noc.

```
crontab -e
```

Przykład:

```
0 3 * * * /docker/apps/teamcity/skrypty/kopia-nocna.sh >> /var/log/teamcity-kopia.log 2>&1
```

5. Miejsce przechowywania i test odtworzenia

Kopie trafiają na `bkp01`, do katalogu `/backup/teamcity/`, tym samym mechanizmem co reszta serwerów opisanych w Świat przykładów: Zielona. Raz na kwartał odtwarzamy najnowszy zrzut na osobnej, testowej instalacji tej samej wersji: przywrócenie przez `maintainDB.sh restore`, wskazanie

testowej bazy, pierwszy start. Kopia, której nikt nigdy nie spróbował odtworzyć, jest tylko przypuszczeniem, że działa.

Aktualizacja

1. Przeczytanie informacji o wydaniu

Przed zmianą tagu sprawdzamy release notes na `jetbrains.com/help/teamcity` pod kątem zmian wymagających ręcznej ingerencji: zmian w formacie parametrów, wycofanych wtyczek, ostrzeżeń o migracji, która potrwa długo na dużej instalacji.

2. Wykonanie kopii sprzed aktualizacji

Powtarzamy kroki 2 i 3 z sekcji wyżej, poza harmonogramem, bezpośrednio przed aktualizacją: to jest kopia, do której wracamy w razie problemu, świeższa niż ostatnia nocna.

3. Zmiana tagu obrazu

`/docker/compose/teamcity/.env` zostaje bez zmian, zmieniamy tag wprost w `compose.yaml`:

```
image: jetbrains/teamcity-server:2026.3
```

```
docker compose -f /docker/compose/teamcity/compose.yaml pull
docker compose -f /docker/compose/teamcity/compose.yaml up -d
```

Przykład:

```
[+] Running 2/2
 ✔ Container teamcity-server     Started
 ✔ Container teamcity-agent-1    Started
```

Migracja schematu bazy wykonuje się sama przy starcie nowego obrazu: nie ma osobnej komendy migracji do wywołania ręcznie.

4. Aktualizacja agentów

```
image: jetbrains/teamcity-agent:2026.3
```

```
docker compose -f /docker/compose/teamcity/compose.yaml pull
docker compose -f /docker/compose/teamcity/compose.yaml up -d
```

Agent starszej wersji potrafi łączyć się z nowszym serwerem przez krótki czas po aktualizacji, ale zostawiamy oba w tej samej wersji na stałe, żeby uniknąć niespodzianek przy krokach budowania niedostępnych w starszym agencie.

5. Weryfikacja po aktualizacji

```
docker compose -f /docker/compose/teamcity/compose.yaml logs server | tail -n 30
```

Sprawdzamy w logu potwierdzenie zakończonej migracji i brak błędów przy starcie, a w stopce strony logowania w przeglądarce nowy numer wersji.

◫ ZRZUT EKRANU DO UZUPEŁNIENIA

Zrzut do uzupełnienia: stopka strony logowania TeamCity po aktualizacji, z widocznym numerem wersji zgodnym z nowym tagiem obrazu (na przykład „TeamCity 2026.3").

Weryfikacja

```
docker compose -f /docker/compose/teamcity/compose.yaml ps
```

Przykład:

```
NAME                IMAGE                                STATUS
teamcity-server     jetbrains/teamcity-server:2026.3    Up 5 minutes (healthy)
```

```
curl -s -o /dev/null -w "%{http_code}\n" http://ci01:8111
```

Przykład:

```
200
```

Wycofanie zmian

Przed migracją (kopia z kroku 2 aktualizacji jeszcze nie była potrzebna): przerywamy, wracamy do poprzedniego tagu w `compose.yaml`, `docker compose up -d` uruchamia starą wersję na niezmienionej jeszcze bazie.

Po migracji, gdy nowa wersja działa niepoprawnie: samo cofnięcie tagu obrazu nie wystarcza, bo baza ma już nowy schemat, którego stara wersja nie rozumie. Odtwarzamy zamiast tego kopię z kroku 2 aktualizacji przez `maintainDB.sh restore`, dopiero potem uruchamiamy serwer ze starym tagiem obrazu.

Gdy coś nie zadziała

Objaw	Przyczyna	Co robimy
`maintainDB.sh backup` kończy się błędem braku miejsca	katalog `backup` w katalogu danych na tym samym, zapchanym wolumenie co reszta aplikacji	sprawdzamy wolne miejsce na `/docker/apps/teamcity`, zwalniamy albo przenosimy stare kopie na `bkp01`
Kopia z kroku 2 trwa bardzo długo przy działającym serwerze	duża baza i równolegle trwające buildy zwiększające obciążenie	uruchamiamy kopię w oknie serwisowym, przy zatrzymanym serwerze, dla pełnej spójności
Po aktualizacji kontener `teamcity-server` bez końca się restartuje	migracja przerwana w połowie (na przykład restart hosta w trakcie)	odtwarzamy z kopii sprzed aktualizacji, nie próbujemy uruchamiać ponownie tego samego obrazu na uszkodzonej w połowie bazie
Agent po aktualizacji obrazu nie łączy się z serwerem	zbyt duża różnica wersji między serwerem a starym agentem	aktualizujemy agenta do tej samej wersji co serwer, zgodnie z krokiem 4

Powiązane: Instalacja TeamCity w Dockerze · Kopia zapasowa i aktualizacja Gitea · cron: zadania cykliczne · systemd timery: OnCalendar zamiast crona · TeamCity: częste problemy

Źródła: jetbrains.com/help/teamcity, rozdział „TeamCity Data Backup" · jetbrains.com/help/teamcity, rozdział „Creating Backup via maintainDB command-line tool" · jetbrains.com/help/teamcity, rozdział „Restoring TeamCity Data from Backup"

TeamCity: częste problemy

Poziom: operacyjny **Dotyczy: TeamCity 2026.2, Gitea 1.27, PostgreSQL 16 na pg01**

Czas: ~20 min

Zestawienie objawów, na które zespół IT trafia najczęściej przy pracy z TeamCity: logowanie, licencja, kolejka buildów i połączenie z Gitea. Tabela niżej daje szybką odpowiedź, rozwinięcie pod nią tłumaczy przyczynę do końca.

Skróty użyte na tej stronie (3)

Skrót	Co znaczy
IT	Information Technology: dział albo praca związana z komputerami i infrastrukturą firmy
LDAP	Lightweight Directory Access Protocol: protokół do odpytywania katalogu użytkowników (AD, FreeIPA)
VCS	Version Control System: system kontroli wersji, program taki jak Git, śledzący zmiany w plikach i ich historię

Fiszka

TYLKO ODCZYT

Sprawdzenie logu kontenera serwera: docker compose logs

```
docker compose -f /docker/compose/teamcity/compose.yaml logs --tail 100 server
```

Pierwsze miejsce diagnozy niemal każdego z objawów niżej.

docker = narzędzie do kontenerów

TYLKO ODCZYT

Sprawdzenie stanu agentów: curl REST API

```
curl -H "Authorization: Bearer TOKEN_DOSTEPU"
https://ci.zielona.example/app/rest/agents?locator=authorized:any
```

Pokazuje wszystkich znanych serwerowi agentów naraz, autoryzowanych i nie, razem z ich stanem połączenia.

REST = sposób budowania API przez zwykłe żądania HTTP (GET, POST) zamiast osobnego protokołu · **API** = Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą · **curl** = client URL: pobieranie i wysyłanie przez HTTP z terminala

KLIKAMY W INTERFEJSIE

Sprawdzenie kolejki buildów: Administracja → Build Queue

```
Administracja → Build Queue
```

Kolejka pokazuje każdy build czekający na wolnego i zgodnego agenta, razem z powodem, dla którego jeszcze nie wystartował.

Częste błędy

Objaw	Przyczyna	Co robimy
Nowy build w ogóle nie startuje, komunikat o limicie licencji	przekroczony darmowy limit stu konfiguracji budowania albo trzech agentów	archiwizujemy nieużywane konfiguracje albo odautoryzowujemy zbędnego agenta, patrz Plan wdrożenia TeamCity w Zielonej
Build czeka w kolejce mimo widocznych, wolnych agentów	żaden z wolnych agentów nie spełnia wymagań konfiguracji (agent requirements)	zakładka „Compatible Agents" konfiguracji budowania, porównujemy z parametrami agenta
Logowanie kontem domenowym odrzucone mimo istniejącego konta w AD	filtr logowania LDAP bez konta w wymaganej grupie, albo konto poza filtrem	sprawdzamy `teamcity.users.login.filter` i członkostwo w `TeamCity-Uzytkownicy`, patrz Logowanie kontem z Active Directory
Krok `Docker: Push Image` kończy się błędem `unauthorized`	token do rejestru Gitei bez zakresu `write:package` albo wygasły	generujemy nowy token, aktualizujemy parametr konfiguracji
Status commitu nie pojawia się w Gitea mimo udanego builda	zły adres API w Commit Status Publisher albo token bez zakresu `repository`	sprawdzamy adres `https://git.zielona.example/api/v1` i zakres tokenu

Objaw	Przyczyna	Co robimy
Build nie startuje po pushu, mimo poprawnego korzenia VCS	wyzwalacz VCS jeszcze nie odpytał repozytorium, brak przyspieszenia przez hook	czekamy na odpytywanie albo dodajemy powiadomienie REST z webhooka Gitei
Po aktualizacji serwer bez końca się restartuje	migracja bazy przerwana w połowie	odtwarzamy z kopii sprzed aktualizacji, patrz Kopia zapasowa i aktualizacja TeamCity
Nowy agent nie pojawia się na liście `Unauthorized`	zły adres `SERVER_URL` albo agent bez łączności sieciowej do serwera	sprawdzamy log agenta, adres musi być osiągalny z jego maszyny, nie tylko z przeglądarki

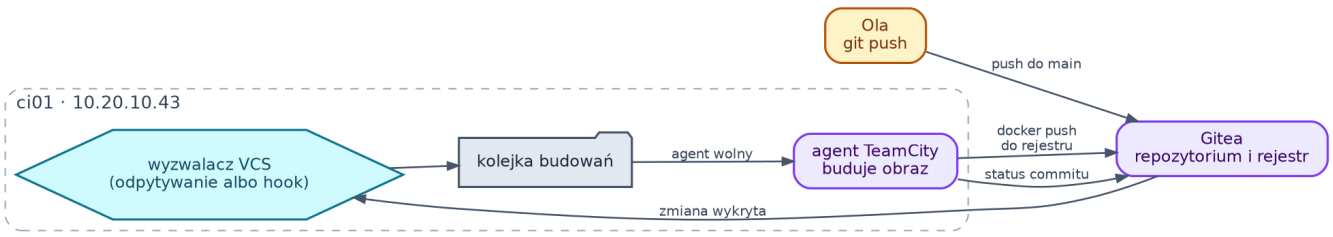
Limit licencji zablokował nowy build

Przyczyna. Darmowa edycja Professional liczy równocześnie istniejące konfiguracje budowania i pipeline'y, do stu i dziesięciu, oraz autoryzowanych agentów, do trzech. Po przekroczeniu limitu TeamCity nie kasuje nic samo z siebie: po prostu nie uruchamia nowych buildów, dopóki nie zwolnimy miejsca.

Naprawa. Archiwizujemy konfiguracje nieużywane od dawna (`Konfiguracja budowania → General Settings → Archive`) albo odautoryzowujemy agenta, który nikomu już nie służy. Dopiero gdy to nie wystarcza, rozważamy licencję Enterprise, opisaną na stronie Plan wdrożenia TeamCity w Zielonej.

Build czeka w kolejce, agent wygląda na wolnego

Droga zdarzenia od pushu do wyniku pomaga zlokalizować, na którym odcinku utknęło zadanie:



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

„Wolny" w interfejsie oznacza tylko, że agent nie wykonuje w tej chwili innego builda, nie że jest zgodny z wymaganiami akurat tej konfiguracji. Zakładka „Compatible Agents" pokazuje explicite, którzy z widocznych agentów w ogóle kwalifikują się do tego builda, i dlaczego reszta odpada, na przykład brak zainstalowanego Dockera albo niepasujący system operacyjny.

Status commitu w Gitea się nie pojawia

Ten objaw dotyczy zwykle jednego z dwóch tokenów, nie jednego uniwersalnego problemu z połączeniem. Token do rejestru pakietów (`write:package`) i token do statusu commitu (`repository`) to dwa osobne poświadczenia w Commit Status Publisher, mimo że oba pochodzą z tego samego konta w Gitea: pomyłka polegająca na wpisaniu jednego w miejsce drugiego kończy się różnymi błędami, żaden z nich nie mówi wprost „zły token".

□□ Ćwiczenia

1. Nowa konfiguracja budowania w TeamCity nie uruchamia się, a interfejs pokazuje komunikat o przekroczonym limicie licencji, mimo że zespół Oli ma tylko dwie aktywne konfiguracje. Wyjaśniamy, dlaczego limit i tak może być przekroczony.

Odpowiedź

Limit stu konfiguracji liczy wszystkie istniejące konfiguracje razem z zarchiwizowanymi, nie tylko te aktywnie używane. Stare, zapomniane konfiguracje z wcześniejszych eksperymentów, nigdy nieusunięte, wliczają się do tego samego limitu co dwie aktywne konfiguracje zespołu Oli.

1. Build wykonuje `Docker: Push Image` i kończy się błędem `unauthorized`, mimo że ten sam token działał poprawnie tydzień temu przy ręcznym `docker login` z własnej stacji. Wskazujemy najbardziej prawdopodobną przyczynę.

Odpowiedź

Token dostępu osobistego w Gitea mógł wygasnąć albo zostać odwołany od czasu ostatniego udanego builda: sprawdzamy datę ważności i zakres tokenu w ustawieniach konta Gitea, zamiast szukać przyczyny w samej konfiguracji TeamCity.

Powiązane: Plan wdrożenia TeamCity w Zielonej · Agenci TeamCity: dodanie, autoryzacja, pule · Pierwszy projekt: cennik z Gitea budowany w TeamCity · Logowanie kontem z Active Directory · Kopia zapasowa i aktualizacja TeamCity

Źródła: jetbrains.com/help/teamcity, rozdział „Licensing Policy" · jetbrains.com/help/teamcity, rozdział „Commit Status Publisher" · jetbrains.com/help/teamcity, rozdział „LDAP Integration"