Instalacja i agenci

W Dockerze i bez Dockera, agenci i pule.

- Instalacja TeamCity w Dockerze
- Instalacja TeamCity bez Dockera
- Agenci TeamCity: dodanie, autoryzacja, pule
- Agent TeamCity w Dockerze na maszynie fizycznej: Linux i Windows, agent, który przenosi się między hostami

Instalacja TeamCity w Dockerze

Typ: procedura

Dotyczy: TeamCity 2026.2, obrazy jetbrains/teamcity-server:2026.2 i jetbrains/teamcity-agent:2026.2

Czas: ~35 min

Efekt: działający serwer TeamCity z jednym agentem, dostępny po WWW na porcie 8111, z bazą PostgreSQL na `pg01`, gotowy do przejścia kreatora w przeglądarce.

Skróty użyte na tej stronie (5)

Skrót	Co znaczy
CLI	Command Line Interface: praca przez wpisywanie komend w terminalu
HSQLDB	HyperSQL DataBase: baza danych wbudowana w sam TeamCity, zapisywana na dysku obok reszty danych serwera, przeznaczona wyłącznie do testów i ewaluacji
JDBC	Java Database Connectivity: standardowy sposób, w jaki program w Javie łączy się z bazą danych
LDAP	Lightweight Directory Access Protocol: protokół do odpytywania katalogu użytkowników (AD, FreeIPA)
WWW	World Wide Web: strony i aplikacje otwierane w przeglądarce po HTTP albo HTTPS

Zanim zaczniemy

Komendy wykonujemy z CLI bash jako user z uprawnieniami sudo albo root, na `ci01`, z zainstalowanym Dockerem i wtyczką Compose. Poniżej dwa warianty: laboratoryjny, z wbudowaną bazą HSQLDB do samego zapoznania się z interfejsem, oraz produkcyjny Zielonej, z bazą na osobnym serwerze `pg01`. Do jakiejkolwiek pracy zespołu wybieramy wariant b: wariant a nie nadaje się do niczego poza pierwszym rzutem oka na ekran.

> W naszym środowisku dane aplikacji trzymamy w `/docker/apps/teamcity`, a plik Compose w `/docker/compose/teamcity/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.

Wymaganie	Uwaga
Docker Engine i wtyczka Compose	sprawdzamy `docker compose version`
Wolny port `8111` na hoście	interfejs WWW serwera
Wariant b: osiągalny `pg01` na porcie `5432`	baza tworzona ręcznie przed pierwszym startem
Wariant b: konto i baza założone w PostgreSQL na `pg01`	patrz krok 2 wariantu b

Uważaj: obraz zawsze przypinamy do konkretnej wersji (`2026.2`), nigdy do `latest`. Aktualizacja to świadoma zmiana numeru w `compose.yaml`, opisana na stronie Kopia zapasowa i aktualizacja TeamCity, nie przypadkowe pobranie nowszej wersji przy kolejnym `docker compose pull`.

Przebieg

Katalogi danych, zgodnie z konwencją ścieżek

```
sudo mkdir -p /docker/apps/teamcity/data /docker/apps/teamcity/logs
/docker/apps/teamcity/agent-conf
sudo mkdir -p /docker/compose/teamcity
```

Katalog `data` trzyma całą konfigurację serwera, w tym poświadczenia i klucz szyfrujący: to jest odpowiednik `JENKINS_HOME` po stronie TeamCity, opisanego w książce o Jenkinsie na stronie Architektura Jenkinsa, kontroler, agenci, workspace.

Wariant a: laboratorium, wbudowana baza HSQLDB

`/docker/compose/teamcity/compose.yaml`:

```
services:
  server:
    image: jetbrains/teamcity-server:2026.2
```

```
    container_name: teamcity-server
    restart: always
    ports:
      - "8111:8111"
    volumes:
      - /docker/apps/teamcity/data:/data/teamcity_server/datadir
      - /docker/apps/teamcity/logs:/opt/teamcity/logs

  agent:
    image: jetbrains/teamcity-agent:2026.2
    container_name: teamcity-agent-1
    restart: always
    environment:
      - SERVER_URL=http://server:8111
      - AGENT_NAME=ci01-agent-1
    volumes:
      - /docker/apps/teamcity/agent-conf:/data/teamcity_agent/conf
    depends_on:
      - server
```

Pierwsze uruchomienie:

```
cd /docker/compose/teamcity
docker compose up -d
```

Uważaj: ten wariant zapisuje bazę na dysku wewnątrz `datadir`, bez osobnego serwera bazy danych. Dokumentacja JetBrains wprost ogranicza wbudowaną bazę do testów i ewaluacji: brak wygodnej kopii zapasowej na żywo i słabsza wydajność przy większej liczbie buildów wykluczają ją z jakiejkolwiek pracy produkcyjnej.

Wariant b: produkcja Zielonej, baza na zewnętrznym pg01

Zanim uruchomimy kontener, zakładamy bazę i użytkownika na `pg01`. Łączymy się tam jako administrator PostgreSQL:

```
psql -h pg01 -U postgres
```

W sesji `psql` tworzymy bazę i konto aplikacji:

```
CREATE DATABASE teamcity WITH ENCODING 'UTF8';
CREATE USER teamcity WITH ENCRYPTED PASSWORD 'HasloDoBazy';
GRANT ALL PRIVILEGES ON DATABASE teamcity TO teamcity;
ALTER DATABASE teamcity OWNER TO teamcity;
```

TeamCity nie zawiera w obrazie sterownika JDBC do PostgreSQL: pobieramy go raz, do katalogu danych, gdzie serwer sam go znajdzie przy starcie.

```
sudo mkdir -p /docker/apps/teamcity/data/lib/jdbc
sudo curl -L -o /docker/apps/teamcity/data/lib/jdbc/postgresql-42.7.9.jar \
  https://jdbc.postgresql.org/download/postgresql-42.7.9.jar
```

Plik `/docker/compose/teamcity/.env` z hasłem do bazy, obok pliku `compose.yaml`, nie w nim samym:

```
TEAMCITY_DB_PASSWORD=HasloDoBazy
```

Plik `/docker/compose/teamcity/compose.yaml`:

```
services:
  server:
    image: jetbrains/teamcity-server:2026.2
    container_name: teamcity-server
    restart: always
    ports:
      - "8111:8111"
    environment:
      - TEAMCITY_DB_URL=jdbc:postgresql://pg01:5432/teamcity
      - TEAMCITY_DB_USER=teamcity
      - TEAMCITY_DB_PASSWORD=${TEAMCITY_DB_PASSWORD}
    volumes:
      - /docker/apps/teamcity/data:/data/teamcity_server/datadir
      - /docker/apps/teamcity/logs:/opt/teamcity/logs

  agent:
    image: jetbrains/teamcity-agent:2026.2
    container_name: teamcity-agent-1
    restart: always
    environment:
      - SERVER_URL=http://server:8111
      - AGENT_NAME=ci01-agent-1
    volumes:
```

```
      - /docker/apps/teamcity/agent-conf:/data/teamcity_agent/conf
      - /var/run/docker.sock:/var/run/docker.sock
    depends_on:
      - server
```

> Wolumen `/var/run/docker.sock` daje kontenerowi agenta pełną władzę nad demonem Dockera hosta `ci01`, dokładnie z tym samym ryzykiem, co przy runnerze Gitea opisanym na stronie Gitea Actions: runner i pierwszy workflow: build wykonany przez tego agenta mógłby zatrzymać albo odczytać dane dowolnego innego kontenera na `ci01`. Montujemy to gniazdo tylko wtedy, gdy agent faktycznie ma budować obrazy Dockera, i tylko dla projektów, którym ufamy.

Pierwsze uruchomienie:

```
cd /docker/compose/teamcity
docker compose up -d
```

Kreator instalacji w przeglądarce

Otwieramy adres serwera:

```
http://ci01:8111
```

Wariant b, z bazą już wskazaną przez zmienne środowiskowe, pomija ekran wyboru bazy danych i przechodzi wprost do akceptacji licencji, a potem do założenia pierwszego konta administratora. Dalsze kroki kreatora, LDAP i pierwszy projekt, opisują kolejne strony tej książki.

◫ ZRZUT EKRANU DO UZUPEŁNIENIA

Zrzut do uzupełnienia: pierwszy ekran otwarty pod `http://ci01:8111`, ekran „Database connection setup" z widoczną wartością „PostgreSQL" i statusem „Connection successful".

Weryfikacja

Sprawdzamy, że kontenery działają:

```
docker compose ps
```

Przykład:

```
NAME               IMAGE                                  STATUS
teamcity-server    jetbrains/teamcity-server:2026.2       Up 3 minutes
teamcity-agent-1   jetbrains/teamcity-agent:2026.2        Up 3 minutes
```

Sprawdzamy log serwera, szukając potwierdzenia połączenia z bazą:

```
docker compose logs --tail 50 server
```

Przykład:

```
teamcity-server  | [TeamCity] Using database: jdbc:postgresql://pg01:5432/teamcity
teamcity-server  | [TeamCity] Server startup finished
```

Sprawdzamy, że agent zdążył się już zarejestrować i czeka na autoryzację, zgodnie ze stroną Agenci TeamCity: dodanie, autoryzacja, pule:

```
Administracja → Agents → Unauthorized
```

Wycofanie zmian

Zatrzymanie i usunięcie kontenerów, bez kasowania danych z wolumenów:

```
docker compose down
```

Pełne usunięcie razem z danymi, tylko gdy naprawdę chcemy zacząć od zera:

```
docker compose down -v
sudo rm -rf /docker/apps/teamcity
```

Baza `teamcity` na `pg01` zostaje po tej operacji nietknięta: usuwamy ją osobno, tym samym poleceniem `DROP DATABASE teamcity`, tylko jeśli naprawdę nie planujemy wracać do tej instalacji.

Gdy coś nie zadziała

Objaw	Przyczyna	Naprawa

Kontener `server` restartuje się w pętli zaraz po starcie	brak sterownika JDBC w `data/lib/jdbc` albo `pg01` nieosiągalny	`docker compose logs server`, sprawdzamy plik `.jar` i `telnet pg01 5432` z hosta `ci01`
W logu serwera `FATAL: password authentication failed for user "teamcity"`	hasło w `.env` inne niż w bazie na `pg01`	poprawiamy hasło w `.env` albo w bazie przez `ALTER USER teamcity WITH PASSWORD`
Kreator w przeglądarce wisi na ekranie „Starting..." dłużej niż kilka minut	za mało pamięci przydzielonej kontenerowi serwera	sprawdzamy `docker stats teamcity-server`, podnosimy limit RAM hosta albo `TEAMCITY_SERVER_MEM_OPTS`
Agent w ogóle nie pojawia się na liście `Unauthorized`	zły adres w `SERVER_URL` albo agent nie widzi serwera po sieci Dockera	sprawdzamy `docker compose logs agent`, `SERVER_URL` musi wskazywać nazwę usługi `server`, nie `localhost`

Powiązane: Instalacja TeamCity bez Dockera · Agenci TeamCity: dodanie, autoryzacja, pule · Konwencja Dockera w Zielonej

Źródła: jetbrains.com/help/teamcity, rozdział „Installing and Configuring the TeamCity Server" · github.com/JetBrains/teamcity-docker-images, `dockerhub/teamcity-server/README.md` i `dockerhub/teamcity-agent/README.md` · jetbrains.com/help/teamcity, rozdział „Set up External Database"

Instalacja TeamCity bez Dockera

Typ: procedura **Dotyczy: TeamCity 2026.2, dystrybucja .tar.gz, Rocky Linux 9**

Czas: ~30 min

Efekt: serwer TeamCity uruchomiony jako usługa systemd na dedykowanym koncie systemowym, bez Dockera, z Javą, firewalld i katalogiem danych dopasowanymi do Rocky Linux 9.

Skróty użyte na tej stronie (4)

Skrót	Co znaczy
CLI	Command Line Interface: praca przez wpisywanie komend w terminalu
DNS	Domain Name System: książka telefoniczna internetu, zamienia nazwy na adresy IP
SELinux	Security-Enhanced Linux: dodatkowa warstwa uprawnień w jądrze RHEL
WWW	World Wide Web: strony i aplikacje otwierane w przeglądarce po HTTP albo HTTPS

Zanim zaczniemy

Komendy wykonujemy z CLI bash jako user z uprawnieniami sudo albo root. Serwer musi rozwiązywać nazwy DNS i mieć ruch wychodzący do `download.jetbrains.com`, żeby pobrać dystrybucję.

Wymaganie	Uwaga
Rocky Linux 9 albo RHEL 9	polecenia SELinux i firewalld poniżej dotyczą tej rodziny systemów
Java 21 (OpenJDK)	TeamCity nie ma własnej Javy w tym pakiecie, w przeciwieństwie do obrazu Dockera
Ruch wychodzący do `download.jetbrains.com`	pobranie dystrybucji `.tar.gz`
Wolny port `8111`	interfejs WWW serwera
Baza danych osiągalna z tego serwera	PostgreSQL na `pg01`, patrz Instalacja TeamCity w Dockerze po kroki założenia bazy

Kiedy wybieramy ten wariant zamiast Dockera:

Sytuacja	Docker	Bez Dockera
Mamy już standard Dockera w firmie	tak, spójnie z resztą aplikacji	dodatkowy wyjątek do utrzymania
Aktualizacja ma być podmianą jednego katalogu, bez rejestru obrazów	trzeba śledzić tagi obrazu	podmiana dystrybucji, prościej do audytu
Zespół zna dobrze systemd, SELinux i firewalld	mniej istotne, izolacja z kontenera	pełna kontrola, ale więcej ręcznej pracy

Przebieg

1. Instalacja Javy

```
sudo dnf install -y java-21-openjdk
```

TeamCity 2026.2 wymaga Javy w wersji 21: starsza Java kończy się błędem już przy pierwszym uruchomieniu skryptu startowego, nie dopiero przy pracy serwera.

2. Utworzenie konta systemowego teamcity

```
sudo useradd --system --shell /bin/bash --comment "TeamCity Server" --create-home --home-dir /var/lib/teamcity teamcity
```

Konto `teamcity` uruchamia usługę i jest właścicielem katalogu danych, bez prawa logowania hasłem.

3. Pobranie i rozpakowanie dystrybucji

```
curl -L -o TeamCity.tar.gz https://download.jetbrains.com/teamcity/TeamCity-2026.2.tar.gz
sudo mkdir -p /opt/teamcity
sudo tar -xzf TeamCity.tar.gz -C /opt/teamcity --strip-components=1
sudo chown -R teamcity:teamcity /opt/teamcity
```

Dystrybucja `.tar.gz` zawiera już dodatkowo agenta budującego, przydatnego do szybkich testów na tej samej maszynie: instalację osobnego agenta na innej maszynie opisuje strona Agenci TeamCity: dodanie, autoryzacja, pule.

4. Katalog danych

```
sudo mkdir -p /var/lib/teamcity/.BuildServer
sudo chown -R teamcity:teamcity /var/lib/teamcity
sudo chmod -R 750 /var/lib/teamcity
```

> **Pojęcie: katalog danych TeamCity.** Miejsce, w którym serwer trzyma konfigurację projektów, poświadczenia i klucz szyfrujący, niezależnie od samej instalacji programu w `/opt/teamcity`. Domyślnie to `.BuildServer` w katalogu domowym konta uruchamiającego usługę: utrata tego katalogu bez kopii zapasowej, opisanej na stronie Kopia zapasowa i aktualizacja TeamCity, oznacza utratę całej historii i konfiguracji.

5. Jednostka systemd

Plik `/etc/systemd/system/teamcity.service`:

```
[Unit]
Description=TeamCity Server
After=network.target

[Service]
Type=forking
User=teamcity
Group=teamcity
Environment=JAVA_HOME=/usr/lib/jvm/jre-21
Environment=TEAMCITY_DATA_PATH=/var/lib/teamcity/.BuildServer
ExecStart=/opt/teamcity/bin/teamcity-server.sh start
ExecStop=/opt/teamcity/bin/teamcity-server.sh stop
RestartSec=5s
Restart=on-failure

[Install]
```

```
WantedBy=multi-user.target
```

Włączamy i uruchamiamy usługę:

```
sudo systemctl daemon-reload
sudo systemctl enable --now teamcity
```

Sprawdzamy status:

```
sudo systemctl status teamcity
```

Przykład:

```
● teamcity.service - TeamCity Server
     Loaded: loaded (/etc/systemd/system/teamcity.service; enabled; preset: disabled)
     Active: active (running) since Sun 2026-09-27 11:02:10 UTC; 14s ago
```

6. SELinux i firewalld na Rocky 9

Otwieramy port WWW na firewalld:

```
sudo firewall-cmd --add-port=8111/tcp --permanent
sudo firewall-cmd --reload
```

Weryfikujemy otwarte porty:

```
sudo firewall-cmd --list-ports
```

Przykład:

```
8111/tcp
```

Sprawdzamy kontekst SELinux katalogu danych, gdyby serwer nie mógł do niego zapisywać mimo poprawnych uprawnień systemowych:

```
ls -Z /var/lib/teamcity
```

Jeśli kontekst różni się od oczekiwanego typu dla katalogów domowych, przywracamy domyślny:

```
sudo restorecon -Rv /var/lib/teamcity
```

Weryfikacja

```
curl -I http://localhost:8111
```

Przykład:

```
HTTP/1.1 200 OK
```

Sprawdzamy logi serwera, gdyby coś odbiegało od oczekiwań:

```
sudo tail -n 50 /opt/teamcity/logs/teamcity-server.log
```

◫ ZRZUT EKRANU DO UZUPEŁNIENIA

Zrzut do uzupełnienia: pierwszy ekran kreatora TeamCity otwarty pod `http://ci01:8111`, ekran akceptacji licencji z widocznym numerem wersji „2026.2".

Wycofanie zmian

```
sudo systemctl disable --now teamcity
sudo rm -f /etc/systemd/system/teamcity.service
sudo systemctl daemon-reload
sudo rm -rf /opt/teamcity /var/lib/teamcity
sudo userdel -r teamcity
```

Gdy coś nie zadziała

Objaw	Przyczyna	Naprawa
`systemctl status teamcity` pokazuje `failed` zaraz po starcie	zła wersja Javy albo brak `JAVA_HOME` widocznego dla usługi	`journalctl -u teamcity`, sprawdzamy `java -version` na koncie `teamcity`
`curl -I http://localhost:8111` zwraca `Connection refused`, mimo że usługa działa	firewalld blokuje port	`firewall-cmd --list-ports`, dodajemy `8111/tcp`
Serwer startuje, ale zapisuje dane w nieoczekiwanym katalogu	`TEAMCITY_DATA_PATH` nieustawiony albo inny katalog domowy konta `teamcity`	sprawdzamy zmienną w jednostce systemd i katalog domowy z `getent passwd teamcity`

Objaw	Przyczyna	Naprawa
Błąd `Permission denied` przy zapisie do `.BuildServer` mimo poprawnego `chown`	kontekst SELinux katalogu niezgodny z oczekiwanym	`ls -Z /var/lib/teamcity`, `restorecon -Rv /var/lib/teamcity`

Powiązane: Instalacja TeamCity w Dockerze · Agenci TeamCity: dodanie, autoryzacja, pule · Instalacja Gitea bez Dockera

Źródła: jetbrains.com/help/teamcity, rozdział „Install TeamCity Server on Linux or macOS" · jetbrains.com/help/teamcity, rozdział „Start TeamCity Server" · jetbrains.com/help/teamcity, „Cheatsheet: Updating Java on Server and Agent Machines" · `man 8 firewall-cmd`

Agenci TeamCity: dodanie, autoryzacja, pule

Poziom: podstawy **Dotyczy: TeamCity 2026.2** **Czas: ~25 min**

Serwer TeamCity sam niczego nie buduje: każdy krok builda wykonuje osobny proces, agent. Ta strona tłumaczy, czym jest agent i pula agentów, jak dodać kolejnego agenta w kontenerze albo na osobnej maszynie, i co w praktyce oznacza darmowy limit trzech agentów.

Fiszka

KLIKAMY W INTERFEJSIE

Autoryzacja nowego agenta w interfejsie: Administracja → Agents → Unauthorized

```
Administracja → Agents → Unauthorized
```

Świeżo podłączony agent czeka tutaj, widoczny, ale bez prawa wykonywania buildów, dopóki administrator go nie zatwierdzi jednym kliknięciem „Authorize".

TYLKO ODCZYT

Dodanie agenta na osobnej maszynie bez Dockera: agent.sh start

```
sudo -u nazwa_usera /opt/buildagent/bin/agent.sh start
```

Wykonujemy po rozpakowaniu archiwum agenta i wskazaniu adresu serwera w pliku `buildAgent.properties`, patrz krok w sekcji „Jak to działa" niżej.

TYLKO ODCZYT

Sprawdzenie listy agentów z poziomu REST API: curl

```
curl -H "Authorization: Bearer TOKEN_DOSTEPU" https://ci.zielona.example/app/rest/agents
```

Zwraca listę wszystkich znanych serwerowi agentów, autoryzowanych i nie, razem z ich stanem połączenia: przydatne do szybkiego sprawdzenia, ile agentów faktycznie zużywa dziś darmowy limit trzech.

Przykład:

```
<agents count="3">
  <agent id="1" name="ci01-agent-1" connected="true" authorized="true"/>
  <agent id="2" name="ci01-agent-2" connected="true" authorized="true"/>
  <agent id="3" name="laptop-tomka" connected="true" authorized="false"/>
</agents>
```

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

Z życia wzięte

Z ŻYCIA WZIĘTE

Tomek, chcąc przyspieszyć testowanie, podłącza swój roboczy laptop jako czwartego agenta, obok dwóch już pracujących na `ci01`. Agent pojawia się na liście `Unauthorized`, ale przycisk „Authorize" kończy się komunikatem o przekroczonym limicie licencji. Trzy darmowe agenty to trzy agenty naraz, zatwierdzone, nie trzy dowolnie wymieniane: żeby dodać czwartego na stałe, ktoś musiałby najpierw odautoryzować jednego z istniejących albo dokupić licencję.

Jak to działa

Rozwiń: jak to działa pod spodem

Pojęcie: agent. Osobny proces, uruchomiony w kontenerze albo na osobnej maszynie, który faktycznie wykonuje kroki builda: checkout kodu, kompilację, testy, budowanie obrazu. Serwer TeamCity zleca agentowi zadanie z kolejki i czeka na wynik, podobnie jak kontroler Jenkinsa zleca zadanie swojemu agentowi, opisanemu na stronie Architektura Jenkinsa, kontroler, agenci, workspace.

Pojęcie: pula agentów (agent pool). Grupa agentów przypisana do konkretnych projektów: build z projektu przypisanego do danej puli trafia wyłącznie do agentów tej puli, nigdy do agentów przypisanych gdzie indziej. Każda nowa instalacja ma jedną pulę domyślną (`Default`), obejmującą wszystkich nieautoryzowanych jeszcze agentów: dodatkowe pule zakładamy dopiero wtedy, gdy różne zespoły albo różne projekty mają nie dzielić między sobą tych samych maszyn.

Pojęcie: wymagania agenta (agent requirements). Warunki, które konfiguracja budowania stawia agentowi, zanim ten w ogóle dostanie zlecenie: konkretna wersja narzędzia, konkretny system operacyjny, obecność Dockera. TeamCity dopasowuje wymagania do parametrów zgłoszonych przez każdego agenta automatycznie, zamiast wymagać ręcznie przypiętej etykiety tekstowej, jak w Jenkinsie: agent zgłasza to, co ma zainstalowane, a konfiguracja opisuje, czego potrzebuje, i dopasowanie dzieje się samo.

Agent w kontenerze

Agent w kontenerze, opisany na stronie Instalacja TeamCity w Dockerze, rejestruje się w serwerze sam, od razu po pierwszym starcie, zgłaszając parametry środowiska (system, zainstalowane narzędzia) automatycznie. Restart kontenera nie zakłada nowego agenta od zera: tożsamość agenta trzyma się w katalogu `conf`, zamontowanym jako trwały wolumen, dokładnie tak jak plik `.runner` po stronie Gitea Actions.

Agent na osobnej maszynie, bez Dockera

Pobieramy archiwum agenta ze strony serwera (dostępne bez logowania pod adresem serwera, zakładka pobierania agenta) i rozpakowujemy je na docelowej maszynie. Kluczowy plik konfiguracyjny to `conf/buildAgent.properties`:

```
serverUrl=http://ci01:8111
name=laptop-tomka
```

Uruchamiamy agenta skryptem `bin/agent.sh start`, tak jak w Fiszce wyżej. Od tego momentu agent zgłasza się do serwera i czeka w sekcji `Unauthorized`, dokładnie jak agent kontenerowy.

Autoryzacja i co dalej

Nowy agent, autoryzowany czy nie, od razu trafia do puli `Default`. Przenosimy go do innej puli z poziomu `Administracja → Agents → Agent Pools`, przeciągając albo wybierając pulę z listy rozwijanej przy agencie. Dopiero autoryzowany agent w puli przypisanej do danego projektu bierze udział w kolejce buildów tego projektu.

> 🖼 Zrzut do uzupełnienia: ekran Administracja → Agents → Unauthorized z widocznym wierszem nowego agenta i przyciskiem „Authorize", zaraz po podłączeniu trzeciego agenta w Zielonej.

Limit trzech agentów w praktyce

Darmowa edycja Professional liczy agenty autoryzowane naraz, nie łączną liczbę maszyn, które kiedykolwiek się podłączyły. Odautoryzowanie starego agenta zwalnia miejsce dla nowego bez żadnej dodatkowej opłaty: licencja liczy stan bieżący, nie historię. Dla zespołu Oli, pracującego dziś na dwóch agentach kontenerowych na `ci01`, trzeci wolny slot starcza na agenta testowego albo tymczasowy agent na laptopie, bez zbliżania się do granicy opisanej na stronie Plan wdrożenia TeamCity w Zielonej.

Częste błędy

Objaw	Przyczyna	Co robimy
Przycisk „Authorize" zwraca błąd o przekroczonym limicie licencji	trzej agenci autoryzowani już zajmują cały darmowy limit	odautoryzowujemy nieużywanego agenta albo dokupujemy licencję, patrz Plan wdrożenia TeamCity w Zielonej
Agent widoczny jako podłączony, ale żaden build nie trafia do niego	agent w innej puli niż projekt, którego build czeka w kolejce	Administracja → Agents → Agent Pools, sprawdzamy przypisanie projektu i agenta

Objaw	Przyczyna	Co robimy
Nowy agent w ogóle nie pojawia się na liście `Unauthorized`	zły adres w `serverUrl` (bez Dockera) albo w `SERVER_URL` (w kontenerze)	sprawdzamy log agenta, adres musi być osiągalny z jego maszyny, nie tylko z przeglądarki administratora
Build czeka w kolejce mimo wolnego agenta	agent nie spełnia wymagań agenta zadeklarowanych przez konfigurację (brak narzędzia, zły system)	sprawdzamy zakładkę „Compatible Agents" konfiguracji budowania, porównujemy z parametrami agenta

□□ Ćwiczenia

1. Zespół Oli ma dwóch autoryzowanych agentów i chce podłączyć trzeciego, tymczasowego, na laptopie Tomka na czas jednego eksperymentu. Wyjaśniamy, czy zmieści się to w darmowej licencji, i co się stanie po zakończeniu eksperymentu, jeśli nikt nie odautoryzuje tego agenta.

Odpowiedź

Trzeci agent mieści się dokładnie w darmowym limicie trzech autoryzowanych agentów. Jeśli nikt go nie odautoryzuje po zakończeniu eksperymentu, zajmuje na stałe ostatni wolny slot licencji: kolejny agent, potrzebny na przykład do stałej pracy zespołu, odbije się od tego samego limitu, dopóki ktoś nie zwolni miejsca albo nie dokupi licencji.

1. Build konfiguracji wymagającej Dockera czeka w kolejce mimo dwóch widocznych, autoryzowanych i wolnych agentów. Wskazujemy, gdzie sprawdzamy przyczynę w pierwszej kolejności.

Odpowiedź

Zakładkę „Compatible Agents" tej konfiguracji budowania: pokazuje ona, które z autoryzowanych agentów w ogóle spełniają wymagania (na przykład obecność Dockera), zanim zaczniemy szukać przyczyny gdzie indziej.

Powiązane: Instalacja TeamCity w Dockerze · Instalacja TeamCity bez Dockera · Architektura Jenkinsa, kontroler, agenci, workspace · Plan wdrożenia TeamCity w Zielonej · Agent TeamCity w Dockerze na maszynie fizycznej: Linux i Windows, agent, który przenosi się między hostami

Źródła: jetbrains.com/help/teamcity, rozdział „Configuring Agent Pools" · jetbrains.com/help/teamcity, rozdział „Assigning Build Configurations to Specific Build Agents" · jetbrains.com/help/teamcity/rest, rozdział „Agent"

Agent TeamCity w Dockerze na maszynie fizycznej: Linux i Windows, agent, który przenosi się między hostami

Poziom: średni

Dotyczy: TeamCity 2026.2, obraz jetbrains/teamcity-agent, Docker, Rocky Linux 9, Windows Server 2022

Czas: ~50 min

Agent TeamCity to program, który faktycznie buduje i testuje kod, więc czasem musi stać na prawdziwej maszynie: z kartą graficzną, dużym dyskiem albo Windowsem z narzędziami do budowania. Ta strona pokazuje, jak uruchomić takiego agenta w kontenerze na Linuksie i na Windowsie oraz jak go przenieść na inny host, nie zużywając drugiej licencji. Wracamy tu, gdy agent ma zmienić maszynę albo gdy kończy się darmowy limit trzech agentów.

Komendy wykonujemy z CLI bash jako user z uprawnieniami sudo na hoście z Linuksem oraz w PowerShellu jako administrator na hoście z Windows Server. Wersje obrazów i parametry sprawdzone 3 października 2026. Podstawy agentów (pule, autoryzacja, limit trzech) opisuje strona Agenci TeamCity: dodanie, autoryzacja, pule, a serwer z agentem na `ci01` strona Instalacja TeamCity w Dockerze. Tu zajmujemy się tym, co dochodzi przy osobnej maszynie fizycznej.

> W naszym środowisku dane aplikacji trzymamy w `/docker/apps/teamcity-agent`, a plik Compose w `/docker/compose/teamcity-agent/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.

Skróty użyte na tej stronie (7)

Skrót	Co znaczy

CLI	Command Line Interface: praca przez wpisywanie komend w terminalu
GPU	Graphics Processing Unit: karta graficzna, która liczy modele językowe wielokrotnie szybciej niż procesor
HTTP	Hypertext Transfer Protocol: protokół stron WWW
HTTPS	HTTP zabezpieczony szyfrowaniem TLS
LVM	Logical Volume Manager: warstwa w Linuksie składająca dyski w elastyczne wolumeny
README	Read Me: plik tekstowy z opisem projektu, wykresu lub szablonu, czytany jako pierwszy
SDK	Software Development Kit: zestaw narzędzi i bibliotek do budowania programów dla jakiejś platformy, tu zależny od hosta agenta budującego

Fiszka

□□ ZMIENIA SYSTEM

Instalacja NVIDIA Container Toolkit na Rocky Linux 9: dnf install

Sterownik karty zakładamy według strony Modele lokalne: AI na własnym sprzęcie. Dopiero potem dodajemy narzędzie, które pozwala kontenerowi korzystać z karty. Komendy można wykonywać całościowo jako blok tekstu:

```
# repozytorium NVIDIA dla pakietów kontenerowych
curl -s -L https://nvidia.github.io/libnvidia-container/stable/rpm/nvidia-container-
toolkit.repo | sudo tee /etc/yum.repos.d/nvidia-container-toolkit.repo
# pakiet z narzędziem
sudo dnf install -y nvidia-container-toolkit
# wskazanie Dockera jako środowiska, które ma używać karty
sudo nvidia-ctk runtime configure --runtime=docker
sudo systemctl restart docker
```

Dokładne nazwy pakietów i wersji sprawdzamy w instrukcji instalacji NVIDIA, bo zmieniają się z wydaniami. Restart Dockera zatrzymuje kontenery na tym hoście, więc robimy go w oknie serwisowym.

□□ **AI** = Artificial Intelligence: sztuczna inteligencja, czyli programy naśladujące ludzkie rozumowanie · **curl** = client URL: pobieranie i wysyłanie przez HTTP z terminala · **dnf** = Dandified YUM: menedżer pakietów RHEL/Rocky · **nvidia-ctk** = NVIDIA Container Toolkit CLI: konfiguruje środowisko uruchomieniowe kontenerów tak, aby widziały kartę graficzną NVIDIA · **systemctl** = system control: sterowanie usługami systemd

◫◫ OSTROŻNIE · NIEODWRACALNE

Sprawdzenie karty z poziomu kontenera: docker run --gpus

```
docker run --rm --gpus all ubuntu:24.04 nvidia-smi
```

Jeśli kontener wypisze nazwę karty i pamięć, agent też ją zobaczy. Błąd przy tym poleceniu oznacza problem ze sterownikiem albo z narzędziem z poprzedniej fiszki, nie z TeamCity.

Przykład:

```
| NVIDIA-SMI 570.00       Driver Version: 570.00       CUDA Version: 12.8     |
|   0  NVIDIA GeForce RTX 4090   ...   |  1MiB / 24564MiB |
```

◫◫ **docker** = narzędzie do kontenerów

◫◫ ZMIENIA SYSTEM

Utworzenie katalogów agenta: mkdir

```
sudo mkdir -p /docker/apps/teamcity-agent/conf /docker/compose/teamcity-agent
sudo mkdir -p /opt/buildagent/work /opt/buildagent/temp /opt/buildagent/tools
/opt/buildagent/plugins /opt/buildagent/system
```

Katalog `conf` trzyma tożsamość agenta, więc po jego utracie agent staje się nowym agentem. Katalogi `/opt/buildagent/...` to miejsca na pobrany kod i pamięć podręczną, które rosną razem z historią buildów. Zgodnie z konwencją LVM dajemy im osobny wolumen `vgTeamCity_Agent/lvTeamCity_Agent` zamontowany jako `/opt/buildagent` (procedura na stronie Instalacja Rocky Linux z własnym nazewnictwem VG i LV).

◫◫ **LVM** = Logical Volume Manager: warstwa w Linuksie składająca dyski w elastyczne wolumeny · **VG** = Volume Group: grupa wolumenów w LVM, pula złożona z dysków · **LV** = Logical Volume: wolumen logiczny w LVM, na nim leży system plików · **mkdir** = make directory: utwórz katalog

◫◫ TYLKO ODCZYT

Plik Compose agenta z gniazdem Dockera hosta: compose.yaml

```
sudo vim /docker/compose/teamcity-agent/compose.yaml
```

```
services:
  agent:
    image: jetbrains/teamcity-agent:2026.2
    container_name: teamcity-agent-gpu
    restart: unless-stopped
    user: "0"
    environment:
      SERVER_URL: "http://ci01:8111"
      AGENT_NAME: "agent-gpu"
      GPU_VRAM_GB: "24"
    volumes:
      - /docker/apps/teamcity-agent/conf:/data/teamcity_agent/conf
      - /var/run/docker.sock:/var/run/docker.sock
      - /opt/buildagent/work:/opt/buildagent/work
      - /opt/buildagent/temp:/opt/buildagent/temp
      - /opt/buildagent/tools:/opt/buildagent/tools
      - /opt/buildagent/plugins:/opt/buildagent/plugins
      - /opt/buildagent/system:/opt/buildagent/system
    deploy:
      resources:
        reservations:
          devices:
            - driver: nvidia
              count: 1
              capabilities: [gpu]
```

(gdzie `SERVER_URL` to adres serwera osiągalny z tego hosta, a nie `localhost`, który w kontenerze oznacza sam kontener)

Obraz ma przypiętą wersję `2026.2`, tak samo jak serwer. Katalogi `/opt/buildagent/...` mają na hoście i w kontenerze tę samą ścieżkę, bo kroki budowania uruchamiane przez gniazdo Dockera montują katalogi według ścieżek hosta. Zmienna `GPU_VRAM_GB` trafia do agenta jako parametr `env.GPU_VRAM_GB`, którego użyjemy w wymaganiach buildów. Przy `SERVER_URL` przez HTTP kod i poświadczenia idą siecią bez szyfrowania, więc po dodaniu CA-Zielona do magazynu zaufania agenta przechodzimy na HTTPS.

> Po przenosinach urzędu według strony Przenosiny urzędu z dc01 bez przerwy: dwa urzędy równolegle i wygaszenie CA-Zielona dokładamy tu w ten sam sposób zaufanie do `Zielona-Root-CA`, a `CA-Zielona.crt` zostaje obok do wygaszenia starego urzędu.

HTTP = Hypertext Transfer Protocol: protokół stron WWW · **HTTPS** = HTTP zabezpieczony szyfrowaniem TLS · **vim** = Vi IMproved: ulepszony edytor vi

PLIK KONFIGURACYJNY

Plik Compose agenta z Dockerem w Dockerze: compose.yaml

```
services:
  agent:
    image: jetbrains/teamcity-agent:2026.2-linux-sudo
    container_name: teamcity-agent-dind
    restart: unless-stopped
    user: "0"
    privileged: true
    environment:
      SERVER_URL: "http://ci01:8111"
      AGENT_NAME: "agent-dind"
      DOCKER_IN_DOCKER: "start"
    volumes:
      - /docker/apps/teamcity-agent/conf:/data/teamcity_agent/conf
      - /docker/apps/teamcity-agent/docker:/var/lib/docker
```

Agent uruchamia własnego demona Dockera w kontenerze, więc nie dotyka obrazów i kontenerów hosta. Wymaga trybu `privileged`, a ten też daje szerokie prawa na hoście, więc to wybór mniejszego zła, nie bezpieczne rozwiązanie. Karty graficznej w tym wariancie nie przekazujemy: dostęp do karty z wewnętrznego Dockera to osobny temat, który sprawdzamy w dokumentacji NVIDIA.

ZMIENIA SYSTEM

Pierwszy start agenta: docker compose up -d

```
cd /docker/compose/teamcity-agent
docker compose up -d
```

Przykład:

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

□□ **cd** = change directory: przejdź do katalogu · **docker** = narzędzie do kontenerów

□□ KLIKAMY W INTERFEJSIE

Autoryzacja agenta po pierwszym starcie

```
Agents → Unauthorized → agent-gpu → Authorize
```

Nowy agent czeka na liście nieautoryzowanych, dopóki administrator go nie zatwierdzi. Zatwierdzenie zajmuje jedno z trzech darmowych miejsc.

□□ ZRZUT EKRANU DO UZUPEŁNIENIA

Zrzut do uzupełnienia: ekran Agents → Unauthorized z wierszem `agent-gpu`, widocznym adresem `10.20.10.45` i przyciskiem Authorize, zrobiony zaraz po pierwszym starcie kontenera na `gpu01`.

□□ TYLKO ODCZYT

Sprawdzenie tożsamości agenta bez pokazywania tokenu: sed

```
sudo sed 's/^authorizationToken=.*/authorizationToken=UKRYTY/' /docker/apps/teamcity-agent/conf/buildAgent.properties
```

Plik `buildAgent.properties` zawiera nazwę agenta i token, po którym serwer rozpoznaje tę maszynę jako zatwierdzonego agenta. Token traktujemy jak hasło: nie wklejamy go na czat, do zgłoszeń ani do repozytorium.

Przykład:

```
serverUrl=http://ci01:8111
name=agent-gpu
authorizationToken=UKRYTY
```

□□ **sed** = stream editor: edytor strumienia tekstu

□□ ZMIENIA SYSTEM

Sprawdzenie zgodności wersji Windows hosta i obrazu: docker image inspect

```
(Get-ItemProperty 'HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion').CurrentBuildNumber
docker pull jetbrains/teamcity-agent:2026.2-windowsservercore-2022
docker image inspect jetbrains/teamcity-agent:2026.2-windowsservercore-2022 --format "{{.OsVersion}}"
```

Pierwsze polecenie podaje numer kompilacji hosta, trzecie wersję systemu w obrazie. Dla Windows Server 2022 w obu wynikach widzimy numer `20348`, a dla Windows Server 2019 numer `17763` (obraz z tagiem `1809`).

Przykład:

```
20348
10.0.20348.2113
```

HKLM = HKEY_LOCAL_MACHINE: gałąź rejestru Windows z ustawieniami całego komputera, a nie pojedynczego użytkownika · **NT** = pierwotnie kryptonim N-Ten, od procesora, dla którego system miał powstać; popularnie dziś odczytywane jako New Technology, choć to rozwinięcie nie jest oficjalne: rdzeń dzisiejszego Windows Server · **docker** = narzędzie do kontenerów

ZMIENIA SYSTEM

Przygotowanie katalogów agenta na Windows Server: icacls

```
New-Item -ItemType Directory -Force D:\docker\apps\teamcity-agent\conf
icacls D:\docker\apps\teamcity-agent\conf /grant "Authenticated Users:(OI)(CI)F"
```

Przy izolacji procesowej Docker używa konta, które bez tego uprawnienia nie zapisze plików w katalogu wolumenu. Katalogu `C:\BuildAgent\work` nie montujemy jako wolumenu, bo Git for Windows kończy wtedy błędem ścieżki, co opisuje dokumentacja JetBrains.

icacls = PowerShell/CMD: odczytaj albo nadaj uprawnienia NTFS na pliku lub folderze z poziomu konsoli

ZMIENIA SYSTEM

Uruchomienie agenta w kontenerze Windows: docker run

```
docker run -d --name teamcity-agent-win --restart unless-stopped --isolation=process `
  -e SERVER_URL="http://ci01:8111" `
  -e AGENT_NAME="NazwaAgenta" `
```

```
    -v D:\docker\apps\teamcity-agent\conf:C:\BuildAgent\conf `
    jetbrains/teamcity-agent:2026.2-windowsservercore-2022
```

(gdzie `NazwaAgenta` to unikalna nazwa agenta na serwerze, a `D:\docker\apps\teamcity-agent\conf` to katalog z tożsamością, jedyny, który musi przetrwać wymianę kontenera)

Obraz `windowsservercore` pasuje do klasycznych narzędzi budowania, a `nanoserver` jest odchudzony, bez PowerShella, więc wybieramy go tylko, gdy budujemy .NET i nie potrzebujemy instalatorów. Na hoście musi działać rola kontenerów Windows i silnik Dockera, co opisuje dokumentacja Microsoftu.

docker = narzędzie do kontenerów

TYLKO ODCZYT

Zatrzymanie agenta na hoście źródłowym: docker compose stop

```
cd /docker/compose/teamcity-agent
docker compose stop
```

To pierwszy krok przeniesienia agenta. Zatrzymujemy go przed kopiowaniem, żeby plik z tożsamością był spójny i żeby dwa procesy z jednym tokenem nigdy nie działały równocześnie.

cd = change directory: przejdź do katalogu · **docker** = narzędzie do kontenerów

TYLKO ODCZYT

Spakowanie konfiguracji agenta: tar

```
sudo tar -C /docker/apps/teamcity-agent -czpf /tmp/agent-gpu-conf.tar.gz conf
```

Archiwum zawiera token, więc zapisujemy je w `/tmp` tylko na czas przesłania, a po wszystkim usuwamy.

tar = tape archive: archiwum plików (kiedyś na taśmę)

ZMIENIA SYSTEM

Przesłanie konfiguracji na host docelowy: scp

```
sudo chmod 600 /tmp/agent-gpu-conf.tar.gz
sudo chown nazwa_usera /tmp/agent-gpu-conf.tar.gz
scp /tmp/agent-gpu-conf.tar.gz nazwa_usera@NazwaHostaDocelowego:/tmp/
```

(gdzie `NazwaHostaDocelowego` to host, na którym agent ma zacząć pracę, u nas `gpu01`)

chmod = change mode: zmień prawa dostępu · **chown** = change owner: zmień właściciela · **scp** = secure copy: kopiowanie plików przez SSH

OSTROŻNIE · NIEODWRACALNE

Rozpakowanie i start agenta na hoście docelowym: tar, docker compose up -d

Na hoście docelowym najpierw przygotowujemy katalogi i plik Compose tak jak na hoście źródłowym (fiszki wyżej). Potem:

```
sudo tar -C /docker/apps/teamcity-agent -xzpf /tmp/agent-gpu-conf.tar.gz
sudo rm -f /tmp/agent-gpu-conf.tar.gz
cd /docker/compose/teamcity-agent
docker compose up -d
```

Agent wstaje z tym samym `name` i tym samym `authorizationToken`, więc serwer widzi znanego agenta, który po prostu łączy się z innego adresu.

tar = tape archive: archiwum plików (kiedyś na taśmę) · **rm** = remove: usuń (bez kosza, na zawsze) · **cd** = change directory: przejdź do katalogu · **docker** = narzędzie do kontenerów

TYLKO ODCZYT

Sprawdzenie liczby autoryzowanych agentów: curl

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

(gdzie `TokenDostepu` to token dostępu użytkownika TeamCity z prawem odczytu listy agentów)

Po przeniesieniu liczba ma zostać taka sama jak przed nim, a agent o tej nazwie ma być połączony.

Przykład:

```
<agents count="3">
  <agent id="1" name="ci01-agent-1" connected="true" authorized="true"/>
  <agent id="2" name="ci01-agent-2" connected="true" authorized="true"/>
  <agent id="3" name="agent-gpu" connected="true" authorized="true"/>
</agents>
```

□□ **curl** = client URL: pobieranie i wysyłanie przez HTTP z terminala

□□ **OSTROŻNIE · NIEODWRACALNE**

Usunięcie starej konfiguracji na hoście źródłowym: rm -rf

```
cd /docker/compose/teamcity-agent
docker compose down
sudo rm -rf /docker/apps/teamcity-agent/conf /tmp/agent-gpu-conf.tar.gz
```

Robimy to dopiero wtedy, gdy agent działa na hoście docelowym. Kopia tokenu, która zostaje na starej maszynie, jest pułapką: pierwszy `docker compose up -d` wykonany tam z rozpędu uruchomi drugiego agenta z tą samą tożsamością.

□□ **cd** = change directory: przejdź do katalogu · **docker** = narzędzie do kontenerów · **rm** = remove: usuń (bez kosza, na zawsze)

□□ **KLIKAMY W INTERFEJSIE**

Przeniesienie z ponowną autoryzacją: cofnięcie i nadanie licencji

```
Agents → agent-gpu → Unauthorize
(start kontenera na nowym hoście z pustym katalogiem conf)
Agents → Unauthorized → agent-gpu → Authorize
```

To wariant bez kopiowania tokenu. Cofnięcie autoryzacji zwalnia miejsce, a nowy agent z pustym `conf` dostaje nowy token i nową tożsamość na serwerze. Pulę, stan włączony i wyłączony oraz własne parametry przypisujemy mu od nowa.

□□ **TYLKO ODCZYT**

Wymagania builda wobec agenta: Kotlin DSL

```
requirements {
    startsWith("teamcity.agent.jvm.os.name", "Linux")
    exists("env.GPU_VRAM_GB")
}
```

Build z takimi wymaganiami trafi tylko do agenta na Linuksie, który zgłosił parametr `env.GPU_VRAM_GB`. To samo ustawiamy w interfejsie w ustawieniach konfiguracji budowania, w sekcji Agent Requirements.

DSL = Domain Specific Language: język programowania zaprojektowany wąsko, do jednego zadania, tu do zapisania konfiguracji TeamCity jako kod w Kotlinie

Z życia wzięte

Z ŻYCIA WZIĘTE

Środa, 9:20. Ola chce, żeby testy osadzania z projektu `import-cen` liczyły się na karcie w `gpu01`, a nie na procesorze `ci01`, gdzie trwają czterdzieści minut. Pierwszy pomysł, czyli nowy agent na `gpu01`, kończy się komunikatem o limicie, bo trzy miejsca zajmują `ci01-agent-1`, `ci01-agent-2` i `agent-gpu`. Ola zatrzymuje więc `agent-gpu` na `ci01`, kopiuje jego katalog `conf` na `gpu01` i uruchamia kontener z tą samą nazwą. W interfejsie nadal widać trzy autoryzowane agenty, a build trwa cztery minuty. Tomek zauważa jeszcze, że Ollama trzyma model w pamięci karty przez trzydzieści minut, więc pierwszy build po teście czatu musi poczekać na wolną pamięć.

Jak to działa

Rozwiń: jak to działa pod spodem

Po co agent na fizycznej maszynie

Serwer TeamCity nie musi widzieć sprzętu, ale agent tak. Trzy powody są najczęstsze: karta graficzna do testów modeli, bardzo duży dysk na kod, obrazy i pamięć podręczną oraz Windows z narzędziami do budowania, których nie postawimy w kontenerze Linuksa. W Zielonej karta stoi w `gpu01`, więc tam ląduje agent do testów z kartą.

Sytuacja	Agent na hoście fizycznym	Agent na maszynie wirtualnej
Karta graficzna do obliczeń	najprostsze przekazanie karty do kontenera	wymaga przekazania karty do maszyny wirtualnej
Szybki dysk i duża pamięć podręczna	pełna szybkość lokalnych dysków	zależy od puli dysków hosta
Windows z narzędziami budowania	kontener Windows albo agent bez kontenera	wygodne, łatwo klonować
Wyizolowanie od reszty serwera	słabsze, agent dzieli host	silniejsze, własny system

Warianty obrazu agenta

Obraz `jetbrains/teamcity-agent` ma warianty dla Linuksa i Windowsa. Lista tagów z Docker Hub z 2 września 2026 zawiera dla wydania 2026.2 między innymi te pozycje:

Tag	Dla czego	Uwagi
`2026.2` i `2026.2-linux`	Linux, amd64	podstawowy wariant z compose wyżej
`2026.2-linux-sudo`	Linux, Docker w Dockerze	wymagany przy `DOCKER_IN_DOCKER`
`2026.2-linux-arm64`	Linux na ARM	w Zielonej nieużywany
`2026.2-windowsservercore-2022`	Windows Server 2022	pasuje do hosta z kompilacją 20348
`2026.2-nanoserver-2022`	Windows Server 2022, odchudzony	bez PowerShella
`2026.2-windowsservercore-1809` i `2026.2-nanoserver-1809`	Windows Server 2019	host z kompilacją 17763
`2026.2-windowsservercore`	wariant bez wersji w nazwie	nie wiemy z samej nazwy, na którą wersję Windows wskazuje, więc w produkcji wolimy tag z numerem

Tagu z wersją 2025 dla Windows Server 2025 na tej liście nie było. Przy hoście z Windows Server 2025 sprawdzamy w tabeli zgodności Microsoftu, czy obraz dla 2022 uruchomimy z izolacją Hyper-V (`--isolation=hyperv`).

> **Pojęcie: izolacja procesowa i izolacja Hyper-V.** Kontener Windows może współdzielić jądro z hostem (izolacja procesowa, szybka, ale wymaga zgodnej kompilacji systemu) albo dostać własne jądro w lekkiej maszynie wirtualnej (izolacja Hyper-V, wolniejsza, ale toleruje inną wersję systemu w obrazie). Komunikat błędu `The operating system of the container does not match the operating system of the host` oznacza

> niezgodność wersji przy izolacji procesowej.

Linux i Windows: co jest inne

Element	Linux	Windows
Katalog tożsamości agenta	`/data/teamcity_agent/conf`	`C:\BuildAgent\conf`
Gniazdo Dockera hosta	`/var/run/docker.sock`	inny mechanizm, kroki z Dockerem na Windows sprawdzamy w dokumentacji
Docker w Dockerze	tak, tag `-linux-sudo`	nie dotyczy
Karta graficzna	NVIDIA Container Toolkit	osobny temat, w Zielonej agent z kartą działa na Linuksie
Zmienne `OWN_ADDRESS` i `OWN_PORT`	tak	nie dotyczy
Wolumen `work`	zalecany, zgodny ze ścieżką hosta przy gnieździe	nie montujemy, bo psuje Git
Uprawnienia katalogów	zwykle root w kontenerze	`Authenticated Users` z pełną kontrolą na katalogach wolumenów
Wymaganie w builde	`teamcity.agent.jvm.os.name` zaczyna się od `Linux`	`teamcity.agent.jvm.os.name` zaczyna się od `Windows`

Parametry `env.` agent zgłasza serwerowi sam, z zmiennych środowiskowych procesu. Dlatego parametry zależne od hosta (na przykład `GPU_VRAM_GB` albo ścieżka do SDK) ustawiamy w `environment:` pliku Compose danego hosta, a nie w `buildAgent.properties`, który jedzie razem z agentem. Gdy agent przenosi się na maszynę bez karty, z nowego pliku Compose znika parametr, a wymaganie buildów samo przestaje do niego pasować.

Gniazdo Dockera albo Docker w Dockerze

Krok budowania obrazu wymaga Dockera, więc agent w kontenerze musi go skądś wziąć. Mamy dwie drogi i obie niosą ryzyko.

Wariant	Jak działa	Ryzyko	Kiedy

Gniazdo hosta (`/var/run/docker.sock`)	agent wydaje polecenia demonowi Dockera hosta	dostęp do gniazda to praktycznie root na hoście, build może zatrzymać lub odczytać dowolny kontener	agent tylko dla zaufanych projektów, host bez cudzych danych
Docker w Dockerze (`privileged`)	agent ma własnego demona w kontenerze	tryb uprzywilejowany też daje szerokie prawa na hoście	gdy nie chcemy mieszać obrazów agenta z obrazami hosta

Na `gpu01` stoi Ollama z modelami w `/modele/ollama`, więc build z dostępem do gniazda mógłby je odczytać albo usunąć. Dlatego pulę z takim agentem przypisujemy wyłącznie do projektów zespołu Oli, nie trzymamy na hoście sekretów i nie dajemy agentowi więcej niż trzeba. Ryzyko gniazda opisujemy też na stronach Portainer: Docker w przeglądarce i Gitea Actions: runner i pierwszy workflow.

Licencja: liczą się autoryzowani agenci, nie maszyny

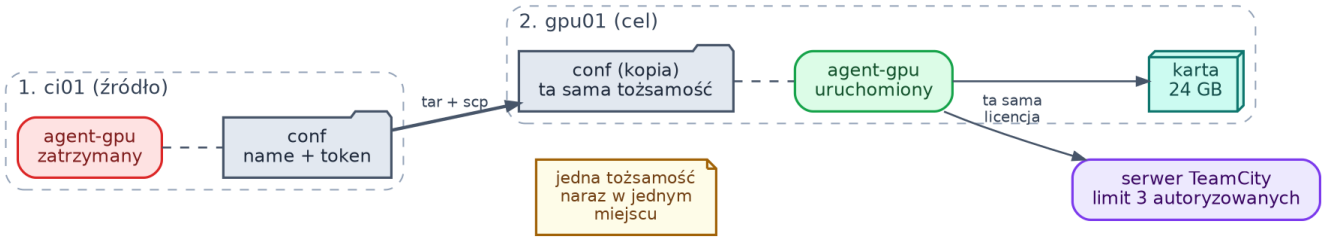
Darmowa edycja Professional ma trzy agenty, a kolejne wymagają dokupienia licencji agenta. Dokumentacja JetBrains mówi wprost, że licencje agentów nie są przypisane do konkretnych agentów, tylko ograniczają liczbę agentów autoryzowanych, bez względu na ich pochodzenie. Przekroczenie limitu kończy się ostrzeżeniem w interfejsie i wstrzymaniem nowych buildów. Z tego wynika kilka konsekwencji, które sprawdziliśmy na tej stronie w dokumentacji (3 października 2026):

1. Licencję zajmuje autoryzacja, nie połączenie. Agent wyłączony, ale wciąż autoryzowany, nadal wlicza się do limitu.
2. Cofnięcie autoryzacji zwalnia miejsce dla innej maszyny. Dokumentacja opisuje to jako możliwość rotowania licencji między maszynami.
3. Licencja nie może służyć kilku serwerom naraz, ale agenci jednego serwera mogą stać na dowolnej liczbie hostów.
4. Przeniesienie agenta z tą samą tożsamością nie dodaje nowej autoryzacji, więc nie zużywa drugiego miejsca.

Dla porównania, TeamCity Cloud liczy inaczej (committerów), ale Zielona używa wersji On-Premises, opisanej na stronie Plan wdrożenia TeamCity w Zielonej.

Przenoszenie agenta: ta sama tożsamość na innym hoście

Tożsamość agenta to plik `conf/buildAgent.properties` z `name` i `authorizationToken`. Token jest generowany przy pierwszym połączeniu i zapisywany z powrotem do pliku, więc plik musi zostać zapisywalny dla agenta. Serwer rozpoznaje agenta po tym tokenie, a licencję liczy dla autoryzowanego agenta. Skoro wolumen `conf` jedzie razem z agentem, agent po starcie na innym hoście jest dla serwera tym samym agentem.



Przeniesienie agenta agent-gpu z ci01 na gpu01: zatrzymanie, kopia katalogu conf z nazwą i tokenem, start na nowym hoście, ta sama licencja (kliknięcie otwiera pełny rozmiar)

Są trzy sposoby przeniesienia, różniące się tym, co zostaje po drodze.

Sposób	Co robimy	Licencja	Na co uważamy
Kopia wolumenu `conf`	zatrzymanie, `tar`, `scp`, start na nowym hoście	bez zmian	usunąć stary katalog `conf`, nigdy nie uruchamiać obu naraz
Ten sam wolumen na dysku sieciowym	wolumen z dysku sieciowego montowany raz na jednym hoście	bez zmian	zawsze tylko jeden host z zamontowanym wolumenem do zapisu, sprawdzamy blokady plików w dokumentacji systemu plików
Cofnięcie autoryzacji i ponowna autoryzacja	`Unauthorize`, start z pustym `conf`, `Authorize`	zwolnienie i zajęcie miejsca	nowa tożsamość: od nowa pula, parametry, stan; starego agenta usuwamy z listy

Wariant z ponowną autoryzacją jest czystszy, gdy host źródłowy przestaje istnieć albo gdy nie ufamy skopiowanemu tokenowi. W tym wariancie stary, odłączony agent nadal jest autoryzowany, jeśli go nie cofniemy, i dalej zajmuje miejsce w limicie.

Pule, wymagania i parametry po przeprowadzce

Pula jest cechą agenta po stronie serwera, więc przy tej samej tożsamości agent zostaje w swojej puli. Przy nowej tożsamości trafia do puli `Default` i przypisujemy go ponownie. Wymagania buildów (Agent Requirements) sprawdzają parametry, które agent zgłasza po starcie. Dlatego po przeniesieniu sprawdzamy w konfiguracji budowania zakładkę Compatible Agents: jeśli agent na nowym hoście ma inny system albo brak karty, build nie trafi do niego, i tak ma być.

Dwa agenty z tym samym tokenem

Dokumentacja konfiguracji agenta nie opisuje wprost, co robi serwer, gdy dwa procesy zgłaszają się z tym samym tokenem. Wątki wsparcia JetBrains pokazują objawy: w logu agenta pojawia się komunikat, że inny agent o tej nazwie jest już zarejestrowany, w logu serwera wzmianka o tym, że agent z tym samym tokenem właśnie się rejestruje, a sklonowane agenty gubią autoryzację albo dostają niezgodny token. Praktyczny wniosek jest prosty: jeden token to jeden działający proces.

> ⚠ Nie traktujemy tego jako sposobu na darmowy czwarty agent. Dwa procesy z jedną tożsamością nie dają dwóch agentów, tylko przepychankę o rejestrację, w której builds mogą się przerywać. Przy klonowaniu obrazu albo katalogu agenta zawsze usuwamy linię `authorizationToken` i zmieniamy `name`, a zachowanie na naszej wersji serwera sprawdzamy najpierw w laboratorium.

Częste błędy

Objaw	Przyczyna	Co robimy
Po przeniesieniu agent pojawia się jako nowy w `Unauthorized`, a stary wisi odłączony	katalog `conf` nie został skopiowany albo kontener startuje z pustym wolumenem	sprawdzamy ścieżkę wolumenu w Compose, kopiujemy `conf` i startujemy ponownie, a nowego agenta bez potrzeby nie autoryzujemy

Objaw	Przyczyna	Co robimy
`Authorize` zwraca błąd o limicie licencji	stary agent po przeniesieniu nadal jest autoryzowany	cofamy autoryzację starego agenta (`Unauthorize`), potem autoryzujemy nowego
Agent łączy się i rozłącza na przemian	dwa kontenery z tą samą tożsamością, na starym i nowym hoście	zatrzymujemy kontener na starym hoście i usuwamy tam `conf`
Kontener Windows nie startuje z błędem `0xc0370101`	wersja hosta nie zgadza się z obrazem przy izolacji procesowej	wybieramy obraz zgodny z hostem albo dodajemy `--isolation=hyperv`
Git na agencie Windows kończy się błędem `/ContainerMappedDirectories`	katalog `work` zamontowany jako wolumen	usuwamy ten wolumen z polecenia `docker run`
Agent Windows nie zapisuje do `conf`	brak uprawnień kontenera do katalogu na hoście	nadajemy `Authenticated Users` pełną kontrolę (fiszka z `icacls`)
Build z kartą kończy się brakiem pamięci	Ollama trzyma model w pamięci karty (`OLLAMA_KEEP_ALIVE`)	skracamy czas trzymania modelu albo planujemy buildy poza godzinami czatu
Błąd certyfikatu przy `SERVER_URL` z `https`	agent nie ufa wewnętrznemu CA-Zielona	dodajemy CA do magazynu zaufania agenta według dokumentacji JetBrains albo zostajemy przy HTTP w VLAN 10
Kontener nie widzi karty	brak NVIDIA Container Toolkit albo restartu Dockera po konfiguracji	powtarzamy `docker run --rm --gpus all ... nvidia-smi`

□□ Ćwiczenia

1. Mamy trzy autoryzowane agenty i uruchamiamy czwartego na nowej maszynie. Co się stanie przy próbie autoryzacji i jak zwolnić miejsce bez dokupowania licencji?

Odpowiedź

Autoryzacja zakończy się komunikatem o przekroczonym limicie licencji (jak na stronie o agentach), a według dokumentacji przekroczenie limitu wstrzymuje nowe buildy. Miejsce zwalniamy, cofając autoryzację agenta, którego nie potrzebujemy, albo przenosimy istniejącego agenta zamiast tworzyć nowego. Licencja liczy agentów autoryzowanych, nie maszyny.

1. Dlaczego przed skopiowaniem katalogu `conf` zatrzymujemy agenta na starym hoście, a po przeniesieniu usuwamy tam ten katalog?

Odpowiedź

Zatrzymanie daje spójny plik z tożsamością i gwarantuje, że dwa procesy nie użyją jednego tokenu naraz. Usunięcie starej kopii chroni przed przypadkowym uruchomieniem drugiego agenta z tą samą tożsamością (na przykład przez `docker compose up -d` wykonane z rozpędu) i przed pozostawieniem tokenu, który działa jak hasło agenta.

1. Agent wrócił z `gpu01` na `ci01` po kilku tygodniach, ale stary katalog `conf` na `ci01` usunęliśmy. Jakie masz dwie możliwości przywrócenia agenta?

Odpowiedź

Pierwsza: skopiować `conf` z `gpu01` z powrotem na `ci01`, zatrzymawszy wcześniej agenta na `gpu01`. Druga: cofnąć autoryzację agenta, wystartować kontener na `ci01` z pustym `conf` i autoryzować go jako nowego, po czym ponownie przypisać pulę i parametry. W obu przypadkach nie wolno trzymać dwóch działających agentów z jednym tokenem.

1. Parametr `GPU_VRAM_GB` ustawiamy w `environment:` pliku Compose, a nie w `buildAgent.properties`. Dlaczego?

Odpowiedź

`buildAgent.properties` jedzie razem z agentem na każdy nowy host, więc parametr o karcie pojechałby na maszynę bez karty i buildy z wymaganiem `exists env.GPU_VRAM_GB` trafiałyby tam błędnie. Plik Compose należy do hosta i zawsze opisuje jego rzeczywisty sprzęt.

1. Który wariant dostępu agenta do Dockera wybieramy dla agenta na `gpu01` i jak ograniczamy ryzyko?

Odpowiedź

Oba warianty dają szerokie prawa na hoście, więc wybór zależy od zaufania do projektów. Przy gnieździe hosta agent może dotknąć Ollamy i modeli, więc przypisujemy go tylko do puli z projektami zespołu Oli, nie trzymamy sekretów na hoście i nie uruchamiamy buildów z cudzych forków. Przy Dockerze w Dockerze (`privileged`) izolujemy obrazy, ale uprawnienia dalej są szerokie.

Powiązane: Agenci TeamCity: dodanie, autoryzacja, pule · Instalacja TeamCity w Dockerze · Plan wdrożenia TeamCity w Zielonej · Modele lokalne: AI na własnym sprzęcie · Hasła i sekrety poza repozytorium: .env, Docker secrets i Vaultwarden

Źródła

- TeamCity, Licensing Policy (liczenie autoryzowanych agentów, limity edycji Professional): https://www.jetbrains.com/help/teamcity/licensing-policy.html
- TeamCity, Install and Start TeamCity Agents (autoryzacja, odautoryzowanie zwalnia licencję): https://www.jetbrains.com/help/teamcity/install-and-start-teamcity-agents.html
- TeamCity, Build Agent Configuration (`name`, `authorizationToken`, `serverUrl`): https://www.jetbrains.com/help/teamcity/build-agent-configuration.html
- TeamCity, Agent Requirements (parametry `env.`, `system.`, `teamcity.agent.jvm.os.name`): https://www.jetbrains.com/help/teamcity/agent-requirements.html
- TeamCity, Configuring Agent Pools: https://www.jetbrains.com/help/teamcity/configuring-agent-pools.html
- TeamCity, Known Issues, Windows Docker containers: https://www.jetbrains.com/help/teamcity/known-issues.html
- JetBrains, README obrazu `teamcity-agent` (zmienne, wolumeny, Docker w Dockerze): https://github.com/JetBrains/teamcity-docker-images/blob/master/dockerhub/teamcity-agent/README.md
- Docker Hub, tagi `jetbrains/teamcity-agent` (sprawdzone 3 października 2026): https://hub.docker.com/r/jetbrains/teamcity-agent/tags
- Wątki wsparcia JetBrains o zdublowanych tokenach i klonowanych agentach: https://teamcity-support.jetbrains.com/hc/en-us/community/posts/205437870 oraz https://teamcity-support.jetbrains.com/hc/en-us/community/posts/206815685
- Microsoft, zgodność wersji kontenerów Windows: https://learn.microsoft.com/en-us/virtualization/windowscontainers/deploy-containers/version-compatibility
- Microsoft, obrazy bazowe kontenerów Windows: https://learn.microsoft.com/en-us/virtualization/windowscontainers/manage-containers/container-base-images
- Docker, dostęp do GPU w Compose: https://docs.docker.com/compose/how-tos/gpu-support/
- NVIDIA, instalacja Container Toolkit: https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/install-guide.html