Kontroler i agenci

Jak zbudowany jest Jenkins, jak go instalujemy i jak podłączamy agentów.

- Architektura Jenkinsa: kontroler, agenci, workspace
- Instalacja i aktualizacja Jenkinsa
- Agenci: dodanie, diagnostyka, utrzymanie
- Instalacja Jenkinsa w Dockerze

Architektura Jenkinsa: kontroler, agenci, workspace

Poziom: podstawy **Dotyczy: Jenkins LTS 2.x** **Czas: ~20 min**

Zanim dodamy pierwszego agenta albo napiszemy pierwszy Jenkinsfile, warto rozumieć, które procesy gdzie żyją i dlaczego build wykonuje się gdzie indziej niż interfejs, który go zlecił.

Skróty użyte na tej stronie (6)

Skrót	Co znaczy
HTTP	Hypertext Transfer Protocol: protokół stron WWW
JDK	Java Development Kit: zestaw do uruchamiania i budowania programów w Javie
JNLP	metoda podłączenia agenta Jenkinsa, w której to agent inicjuje połączenie do kontrolera
LTS	Long Term Support: wersja z długim wsparciem poprawkami
SSH	Secure Shell: szyfrowane zdalne logowanie do serwera
WMS	Warehouse Management System: system zarządzania magazynem

Z życia wzięte

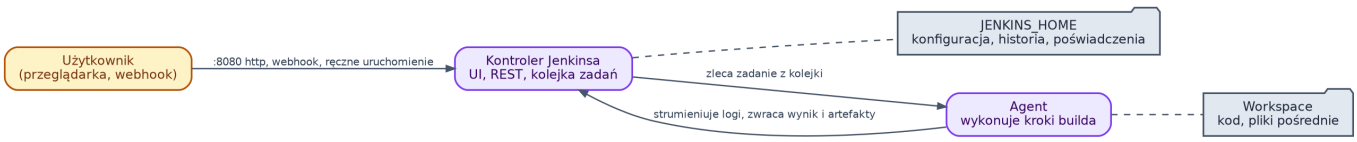
Z ŻYCIA WZIĘTE

W piątek o 16:00 Tomek zgłasza, że pipeline weryfikujący paczkę WMS wisi w kolejce od dwudziestu minut, mimo że w Jenkinsie na `ci01` widać same zielone agenty. Okazuje się, że etykieta `linux-docker` w Jenkinsfile ma literówkę, więc kontroler nie znajduje pasującego agenta i czeka w nieskończoność. Po poprawieniu etykiety build startuje natychmiast na pierwszym wolnym agencie.

Czym jest kontroler

Pojęcie: webhook. Automatyczne powiadomienie, które jeden system wysyła do drugiego przez HTTP zaraz po tym, jak coś się wydarzyło, zamiast czekać, aż ktoś sam to sprawdzi. W Zielonej Gitea wysyła taki webhook do Jenkinsa na `ci01` przy każdym pushu do repozytorium, co uruchamia odpowiedni pipeline bez ręcznego klikania.

Pojęcie: kontroler. Kontroler (w starszej terminologii i części literatury nadal spotykane „master") to główny proces Jenkinsa — jedna aplikacja Java uruchomiona w kontenerze serwletów wbudowanym w dystrybucję. Kontroler serwuje interfejs webowy, trzyma całą konfigurację (zadania, poświadczenia, uprawnienia, konfigurację pluginów), przyjmuje zdarzenia wyzwalające buildy (webhook z systemu kontroli wersji, harmonogram, ręczne uruchomienie) i utrzymuje kolejkę zadań czekających na wolnego agenta. To, czego kontroler **nie robi** w typowej konfiguracji, to samo wykonywanie kroków builda — od tego są agenci.



Kontroler przyjmuje żądania i zleca zadania agentowi, trzymając konfigurację w JENKINS_HOME, a agent pracuje w swoim workspace (kliknięcie otwiera pełny rozmiar)

Kontroler nasłuchuje na dwóch portach, które mają zupełnie różne przeznaczenie:

Port	Przeznaczenie
`8080`	interfejs HTTP — UI, REST API, webhooki z systemów kontroli wersji
`50000`	port TCP dla agentów podłączających się metodą inbound (JNLP)

Port `50000` ma sens tylko wtedy, gdy w ogóle korzystamy z agentów inbound — przy wyłącznie agentach SSH można go zamknąć na firewallu.

JENKINS_HOME

Pojęcie: JENKINS_HOME. Katalog, w którym kontroler trzyma cały swój stan — w instalacji z pakietu to zwykle `/var/lib/jenkins`, w oficjalnym obrazie kontenerowym `/var/jenkins_home`. Utrata tego katalogu bez kopii zapasowej oznacza utratę historii buildów, konfiguracji zadań i poświadczeń — to jedyny katalog, który naprawdę trzeba kopiować przed

jakąkolwiek zmianą.

Ścieżka w JENKINS_HOME	Zawartość
`config.xml`	globalna konfiguracja kontrolera
`jobs/NazwaZadania/config.xml`	definicja pojedynczego zadania
`jobs/NazwaZadania/builds/`	historia wykonań tego zadania, logi, artefakty
`plugins/`	zainstalowane wtyczki (pliki `.jpi`/`.hpi`)
`secrets/`	klucz szyfrujący poświadczenia (`master.key`, `hudson.util.Secret`)
`users/`	konta lokalne, gdy używamy wbudowanej bazy użytkowników
`nodes/NazwaAgenta/config.xml`	konfiguracja pojedynczego agenta
`credentials.xml`	magazyn poświadczeń — zaszyfrowany kluczem z `secrets/`
`.ssh/known_hosts`	odciski kluczy hostów, z którymi kontroler łączył się po SSH (agenci)

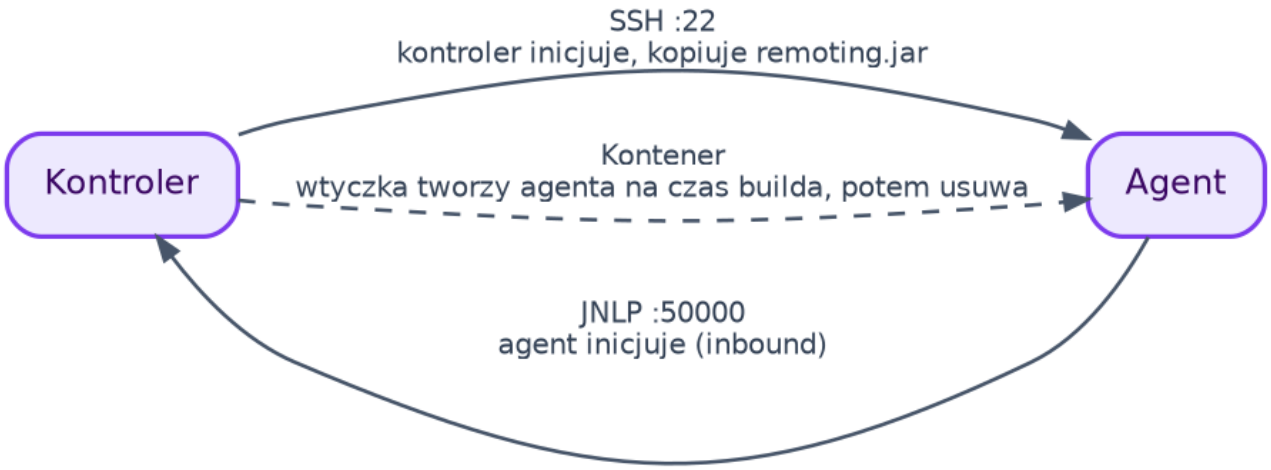
Czym jest agent

Pojęcie: agent. Agent (starsza nazwa „slave", w interfejsie Jenkinsa i dziś spotykana jako „node") to osobny proces Java, uruchomiony na maszynie innej niż kontroler (fizycznej, wirtualnej albo w kontenerze), który wykonuje faktyczne kroki builda — checkout kodu, kompilację, testy, budowanie obrazu. Kontroler zleca agentowi zadanie i czeka na wynik; sam agent nie ma własnego interfejsu webowego ani nie widzi pełnej konfiguracji Jenkinsa, tylko to, co potrzebuje do wykonania przydzielonych kroków.

Rozdzielenie kontrolera od agentów daje trzy rzeczy naraz: kontroler zostaje lekki i stabilny niezależnie od tego, ile i jak ciężkich buildów się aktualnie wykonuje, buildy różnych zespołów mogą mieć różne środowiska (inny JDK, inny system operacyjny) bez konfliktu, a awaria jednego agenta nie zabiera ze sobą kontrolera.

Sposoby podłączenia agenta

Jenkins łączy kontroler z agentem na trzy sposoby, różniące się tym, kto inicjuje połączenie sieciowe.



Trzy sposoby połączenia kontrolera z agentem, różniące się tym, która strona inicjuje ruch sieciowy (kliknięcie otwiera pełny rozmiar)

SSH. Kontroler łączy się do agenta po SSH, kopiuje na niego bibliotekę `remoting.jar` i uruchamia ją jako proces potomny. To metoda wymagająca, żeby kontroler miał dostęp sieciowy do agenta na porcie 22 oraz poświadczenia SSH zapisane w magazynie poświadczeń Jenkinsa — najwygodniejsza, gdy agenci mają stałe adresy i firewall pozwala na ruch wychodzący z kontrolera. Diagnostyka tej metody i najczęstsze błędy stoją na stronie Agenci: dodanie, diagnostyka, utrzymanie.

JNLP / inbound. Odwrotny kierunek — to agent inicjuje połączenie do kontrolera, na port `50000`. Ma to sens, gdy agent siedzi za NAT-em albo restrykcyjnym firewallem, przez który kontroler nie ma jak zainicjować połączenia wychodzącego, a agent owszem — może wybić ruch wychodzący do kontrolera.

Kontener. Agent nie jest maszyną istniejącą na stałe, tylko efemerycznym kontenerem tworzonym na potrzeby pojedynczego builda przez odpowiednią wtyczkę (np. wtyczkę Kubernetes albo Docker), i usuwanym zaraz po jego zakończeniu. Eliminuje to problem „zaśmieconego" agenta — każdy build startuje w czystym środowisku zdefiniowanym obrazem kontenera, kosztem czasu potrzebnego na start kontenera przed każdym buildem.

Etykiety agentów i przypisywanie zadań

> **Pojęcie: etykieta (label).** Dowolny tekstowy tag przypisany agentowi w jego konfiguracji, w polu „Labels" — jeden agent może mieć kilka etykiet oddzielonych spacją. Zadanie deklaruje, na jakim agencie chce się wykonać, wskazując etykietę zamiast konkretnej nazwy node'a: w pipeline deklaratywnym przez `agent { label 'NazwaEtykiety' }`, w zadaniu typu

> freestyle przez pole „Restrict where this project can be run". Dzięki temu zadanie nie jest przywiązane do jednej konkretnej maszyny — kontroler sam wybiera dowolnego wolnego agenta z pasującą etykietą, a dodanie kolejnego agenta o tej samej etykiecie automatycznie zwiększa przepustowość bez zmiany definicji zadania.

Workspace i jego czyszczenie

> **Pojęcie: workspace.** Katalog roboczy, w którym agent wykonuje konkretny build konkretnego zadania — tam trafia checkout repozytorium, tam kompilator zostawia pliki pośrednie, stamtąd archiwizujemy artefakty na koniec. Ścieżka to zwykle `SciezkaDoPliku/workspace/NazwaZadania`, licząc od katalogu domowego agenta (`remote root directory` ustawiony w konfiguracji node'a). Workspace **nie jest** czyszczony automatycznie po każdym buildzie — kolejne uruchomienie tego samego zadania na tym samym agencie zwykle dostaje ten sam katalog, żeby skorzystać z niewyczyszczonego stanu (np. z zależności pobranych przez menedżer pakietów).

To ma swoją cenę: workspace rośnie w nieskończoność, jeśli nic go nie czyści. Trzy mechanizmy odpowiadają za utrzymanie go w ryzach — krok `cleanWs()` na końcu pipeline'u kasujący cały katalog, opcja node'a „Disable deferred wipeout on this node" (widoczna w konfiguracji agenta — domyślnie Jenkins i tak czyści workspace przy usuwaniu node'a, ta opcja to wyłącza) oraz po prostu ręczne `rm -rf` z poziomu skryptu, gdy potrzebujemy pełnej kontroli nad momentem czyszczenia.

Jak to działa

Kolejność zdarzeń od kliknięcia „Build Now" do zakończonego builda wygląda tak: kontroler wstawia zadanie do kolejki, przegląda dostępnych agentów i wybiera pierwszego wolnego, którego etykiety pasują do wymagań zadania (albo executor wbudowany w sam kontroler, jeśli zadanie nie wymaga konkretnej etykiety, a administrator nie wyłączył wykonywania buildów na kontrolerze), przekazuje agentowi definicję kroków, agent tworzy lub reużywa workspace, wykonuje kolejne kroki lokalnie na swoim dysku i strumieniuje logi z powrotem do kontrolera w czasie rzeczywistym. Po zakończeniu agent zwraca kod wyjścia i artefakty do zarchiwizowania, a kontroler zapisuje wynik w historii zadania pod `JENKINS_HOME/jobs/NazwaZadania/builds/`.

Częste błędy

Utożsamianie kontrolera z Jenkinsem w ogóle i uruchamianie na nim ciężkich buildów. Kontroler ma domyślnie własne executory i teoretycznie może wykonywać buildy sam, ale każdy

taki build konkuruje o zasoby z UI i harmonogramem całej instalacji — w środowisku z więcej niż kilkoma zadaniami warto ustawić liczbę executorów kontrolera na `0` i zmusić wszystko do agentów.

Zakładanie, że workspace jest czyszczony między buildami. Nowa osoba bywa zaskoczona, że plik z poprzedniego builda wciąż leży w workspace — to zamierzone zachowanie, nie błąd; jeśli build ma zależeć wyłącznie od bieżącego stanu repozytorium, trzeba to wymusić jawnie krokiem `cleanWs()`.

Mylenie etykiety z nazwą node'a. Wpisanie w `agent { label 'app01' }` konkretnej nazwy maszyny zamiast opisowej etykiety działa, dopóki ta jedna maszyna istnieje — ale traci całą elastyczność mechanizmu etykiet i wymaga edycji Jenkinsfile przy każdej zmianie infrastruktury.

□□ Ćwiczenia

> Zadania sprawdzające.

1. Wymieniamy, które z poniższych czynności wykonuje kontroler, a które agent: przyjęcie webhooka z repozytorium, checkout kodu, wysłanie e-maila o wyniku builda, kompilacja projektu.
2. Opisujemy, czym różni się metoda podłączenia agenta przez SSH od metody inbound (JNLP) — który koniec połączenia inicjuje ruch sieciowy w każdej z nich.
3. Dwa zadania mają wspólną etykietę `linux-docker` przypisaną do trzech agentów. Wyjaśniamy, co się stanie, jeśli oba zostaną uruchomione w tym samym momencie.
4. Wskazujemy, który katalog trzeba skopiować przed aktualizacją Jenkinsa, żeby móc wrócić do poprzedniego stanu, i uzasadniamy dlaczego akurat ten.

Instalacja i aktualizacja Jenkinsa

Typ: procedura **Dotyczy: Jenkins LTS, RHEL/Rocky Linux 9 (repozytorium pkg.jenkins.io)**

Czas: ~30 min instalacja, ~15 min aktualizacja

Efekt: kontroler Jenkinsa zainstalowany z oficjalnego repozytorium, uruchomiony jako usługa systemd, odblokowany i gotowy do dodania pierwszych wtyczek i agentów.

Skróty użyte na tej stronie (9)

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
GPG	GNU Privacy Guard: program do podpisywania i szyfrowania plików oraz sprawdzania podpisów paczek
HTTP	Hypertext Transfer Protocol: protokół stron WWW
HTTPS	HTTP zabezpieczony szyfrowaniem TLS
JDK	Java Development Kit: zestaw do uruchamiania i budowania programów w Javie
LTS	Long Term Support: wersja z długim wsparciem poprawkami
RHEL	Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa
TLS	Transport Layer Security: szyfrowanie połączeń, np. w HTTPS

Zanim zaczniemy

Z ŻYCIA WZIĘTE

Tomek instaluje aktualizację Jenkinsa w środę rano na `ci01`, bez wcześniejszej kopii `JENKINS_HOME`, bo to „tylko mała łatka". Po restarcie usługa nie startuje, bo nowa wersja wymaga JDK 21, którego na serwerze jeszcze nie ma. Powrót do poprzedniej wersji zajmuje pół dnia, bo kopii `JENKINS_HOME` po prostu nie było.

Instalację wykonujemy z CLI bash jako user z uprawnieniami sudo/root. Serwer musi mieć zainstalowaną Javę w wersji wymaganej przez aktualną linię LTS oraz rozwiązywać nazwy DNS, żeby pobrać paczkę z repozytorium.

Wymaganie	Uwaga
JDK 17 lub 21	Jenkins LTS od pewnego czasu nie startuje na starszych wersjach — sprawdzamy wymaganą wersję w release notes przed instalacją
Ruch wychodzący do `pkg.jenkins.io`	repozytorium pakietów, potrzebne też przy każdej aktualizacji
Otwarty port `8080` do klienta (lub reverse proxy)	UI kontrolera
Otwarty port `50000`, jeśli planujemy agentów inbound	patrz Architektura Jenkinsa: kontroler, agenci, workspace

Uważaj: przed każdą aktualizacją wykonujemy kopię `JENKINS_HOME` — to jedyny katalog, bez którego powrót do poprzedniego stanu jest niemożliwy.

Przebieg

1. Instalacja Javy

Sprawdzamy, czy Java jest już zainstalowana:

```
java -version
```

Jeśli brak — instalujemy JDK z repozytoriów dystrybucji:

```
sudo dnf install -y java-17-openjdk
```

2. Dodanie repozytorium Jenkinsa

Pobieramy plik repozytorium dla linii LTS:

```
sudo curl -o /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo
```

Importujemy klucz GPG repozytorium:

```
sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key
```

3. Instalacja pakietu

```
sudo dnf install -y jenkins
```

Pakiet instaluje usługę systemd, użytkownika systemowego `jenkins` oraz katalog `JENKINS_HOME` pod `/var/lib/jenkins`.

4. Uruchomienie usługi

```
sudo systemctl enable --now jenkins
```

Sprawdzamy status:

```
sudo systemctl status jenkins
```

Przykład:

```
● jenkins.service - Jenkins Continuous Integration Server
   Loaded: loaded (/usr/lib/systemd/system/jenkins.service; enabled; vendor preset: disabled)
   Active: active (running) since Tue 2026-09-01 09:12:04 UTC; 8s ago
 Main PID: 18422 (java)
   Status: "Jenkins is fully up and running"
    Tasks: 45 (limit: 23684)
   Memory: 412.1M
```

5. Odblokowanie instalacji — initialAdminPassword

Przy pierwszym uruchomieniu Jenkins generuje jednorazowe hasło administratora i zapisuje je do pliku wewnątrz `JENKINS_HOME`:

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

Przykład:

```
3f8a1c9d2e4b47f0a6c1d8e5b3a29f71
```

To hasło wklejamy na ekranie „Unlock Jenkins" pod adresem `http://NazwaSerwera:8080`.

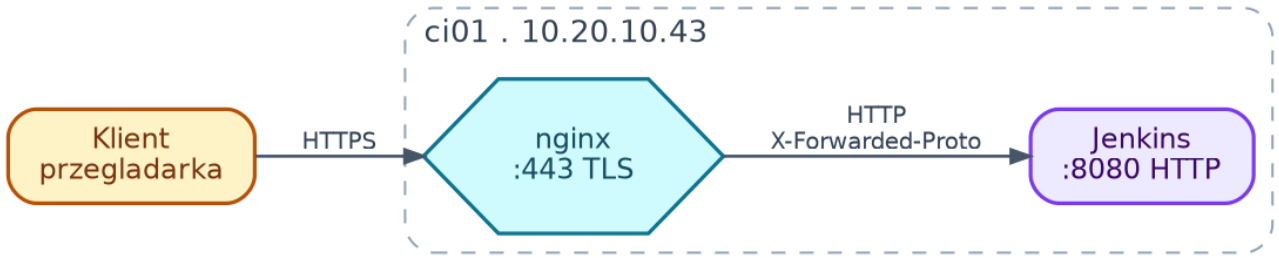
6. Wtyczki podstawowe

Kreator po odblokowaniu proponuje dwie ścieżki — instalujemy zestaw sugerowanych wtyczek (Suggested Plugins), który obejmuje integrację z Gitem, obsługę pipeline'ów i podstawowe raportowanie. Zestaw da się rozszerzyć później z poziomu Manage Jenkins → Manage Plugins, bez ponownej instalacji.

7. Reverse proxy

> **Pojęcie: reverse proxy.** Serwer pośredniczący, który stoi przed aplikacją i przyjmuje ruch z zewnątrz zamiast niej, najczęściej po to, żeby obsłużyć szyfrowanie TLS albo wspólny adres dla kilku usług naraz. W Zielonej przed Jenkinsem na `ci01` stoi właśnie taki reverse proxy, nginx: to on odbiera połączenia HTTPS, a do samego Jenkinsa na porcie `8080` puszcza już zwykły ruch HTTP.

Wystawianie Jenkinsa bezpośrednio na porcie `8080` na świat zewnętrzny jest rzadko pożądane — stawiamy przed nim reverse proxy terminujące TLS.



Nginx na ci01 przyjmuje ruch HTTPS od klienta i przekazuje go zwykłym HTTP do Jenkinsa na porcie 8080 (kliknięcie otwiera pełny rozmiar)

Fragment konfiguracji nginksa:

```
server {
    listen 443 ssl;
    server_name jenkins.zielona.example;

    location / {
        proxy_pass http://127.0.0.1:8080;
```

```
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_redirect http:// https://;
    }
}
```

Bez nagłówka `X-Forwarded-Proto` Jenkins potrafi generować w mailach i webhookach linki z `http://` zamiast `https://`.

Weryfikacja

Sprawdzamy wersję zainstalowanego Jenkinsa i porównujemy z oczekiwaną linią LTS:

```
sudo systemctl show jenkins -p Version 2>/dev/null; rpm -q jenkins
```

Przykład:

```
jenkins-2.462.3-1.1.noarch
```

Sprawdzamy, że UI odpowiada lokalnie, zanim obwinimy reverse proxy o problem:

```
curl -s -o /dev/null -w '%{http_code}\n' http://127.0.0.1:8080/login
```

Przykład:

```
200
```

Wycofanie zmian

Aktualizacja pakietu (RPM)

Przed aktualizacją kopiujemy `JENKINS_HOME`:

```
sudo systemctl stop jenkins
sudo tar czf /SciezkaDoPliku/jenkins_home_backup.tar.gz -C /var/lib/jenkins .
```

```
sudo systemctl start jenkins
```

Aktualizacja samego pakietu:

```
sudo dnf update -y jenkins
sudo systemctl restart jenkins
```

Aktualizacja przez podmianę pliku war

Gdy Jenkins jest uruchamiany bezpośrednio z pliku `.war` (bez pakietu systemowego), aktualizacja polega na podmianie tego pliku i restarcie procesu:

```
sudo systemctl stop jenkins
sudo cp /usr/share/java/jenkins.war /SciezkaDoPliku/jenkins.war.old
sudo cp /SciezkaDoPliku/jenkins.war /usr/share/java/jenkins.war
sudo systemctl start jenkins
```

Wycofanie do poprzedniej wersji

Jeśli po aktualizacji coś nie działa, wracamy do poprzedniej paczki z cache dnf (o ile nie został wyczyszczony) albo przywracamy kopię `JENKINS_HOME` wykonaną przed aktualizacją:

```
sudo systemctl stop jenkins
sudo dnf downgrade -y jenkins
sudo rm -rf /var/lib/jenkins/*
sudo tar xzf /SciezkaDoPliku/jenkins_home_backup.tar.gz -C /var/lib/jenkins
sudo systemctl start jenkins
```

> Wycofanie `JENKINS_HOME` do starszej wersji ma sens tylko razem z wycofaniem samego pakietu — nowszy `JENKINS_HOME` bywa niekompatybilny wstecz ze starszym Jenkinsem, zwłaszcza po migracjach formatu konfiguracji.

Gdy coś nie zadziała

Objaw	Przyczyna	Naprawa

Usługa `jenkins` nie startuje po instalacji	zbyt stara/nowa wersja Javy	sprawdzamy `journalctl -u jenkins` pod kątem komunikatu o wymaganej wersji JDK
`initialAdminPassword` nie istnieje	Jenkins już wcześniej odblokowany, plik usuwany po pierwszej konfiguracji	pomijamy ten krok, logujemy się kontem administratora
UI wisi po aktualizacji na ekranie ładowania	niekompatybilna wtyczka ze starszą wersją zapisaną w konfiguracji	uruchamiamy w trybie bezpiecznym: `java -jar jenkins.war --enable-future-java` lub czasowo wyłączamy podejrzaną wtyczkę usuwając jej `.jpi` z `plugins/`
Po reverse proxy logowanie przekierowuje w pętli	brak nagłówka `X-Forwarded-Proto`	dokładamy nagłówek w konfiguracji proxy i ustawiamy w Jenkinsie właściwy adres URL w Manage Jenkins → System
Aktualizacja pakietu nie ruszyła Jenkinsa	manualnie edytowany plik jednostki systemd wskazuje starą ścieżkę do war	`systemctl daemon-reload` i ponowny restart usługi

Powiązane: Architektura Jenkinsa: kontroler, agenci, workspace · Utrzymanie i diagnostyka · Poświadczenia i bezpieczeństwo

Źródła: dokumentacja Jenkins — Installing Jenkins · `pkg.jenkins.io`

Agenci: dodanie, diagnostyka, utrzymanie

Poziom: operacyjny **Dotyczy: Jenkins LTS, agent SSH na RHEL/Rocky Linux**

Czas: ~40 min

Dodanie agenta to w praktyce dwie osobne roboty — przygotowanie maszyny (dysk, użytkownik, klucze SSH) i konfiguracja node'a w samym Jenkinsie. Ta strona pokazuje obie, razem z diagnostyką najczęstszych powodów, dla których agent nagle staje się offline.

Skróty użyte na tej stronie (7)

Skrót	Co znaczy
CLI	Command Line Interface: praca przez wpisywanie komend w terminalu
GB	Gigabyte: miliard bajtów, jednostka pojemności pamięci i plików, tu najczęściej rozmiar modelu albo pamięci karty graficznej
IP	Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection)
JDK	Java Development Kit: zestaw do uruchamiania i budowania programów w Javie
LTS	Long Term Support: wersja z długim wsparciem poprawkami
RHEL	Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa
SSH	Secure Shell: szyfrowane zdalne logowanie do serwera

Fiszka

Komendy na serwerze agenta wykonujemy z CLI bash z uprawnieniami root (sudo).

TYLKO ODCZYT

Sprawdzenie widocznych dysków: lsblk

```
lsblk
```

Sprawdzamy widoczne dyski.

lsblk = list block devices: pokaż dyski i partycje

ZMIENIA SYSTEM

Utworzenie wolumenu fizycznego: pvcreate

```
pvcreate /dev/sdX
```

(gdzie `/dev/sdX` to nazwa urządzenia blokowego przeznaczonego pod katalog agenta)

pvcreate = physical volume create: oddaj dysk pod LVM

ZMIENIA SYSTEM

Utworzenie grupy wolumenów: vgcreate

```
vgcreate vgJenkins_Agent /dev/sdX
```

Tworzymy grupę wolumenów.

vgcreate = volume group create: utwórz grupę wolumenów

ZMIENIA SYSTEM

Utworzenie wolumenu logicznego: lvcreate

```
lvcreate -l +100%FREE -n lvJenkins_Agent vgJenkins_Agent
```

Tworzymy wolumen logiczny na całej dostępnej przestrzeni tej grupy.

lvcreate = logical volume create: utwórz wolumen logiczny

OSTROŻNIE · NIEODWRACALNE

Założenie systemu plików: mkfs.ext4

```
mkfs.ext4 /dev/mapper/vgJenkins_Agent-lvJenkins_Agent
```

Zakładamy system plików.

mkfs.ext4 = od make filesystem: zakłada na partycji lub wolumenie system plików ext4, niszcząc dotychczasowe dane

TYLKO ODCZYT

Dodanie wpisu w /etc/fstab

```
/dev/mapper/vgJenkins_Agent-lvJenkins_Agent /home/jenkins_agent ext4 defaults 1 2
```

Dodajemy wpis w `/etc/fstab`.

ZMIENIA SYSTEM

Zamontowanie zgodnie z nowym wpisem w fstab: mount -a

```
mount -a
```

Montujemy wolumen zgodnie z nowym wpisem w fstab.

mount = podepnij system plików z urządzenia albo obrazu pod wskazany katalog

TYLKO ODCZYT

Sprawdzenie zamontowania wolumenu: df -h

```
df -h /home/jenkins_agent
```

Sprawdzamy, czy nowy mountpoint się pojawił.

Przykład:

```
Filesystem                                Size  Used Avail Use% Mounted on
/dev/mapper/vgJenkins_Agent-lvJenkins_Agent  9.8G   24K  9.3G   1% /home/jenkins_agent
```

df = disk free: ile miejsca wolnego na dyskach

ZMIENIA SYSTEM

Utworzenie użytkownika systemowego: useradd

```
useradd -d /home/jenkins_agent jenkins_agent
```

Zakładamy dedykowanego użytkownika systemowego, którego katalogiem domowym będzie właśnie zamontowany wolumen.

useradd = user add: tworzy nowe konto systemowe w Linuksie

ZMIENIA SYSTEM

Nadanie uprawnień do katalogu: chown

```
chown -R jenkins_agent:jenkins_agent /home/jenkins_agent
```

Nadajemy właściwe uprawnienia do katalogu.

chown = change owner: zmień właściciela

ZMIENIA SYSTEM

Ograniczenie dostępu do katalogu domowego: chmod

```
chmod 700 /home/jenkins_agent/
```

Ograniczamy dostęp do katalogu domowego wyłącznie dla właściciela.

chmod = change mode: zmień prawa dostępu

ZMIENIA SYSTEM

Instalacja gita: yum install

```
sudo yum install git
```

Instalujemy gita, potrzebnego do checkoutu repozytorium podczas buildów.

yum = Yellowdog Updater Modified: starszy menedżer pakietów RHEL

ZMIENIA SYSTEM

Dodanie użytkownika do grupy docker: usermod

```
usermod -aG docker jenkins_agent
```

Jeśli buildy na tym agencie mają budować obrazy kontenerów, dopisujemy użytkownika do grupy `docker`.

usermod = zmień ustawienia istniejącego konta systemowego, na przykład przynależność do grupy

TYLKO ODCZYT

Wejście do kontenera aplikacji Jenkins (wymiana kluczy SSH): docker exec

```
sudo docker exec -it jenkins_app /bin/bash
```

Połączenie wykonujemy z wnętrza kontenera aplikacji Jenkins, jeśli kontroler jest uruchomiony jako kontener.

SSH = Secure Shell: szyfrowane zdalne logowanie do serwera · **docker** = narzędzie do kontenerów

TYLKO ODCZYT

Połączenie SSH z serwerem agenta: ssh

```
ssh jenkins_agent@app01.ipa.zielona.example
```

Z poziomu tej powłoki łączymy się na docelowego użytkownika na serwerze agenta.

Przykład pierwszego połączenia:

```
The authenticity of host 'app01.ipa.zielona.example (10.20.10.41)' can't be established.
ED25519 key fingerprint is SHA256:mjt4fQzt+BapuKXEQqgPVWQXGnTheNfqd1nkQd94Ri18.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'app01.ipa.zielona.example,10.20.10.41' (ED25519) to the list
of known hosts.
```

SSH = Secure Shell: szyfrowane zdalne logowanie do serwera · **ssh** = Secure Shell: szyfrowane zdalne logowanie

TYLKO ODCZYT

Wejście do kontenera aplikacji Jenkins (naprawa known_hosts): docker exec

```
sudo docker exec -it jenkins_app /bin/bash
```

W ramach naprawy wchodzimy do kontenera aplikacji Jenkins.

docker = narzędzie do kontenerów

TYLKO ODCZYT

Usunięcie przeterminowanego wpisu z known_hosts: ssh-keygen -R

```
ssh-keygen -f "/var/jenkins_home/.ssh/known_hosts" -R "app01.ipa.zielona.example"
```

Usuwamy przeterminowany wpis ze znanych hostów, dokładnie komendą podpowiedzianą w logu.

ssh-keygen = wygeneruj parę kluczy SSH albo sprawdź odcisk klucza

TYLKO ODCZYT

Ręczne zaakceptowanie nowego klucza hosta: ssh

```
ssh app01.ipa.zielona.example
```

Łączymy się ręcznie w celu zaakceptowania nowego odcisku klucza, zanim zrobi to sam Jenkins.

ssh = Secure Shell: szyfrowane zdalne logowanie

TYLKO ODCZYT

Sprawdzenie zajętości wolumenu agenta: df -h

```
df -h /home/jenkins_agent
```

Sprawdzamy zajętość wolumenu.

df = disk free: ile miejsca wolnego na dyskach

□□ TYLKO ODCZYT

Sprawdzenie rzeczywistej ścieżki Javy: readlink

```
readlink -f /etc/alternatives/jre/bin/java
```

Sprawdzamy, gdzie faktycznie stoi Java.

□□ **readlink** = pokaż, na co wskazuje dowiązanie symboliczne

□□ OSTROŻNIE · NIEODWRACALNE

Usunięcie przestarzałego remoting.jar: rm

```
rm -f /home/jenkins_agent/remoting.jar
```

Usuwamy przestarzały plik z katalogu roboczego agenta, żeby wymusić świeżą kopię przy następnym uruchomieniu.

□□ **rm** = remove: usuń (bez kosza, na zawsze)

Z życia wzięte

□□ Z ŻYCIA WZIĘTE

W poniedziałek rano nowy agent na `app01` upiera się, że jest offline, mimo że proces wygląda na uruchomiony. Tomek sprawdza log node'a i widzi ostrzeżenie o niezgodności klucza SSH, bo `app01` został niedawno przeinstalowany po awarii dysku. Po usunięciu starego wpisu z `known_hosts` kontenera Jenkinsa i akceptacji nowego klucza agent wraca online w kilka sekund.

Po co to istnieje

Jeden agent szybko przestaje wystarczać — kolejne zespoły chcą własnego środowiska buildowego (inny JDK, dostęp do innej sieci, więcej mocy obliczeniowej), a kolejka zadań na współdzielonym agencie robi się długa. Dodanie kolejnego agenta jest jedynym sposobem skalowania Jenkinsa w poziomie, bez dokładania zasobów do samego kontrolera.

Dodanie agenta po SSH

Zanim zaczniemy konfigurację po stronie Jenkinsa, sprawdzamy, że ruch po porcie 22 z serwera kontrolera do serwera agenta jest przepuszczony, a agent ma dostęp do wewnętrznego rejestru obrazów i repozytorium Gita, jeśli buildy z nich korzystają.

Przygotowanie dedykowanego wolumenu

Dobrą praktyką jest wydzielenie osobnego dysku pod katalog domowy agenta (standardowo 10 GB na start) zamiast trzymania buildów na wolumenie systemowym — zgodnie z konwencją nazewniczą `vg<Przeznaczenie>` / `lv<Przeznaczenie>` grupa nazywa się `vgJenkins_Agent`, wolumen `lvJenkins_Agent`.

Wymiana kluczy SSH

Zanim skonfigurujemy node w Jenkinsie, kontroler musi choć raz połączyć się po SSH z serwerem agenta, żeby wymienić klucze i zapisać odcisk hosta w swoim `known_hosts` — inaczej pierwsza próba uruchomienia agenta skończy się błędem weryfikacji klucza opisanym niżej.

Dodanie poświadczeń w Jenkinsie

W Manage Jenkins → Manage Credentials wybieramy odpowiednią domenę (np. Global) i dodajemy nowe poświadczenia:

Pole	Wartość
Kind	Username with password (lub SSH Username with private key)
Scope	Global (Jenkins, nodes, items, all child items, etc)
Username	`jenkins_agent`
ID	`jenkins_agent_app01`
Description	Credentials for SSH connection to jenkins agent on app01

Więcej o typach poświadczeń i ich zakresach na stronie Poświadczenia i bezpieczeństwo.

Konfiguracja node'a

W Manage Jenkins → Nodes → New Node wybieramy typ „Permanent Agent". Zamiast wypełniać konfigurację od zera, wygodnie jest skorzystać z opcji „Copy Existing Node" i wskazać już działającego agenta o zbliżonej roli jako szablon — trzeba wtedy tylko dostosować nazwę hosta, etykiety i poświadczenia.

Najważniejsze pola konfiguracji node'a:

Pole	Znaczenie
Remote root directory	`/home/jenkins_agent` — odpowiednik JENKINS_HOME, tylko po stronie agenta
Labels	etykiety, po których zadania wybierają ten agent, patrz Architektura Jenkinsa: kontroler, agenci, workspace
Usage	„Use this node as much as possible" albo „Only build jobs with label expressions matching this node"
Launch method	„Launch agents via SSH"
Host	`app01.ipa.zielona.example`
Credentials	poświadczenia dodane w kroku wyżej
Host Key Verification Strategy	„Known hosts file Verification Strategy" w normalnej pracy
Availability	„Keep this agent online as much as possible"

Po zapisaniu Jenkins przekierowuje do widoku nowo utworzonego node'a i po chwili powinien pojawić się status online.

Przykład poprawnego zakończenia logu uruchomienia agenta:

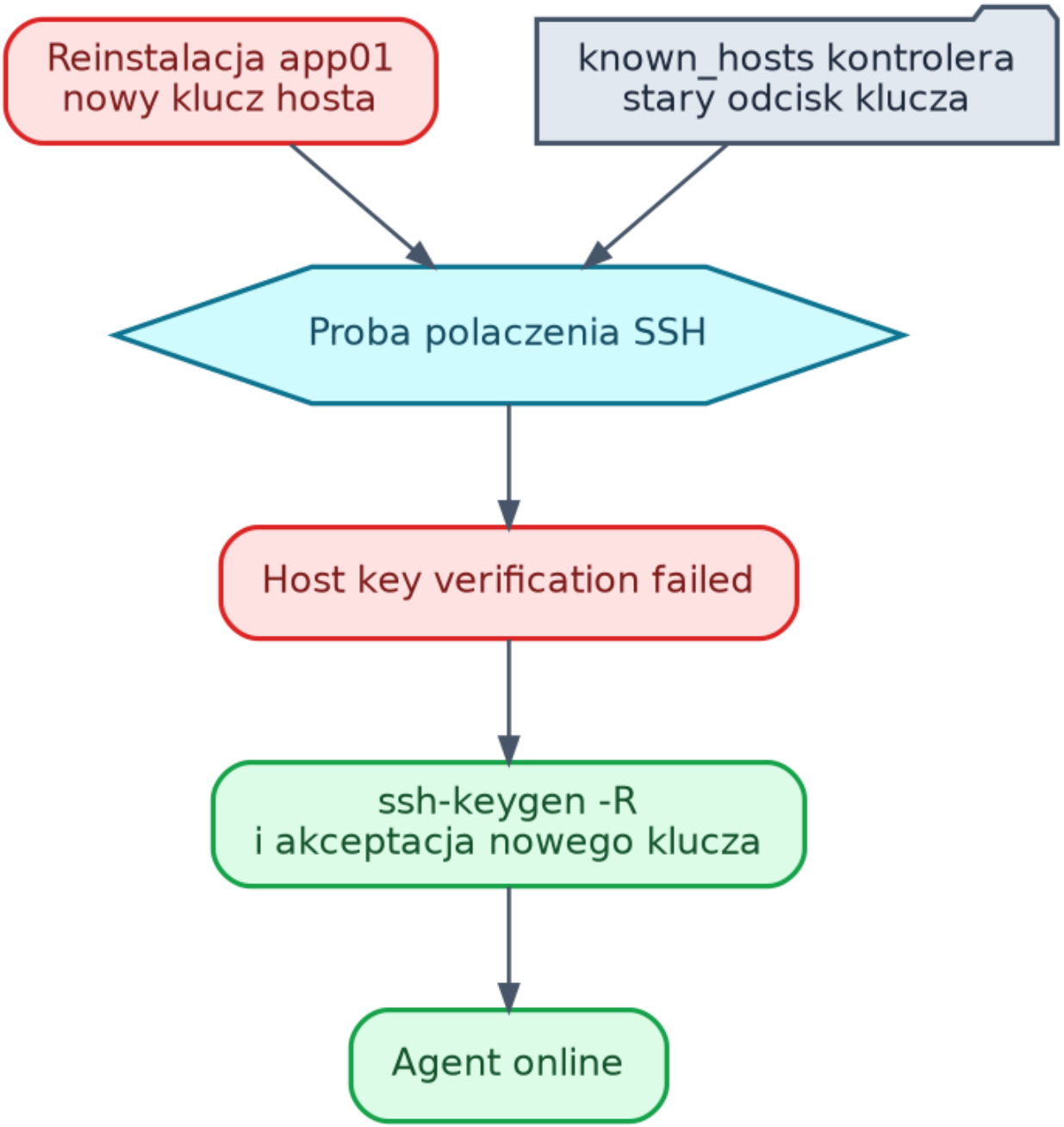
```
Remoting version: 4.5
This is a Unix agent
Agent successfully connected and online
```

Agent w kontenerze

Zamiast trwałej maszyny wirtualnej agent może być kontenerem tworzonym na potrzeby pojedynczego builda przez odpowiednią wtyczkę (Docker, Kubernetes) — konfiguruje się wtedy nie pojedynczy node, tylko szablon (`Cloud`), z którego Jenkins tworzy efemeryczne agenty na żądanie. Zaletą jest zawsze czyste środowisko i brak potrzeby ręcznego utrzymania maszyny; wadą — czas startu kontenera doliczony do każdego builda oraz konieczność trzymania obrazu z odpowiednim zestawem narzędzi (JDK, git, ewentualnie docker-in-docker) zawsze aktualnym.

Diagnostyka

Agent offline z powodu niezgodności klucza SSH



Reinstalacja agenta zmienia klucz hosta, stary odcisk w known_hosts kontrolera odrzuca połączenie, naprawa to usunięcie wpisu i akceptacja nowego klucza (kliknięcie otwiera pełny rozmiar)

> **Pojęcie: known_hosts kontrolera.** Kontener aplikacji Jenkins trzyma własny plik `known_hosts` pod `/var/jenkins_home/.ssh/known_hosts`, niezależny od `known_hosts` hosta, na którym kontener działa. Gdy serwer agenta zostanie przeinstalowany albo zmieni adres IP, jego klucz hosta się zmienia, a stary wpis w tym pliku powoduje odrzucenie połączenia.

Objaw w logu node'a:

```
WARNING: POSSIBLE DNS SPOOFING DETECTED!
The ED25519 host key for app01.ipa.zielona.example has changed,
and the key for the corresponding IP address has a different value.
Offending key for IP in /var/jenkins_home/.ssh/known_hosts: 19
Host key verification failed.
```

Po potwierdzeniu (`yes`) wracamy do konfiguracji node'a w Jenkinsie i klikamy „Launch agent" ponownie — status powinien przejść na online. Jeśli node tymczasowo miał ustawioną „Non verifying Verification Strategy" jako obejście, po naprawie przełączamy go z powrotem na „Known hosts file Verification Strategy" i zapisujemy — inaczej weryfikacja klucza hosta jest wyłączona na stałe, co jest ryzykiem bezpieczeństwa, nie tylko obejściem.

Brak miejsca na wolumenie agenta

Workspace kolejnych buildów i cache narzędzi budowania potrafią wypełnić dedykowany wolumen szybciej niż się wydaje.

Jeśli wolumen jest pełny, pierwszym krokiem jest wyczyszczenie workspace nieużywanych już zadań (patrz Architektura Jenkinsa: kontroler, agenci, workspace), a dopiero potem — jeśli to nie wystarcza — powiększenie wolumenu logicznego, tak jak opisuje strona Powiększanie przestrzeni dyskowej bez restartu.

Zła ścieżka JDK na agencie

Objaw w logu uruchomienia agenta:

```
[SSH] Checking java version of /home/jenkins_agent/jdk/bin/java
bash: /home/jenkins_agent/jdk/bin/java: no such file or directory
```

Jenkins próbuje uruchomić agenta interpreterem Javy wskazanym w konfiguracji node'a (Node Properties → Tool Locations → JAVA_HOME), a ta ścieżka nie istnieje na danym serwerze.

I poprawiamy pole `JAVA_HOME` w konfiguracji node'a na rzeczywistą ścieżkę zainstalowanego JDK — po zapisaniu Jenkins ponawia próbę uruchomienia agenta automatycznie.

Rozjazd wersji remoting.jar

> **Pojęcie: remoting.jar.** Biblioteka, którą kontroler kopiuje na agenta przy każdym uruchomieniu połączenia SSH i która faktycznie realizuje komunikację między obydwoma procesami. Wersja `remoting.jar` jest przywiązana do wersji kontrolera — stary plik pozostawiony na agencie z czasów innej wersji Jenkinsa bywa źródłem niejasnych błędów protokołu, mimo że samo połączenie SSH działa poprawnie.

Sprawdzenie wersji zgłaszanej przy starcie agenta w logu node'a:

```
Remoting version: 4.5
This is a Unix agent
```

Jeśli wersja w logu wygląda podejrzanie nisko w stosunku do aktualnej wersji kontrolera, usuwamy przestarzały plik z katalogu roboczego agenta, żeby wymusić świeżą kopię przy następnym uruchomieniu.

Kolejne uruchomienie node'a (przyciskiem „Launch agent") skopiuje aktualną wersję dopasowaną do kontrolera.

Częste błędy

Zmiana Host Key Verification Strategy na „Non verifying" na stałe. To ustawienie jest wygodnym obejściem na czas diagnozowania problemu z kluczem hosta, ale wyłącza sprawdzanie tożsamości serwera agenta przy każdym kolejnym połączeniu — pozostawione na stałe otwiera furtkę na atak man-in-the-middle na ruch SSH między kontrolerem a agentem.

Próba ponownego uruchomienia agenta, który wciąż jest „connected". Log zgłasza wtedy `The Agent is connected, disconnect it before to try to connect it again` — trzeba go najpierw jawnie rozłączyć z poziomu widoku node'a, zanim Jenkins zgodzi się nawiązać połączenie od nowa.

Zapominanie o wymianie kluczy SSH przed pierwszą konfiguracją node'a. Kolejność ma znaczenie — jeśli kontroler nigdy wcześniej nie łączył się z danym serwerem, pierwsza próba uruchomienia agenta z poziomu Jenkinsa kończy się błędem weryfikacji klucza, bo host nie istnieje

jeszcze w `known_hosts` kontenera.

□□ Ćwiczenia

> Zadania sprawdzające.

1. Opisujemy kolejność czynności potrzebnych do dodania nowego agenta SSH, od przygotowania dysku po pierwszy status online w Jenkinsie.
2. Agent zgłasza `Host key verification failed` po reinstalacji serwera. Wyjaśniamy, dlaczego doszło do tego błędu i jakie dwie komendy trzeba wykonać wewnątrz kontenera Jenkinsa, żeby go naprawić.
3. Wolumen `vgJenkins_Agent` jest zapełniony w 95%. Wskazujemy, co sprawdzamy najpierw, zanim sięgniemy po `lvextend`.
4. Wyjaśniamy, dlaczego zostawienie node'a na stałe z „Non verifying Verification Strategy" jest problemem bezpieczeństwa, nie tylko niedopracowaną konfiguracją.

Powiązane: Architektura Jenkinsa: kontroler, agenci, workspace · Poświadczenia i bezpieczeństwo · Utrzymanie i diagnostyka · Powiększanie przestrzeni dyskowej bez restartu

Instalacja Jenkinsa w Dockerze

Typ: procedura

Dotyczy: Jenkins LTS 2.568.3, obraz jenkins/jenkins:2.568.3-lts-jdk21 (sprawdzone na hub.docker.com dnia 2026-09-28)

Czas: ~30 min

Efekt: kontroler Jenkinsa i jeden agent działające w osobnych kontenerach na `ci01`, wystawione przez reverse proxy pod `jenkins.zielona.example`, gotowe do dodania kolejnych agentów i pierwszego pipeline'u.

Skróty użyte na tej stronie (10)

Skrót	Co znaczy
CI	Continuous Integration: automatyczne budowanie i testowanie kodu po każdej zmianie; w ITIL także configuration item, czyli element konfiguracji: każdy składnik usługi zapisany w CMDB
HTTP	Hypertext Transfer Protocol: protokół stron WWW
IP	Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection)
JNLP	metoda podłączenia agenta Jenkinsa, w której to agent inicjuje połączenie do kontrolera
LTS	Long Term Support: wersja z długim wsparciem poprawkami
README	Read Me: plik tekstowy z opisem projektu, wykresu lub szablonu, czytany jako pierwszy
RPM	RPM Package Manager: format paczek i narzędzie instalujące oprogramowanie w RHEL/Rocky, dnf korzysta z niego pod spodem
SSH	Secure Shell: szyfrowane zdalne logowanie do serwera
WebSocket	stałe, dwukierunkowe połączenie przeglądarki z serwerem, przez które Home Assistant na żywo odświeża panele; serwer pośredniczący musi przekazywać nagłówki Upgrade i Connection
WWW	World Wide Web: strony i aplikacje otwierane w przeglądarce po HTTP albo HTTPS

Zanim zaczniemy

Z ŻYCIA WZIĘTE

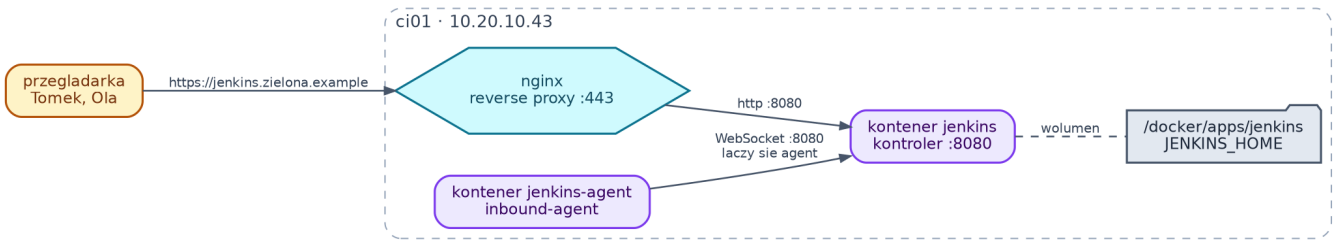
Ola prosi o osobne środowisko CI dla nowego mikroserwisu, żeby jej zespół nie kolejkował się za buildami reszty firmy na wspólnym `ci01`. Zamiast kolejnego agenta SSH stawiamy kontener Jenkinsa obok istniejącej instalacji RPM, żeby sprawdzić migrację do Dockera bez ryzyka dla produkcyjnego kontrolera. Po podłączeniu agenta przez WebSocket build startuje bez otwierania dodatkowego portu na firewallu `ci01`.

Strona Instalacja i aktualizacja Jenkinsa opisuje instalację z pakietu RPM, jako usługę systemd, bez kontenerów. Tutaj stawiamy to samo w Dockerze: dwa kontenery, kontroler i agent, każdy z osobnego obrazu.

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.

Wymaganie	Uwaga
Docker Engine i wtyczka Compose	sprawdzamy `docker compose version`
Wolny port `8080` na hoście	jedyny port, który wystawiamy na zewnątrz kontenera
DNS `jenkins.zielona.example` wskazujący na `ci01`	pod tym adresem stanie reverse proxy
Reverse proxy nginx przed kontenerem	z książki Docker, terminuje TLS tak samo jak dla innych aplikacji na `docker01`

Uważaj: obraz zawsze przypinamy do konkretnej wersji (`2.568.3-lts-jdk21`), nigdy do `lts` czy `latest`. Aktualizacja to świadoma zmiana numeru w `compose.yaml`, opisana na stronie Utrzymanie i diagnostyka, nie przypadkowe pobranie nowszej wersji przy kolejnym `docker compose pull`.



Kontener nginx przyjmuje ruch z przeglądarki, przekazuje go do kontenera jenkins, a agent łączy się z kontrolerem przez WebSocket na tym samym porcie 8080 (kliknięcie otwiera pełny rozmiar)

Przebieg

1. Katalogi zgodnie z konwencją

```
sudo mkdir -p /docker/apps/jenkins /docker/apps/jenkins-agent
sudo mkdir -p /docker/compose/jenkins
```

2. Plik .env

Plik `/docker/compose/jenkins/.env`, obok `compose.yaml`, nie w nim samym. Wartość `JENKINS_AGENT_SECRET` na razie zostawiamy pustym placeholderem, uzupełnimy ją w kroku 6, dopiero po utworzeniu node'a w samym Jenkinsie:

```
JENKINS_AGENT_NAME=agent-docker01
JENKINS_AGENT_SECRET=UzupelnimyPoUtworzeniuNodea
```

3. Plik compose.yaml

```
services:
  jenkins:
    image: jenkins/jenkins:2.568.3-lts-jdk21
    container_name: jenkins
    restart: always
    ports:
      - "8080:8080"
    volumes:
      - /docker/apps/jenkins:/var/jenkins_home
      - /etc/timezone:/etc/timezone:ro
      - /etc/localtime:/etc/localtime:ro

  jenkins-agent:
    image: jenkins/inbound-agent:3391.va_37fa_a_305d6d-3-jdk21
    container_name: jenkins-agent
    restart: always
    depends_on:
      - jenkins
```

```
    environment:
      - JENKINS_URL=http://jenkins:8080/
      - JENKINS_AGENT_NAME=${JENKINS_AGENT_NAME}
      - JENKINS_SECRET=${JENKINS_AGENT_SECRET}
      - JENKINS_WEB_SOCKET=true
    volumes:
      - /docker/apps/jenkins-agent:/home/jenkins/agent
```

> **Pojęcie: WebSocket.** Sposób komunikacji sieciowej, w którym po jednym początkowym żądaniu HTTP obie strony trzymają otwarte to samo połączenie i wysyłają nim dane w obie strony, bez kolejnych osobnych żądań. Dzięki `JENKINS_WEB_SOCKET=true` agent łączy się z kontrolerem tym samym portem `8080`, którego i tak już używa interfejs WWW: nie trzeba wystawiać osobnego portu `50000` na sieć kontenerów ani na firewall hosta, tak jak przy klasycznym połączeniu JNLP opisanym na stronie Architektura Jenkinsa: kontroler, agenci, workspace.

Agent nie ma tu żadnego portu w sekcji `ports`, bo łączy się z kontrolerem sam, poziomo, przez wewnętrzną sieć Dockera utworzoną automatycznie dla tego pliku Compose, adresując go po nazwie usługi `jenkins`, nie po adresie IP.

4. Pierwsze uruchomienie samego kontrolera

Startujemy na razie tylko kontroler, agent poczeka, aż będzie miał czym się połączyć:

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

5. Odblokowanie instalacji i wtyczki

Hasło startowe wyciągamy z wnętrza kontenera zamiast z dysku hosta:

```
docker exec jenkins cat /var/jenkins_home/secrets/initialAdminPassword
```

Przykład:

```
7f2c4b91ae3d4058b6e1c9a2f0d7883c
```

Dalej kreator, wtyczki sugerowane i pierwszy administrator wyglądają dokładnie tak samo, jak opisuje krok 5 i 6 strony Instalacja i aktualizacja Jenkinsa, niezależnie od tego, czy kontroler stoi w kontenerze, czy z pakietu RPM.

6. Utworzenie node'a agenta i sekret połączenia

W Manage Jenkins → Nodes → New Node zakładamy node typu „Permanent Agent" o nazwie `agent-docker01`, dokładnie takiej, jaką wpisaliśmy w `.env` pod `JENKINS_AGENT_NAME`. W polu „Launch method" wybieramy „Launch agent by connecting it to the controller" (inbound). Po zapisaniu Jenkins pokazuje na stronie node'a wygenerowany sekret jednorazowego użytku.

Wklejamy ten sekret do `/docker/compose/jenkins/.env` w miejsce placeholdera:

```
JENKINS_AGENT_NAME=agent-docker01
JENKINS_AGENT_SECRET=a1b2c3d4e5f60718293a4b5c6d7e8f90
```

7. Uruchomienie agenta

```
docker compose up -d jenkins-agent
```

Sprawdzamy log startu:

```
docker compose logs -f jenkins-agent
```

Przykład:

```
jenkins-agent  | INFO: Connecting to http://jenkins:8080/ via WebSocket
jenkins-agent  | INFO: Remoting version: 3391.v...
jenkins-agent  | INFO: Agent successfully connected and online
```

8. Reverse proxy

Fragment konfiguracji nginksa na `ci01`, według wzoru z książki Docker, z tą samą zasadą co przy instalacji bez kontenerów: bez nagłówka `X-Forwarded-Proto` Jenkins generuje linki z `http://` zamiast `https://`. Przy WebSocket dochodzi jeden dodatkowy wymóg, nagłówki `Upgrade`/`Connection`, inaczej agent nie połączy się przez proxy:

```
server {
    listen 443 ssl;
    server_name jenkins.zielona.example;

    location / {
        proxy_pass http://127.0.0.1:8080;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

Weryfikacja

```
docker compose ps
```

Przykład:

```
NAME            IMAGE                                              STATUS
jenkins         jenkins/jenkins:2.568.3-lts-jdk21                  Up 3 minutes
jenkins-agent   jenkins/inbound-agent:3391.va_37fa_a_305d6d-3-jdk21   Up 1 minute
```

```
curl -s -o /dev/null -w '%{http_code}\n' http://127.0.0.1:8080/login
```

Przykład:

```
200
```

Status agenta sprawdzamy też z poziomu Manage Jenkins → Nodes: `agent-docker01` powinien świecić się na zielono, bez czerwonego wykrzyknika offline.

□□ ZRZUT EKRANU DO UZUPEŁNIENIA

Zrzut do uzupełnienia: ekran Manage Jenkins → Nodes z listą node'ów, widoczny wiersz `agent-docker01` ze statusem online (zielona ikonka) i liczbą executorów.

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/jenkins /docker/apps/jenkins-agent
```

Gdy coś nie zadziała

Objaw	Przyczyna	Naprawa
Kontener `jenkins` restartuje się w pętli zaraz po starcie	katalog `/docker/apps/jenkins` należy do roota, a proces w obrazie działa jako użytkownik o UID 1000	`sudo chown -R 1000:1000 /docker/apps/jenkins` przed kolejną próbą startu
`jenkins-agent` w logu pokazuje `Failed to connect` w kółko	zły `JENKINS_AGENT_SECRET` (placeholder nieuzupełniony albo sekret z innego node'a)	wracamy do widoku node'a w Manage Jenkins → Nodes i kopiujemy sekret jeszcze raz, dokładnie
Agent łączy się bez proxy, ale nie przez `jenkins.zielona.example` z zewnątrz	reverse proxy bez nagłówków `Upgrade`/`Connection` dla WebSocket	dokładamy `proxy_http_version 1.1` i oba nagłówki z kroku 8
Po restarcie kontenera znikają zainstalowane wtyczki	wolumen `/docker/apps/jenkins` nie był podpięty w `compose.yaml` albo ścieżka się nie zgadza	sprawdzamy `docker inspect jenkins` w sekcji `Mounts`, poprawiamy wpis `volumes`
Logowanie po reverse proxy przekierowuje w pętli	brak nagłówka `X-Forwarded-Proto`	dokładamy nagłówek w konfiguracji nginksa i ustawiamy właściwy adres URL w Manage Jenkins → System

Powiązane: Instalacja i aktualizacja Jenkinsa · Architektura Jenkinsa: kontroler, agenci, workspace · Konwencja Dockera w Zielonej

Źródła: jenkins.io/download/lts · docs.jenkins.io, rozdział „Installing Jenkins with Docker" · hub.docker.com/r/jenkins/jenkins · github.com/jenkinsci/docker-inbound-agent, plik README (zmienne `JENKINS_WEB_SOCKET`, `JENKINS_SECRET`)