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

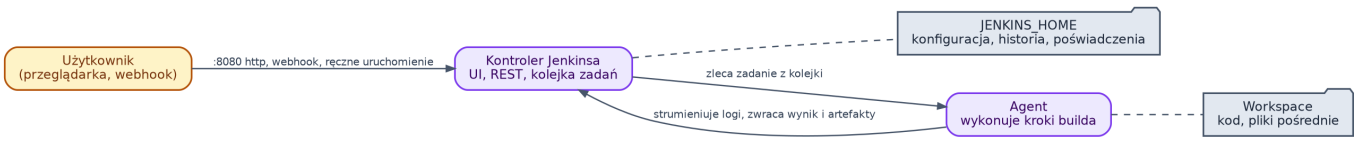
Z ŻYCIA WZIĘTE

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)

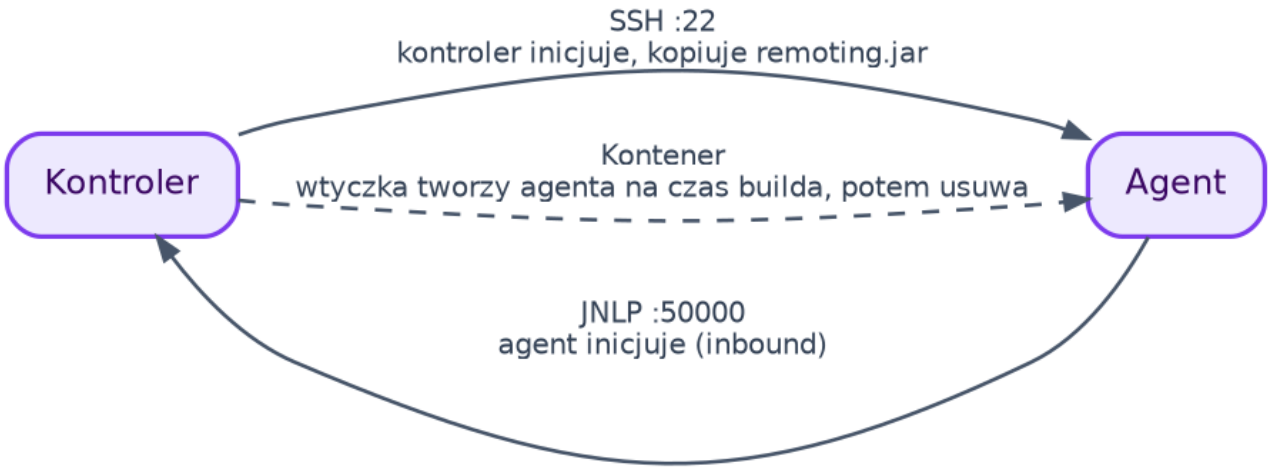
Czym jest agent

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.

Revision #37
Created 2026-09-25 07:39:49 UTC by Tomasz Konieczny
Updated 2026-10-26 14:14:55 UTC by Tomasz Konieczny