

Continuous Integration z wykorzystaniem Jenkins

Continuous Integration (CI) to praktyka w dziedzinie rozwoju oprogramowania, która polega na regularnym, częstym łączeniu zmian dokonywanych przez programistów w centralnym repozytorium. Zdobędziecie wiedzę i umiejętności związane z implementacją tej praktyki przy użyciu narzędzia Jenkins, które jest jednym z popularnych narzędzi do automatyzacji procesów CI/CD.

O trenerze

Imię i nazwisko: Juliusz Marciniak

Zawód: Programista, DevOps, TechLead

Doświadczenie: 10 lat

Technologie:

- Java, Spring
- Jenkins, GitLab, Nexus
- Docker, Kubernetes

Jestem programistą Java z 10-letnim stażem. W codziennej pracy zajmuję się tematami cloudowymi i Kubernetesowymi. Oprócz tego doglądam architektury aplikacji w swoich projektach. Jestem entuzjastą nowych rozwiązań i technologii. Zwracam dużą uwagę, by rzeczy wdrażane przeze mnie były przemyślane i spójne. Czasami też programuję... i rekrutuję.

Agenda

Dzień 1

1. Co to jest CI/CD?
2. Dlaczego CI/CD jest ważne?
3. Architektura Jenkinsa
4. Instalacja i konfiguracja Jenkinsa
5. Tworzenie projektów (jobs, konfiguracja repozytorium, skrypty budowania i testowania)
6. Pipeliny (deklaratywne i skryptowe)

Dzień 2

1. Integracja z narzędziami zewnętrznymi m.in. git, maven gradle
2. Powiadomienia
3. Jak zautomatyzować proces wdrażania
4. Wdrażanie na różne środowiska (testowe, produkcyjne)
5. Bezpieczeństwo i zarządzanie uprawnieniami

Zasady

- Staramy się nie spóźniać
- Telefon odbieramy poza salą
- Jesteśmy aktywni
- Jeśli utknęliśmy na problemie podczas realizacji zadań – natychmiast informujemy
- Nie śmiejemy się z innych i nie krytykujemy pomysłów innych
- Każdy ma prawo do wyrażania swojej opinii
- Przerwa na kawę co 1-1,5h
- Przerwa obiadowa o 13:00

Podstawy teoretyczne

Czym jest CI/CD

CI oznacza ciągłą integrację. W przeciwieństwie do praktyk starszych, w których kod potrafił być tygodniami lub miesiącami trzymany na osobnej gałęzi i nie był mergowany, podejście CI zakłada, że mergujemy kod tak często, jak to możliwe.

Programowanie to praca zespołowa, nad jednym kodem może pracować kilka osób, kilka zespołów, a nawet kilka firm. Im szybciej zmiany zostaną zmergowane do głównych gałęzi. Dzięki takiemu podejściu wiemy, że kod się kompiluje. Dodatkowo narzędzia wspomagające CI powinny automatycznie uruchomić testy jednostkowe/integracyjne, żeby sprawdzić, że aplikacja nadal działa prawidłowo. Małe zmiany są łatwiejsze do przetestowania niż kod, który powstaje przez 6 miesięcy.

CD oznacza ciągłe dostarczanie (delivery) lub wdrażanie (deployment). Zasadnicza różnica między tymi pojęciami jest taka, że ciągłe dostarczanie posiada kroki manualne, np. wdrożenie na produkcję, a ciągłe wdrażanie wszystkie kroki ma automatyczne.

Zarówno jedno, jak i drugie polega na jak najczęstszym wydawaniu aplikacji na środowiska testowe i produkcyjne. Podobnie jak w podejściu CI im szybciej wdrożymy coś na środowisko, tym szybciej będziemy mogli to przetestować i sprawdzić, czy działa. Znowu — mała zmiana, łatwe testy.

Dodatkowo CD będzie niezwykle pomocne w momencie, kiedy zmiana okazała się problematyczna. Dość łatwo będzie wykonać rollback takiej wersji.

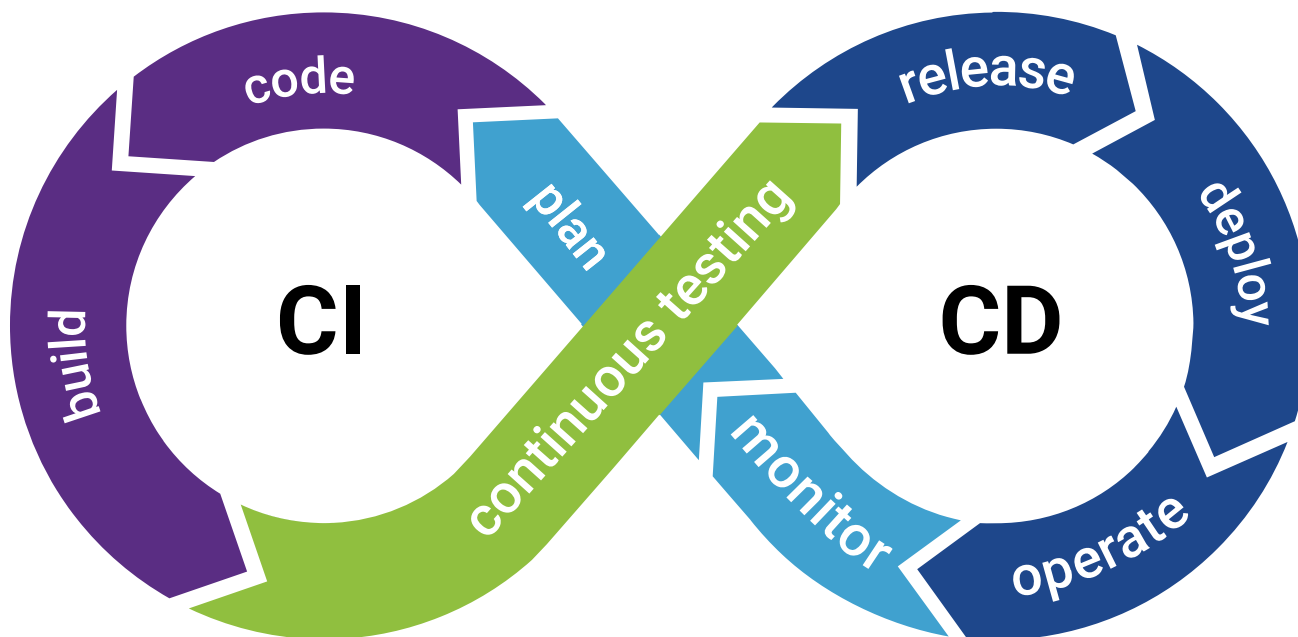


Figure 1. <https://www.synopsys.com/glossary/what-is-cicd.html>

Korzyści

- Lepsza jakość kodu
- Szybsze dostarczanie nowych funkcjonalności
- Automatyzacja procesów
- Zmniejszenie kosztów
- Uproszczony rollback zmian

Kim jest DevOps Engineer

DevOps Engineer — osoba, która jednocześnie jest programistą i administratorem. Działa na przecięciu tych funkcji.

DevOps zajmuje się tworzeniem pipeline'ów CI/CD, monitoringiem aplikacji i infrastruktury, tworzeniem i przygotowywaniem infrastruktury, automatyzacją procesów.

Narzędzia do CI/CD

- [Jenkins](#)
- [TeamCity](#)
- [CircleCI](#)
- [GitLab](#)
- [Bamboo](#)



	 Jenkins	 TeamCity	 circleci	 Bamboo	 GitLab
Open source	Yes	No	No	No	No
Ease of use & setup	Medium	Medium	Medium	Medium	Medium
Built-in features	3/5	4/5	4/5	4/5	4/5
Integration	★★★★★	★★★★	★★★★★	★★★★	★★★★★
Hosting	On premise & Cloud	On premise & Cloud	On premise	On premise & Bitbucket as Cloud	On premise & Cloud
Free version	Yes	Yes	Yes	Yes	Yes
Build agent license pricing	Free	From \$59 per month	From \$15 per month	From \$10 one-off payment	From \$19 per month per user
Supported OSs	Windows, Linux, macOS, Unix-like OS	Linux or MacOS	Windows, Linux, macOS, Solaris, FreeBSD and more	Windows, Linux, macOS, Solaris	Linux distributions: Ubuntu, Debian, CentOS, Oracle Linux

Figure 2. <https://katalon.com/resources-center/blog/ci-cd-tools>

Ekosystem narzędzi DevOps

DevOps Tools Ecosystem 2021

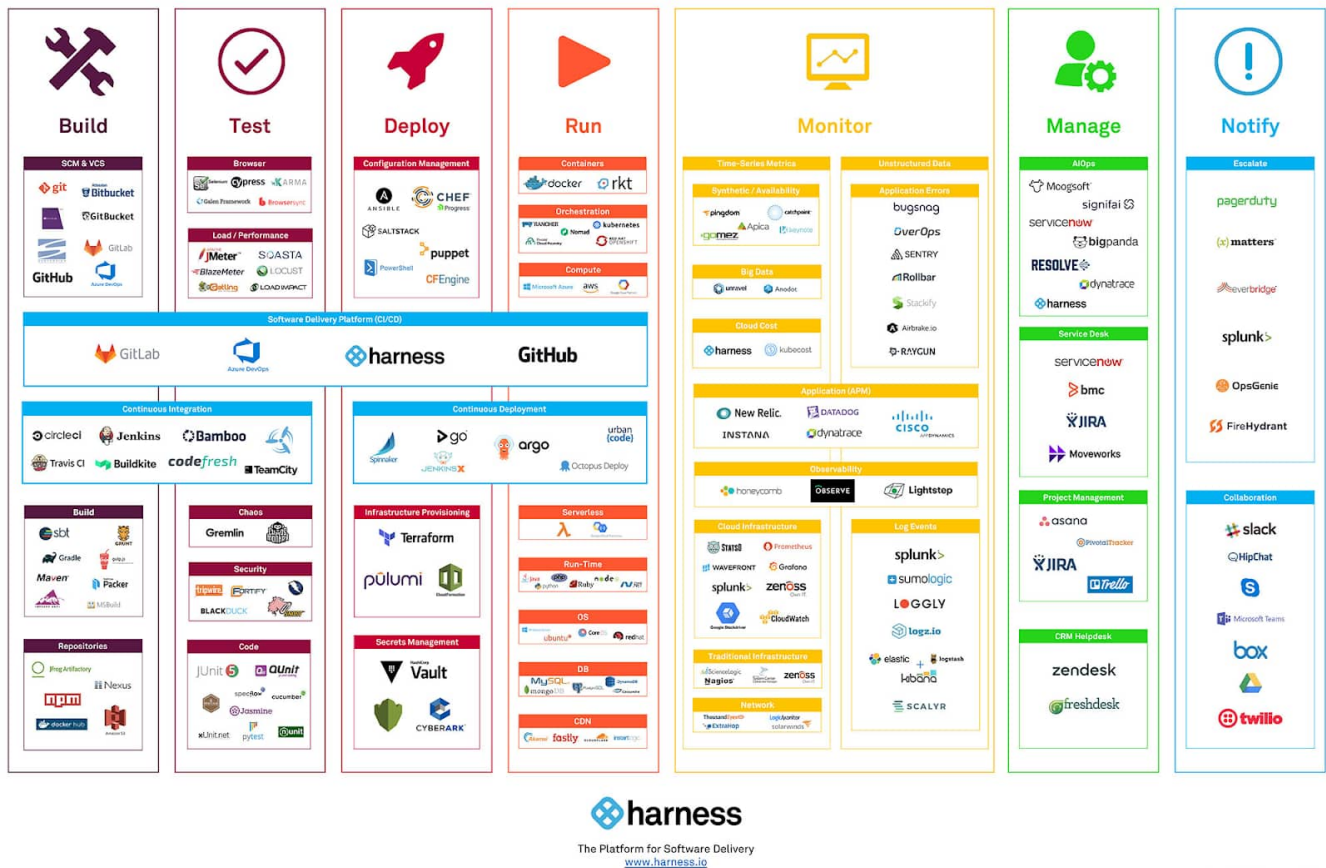


Figure 3. <https://www.harness.io/blog/continuous-delivery-tools>

Jenkins

Opis

Jenkins – serwer typu open source służący do automatyzacji związanej z tworzeniem oprogramowania. W szczególności ułatwia budowanie, testowanie i wdrażanie aplikacji, czyli umożliwia rozwój oprogramowania w trybie ciągłej integracji i ciągłego dostarczania

Jenkins może być rozszerzony o wtyczki. Stanowią one dużą siłę tego rozwiązania, ponieważ domyślnie Jenkins posiada bardzo mało opcji. Mnogość wtyczek pozwala praktycznie dowolnie kształtować instalacje Jenkinsa. Z tego też powodu, instalacje między firmami, a nawet zespołami potrafią się znacząco różnić.

Architektura

Architektura Jenkinsa to typowy master-slave. Mamy główny kontroler i dodatkowe hosty zwane Agentami. Agenci mogą być uruchomieni na dowolnej maszynie. Główny kontroler zleca wykonanie pracy konkretnemu agentowi. Oczywiście agent musi być w stanie wykonać daną operację. Jeżeli nie ma na nim zainstalowanego dockera, to nie będziemy mogli zbudować obrazu.

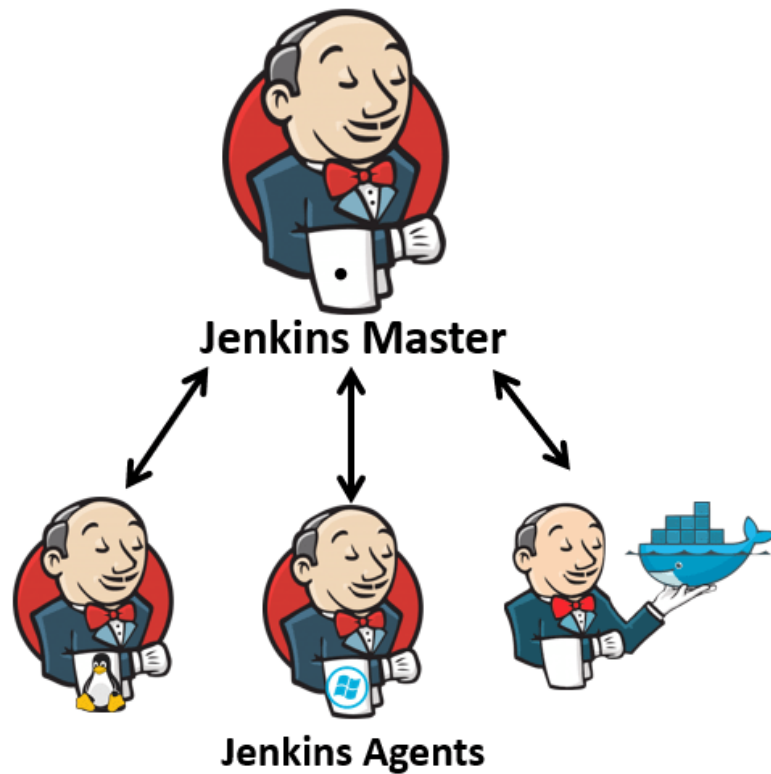


Figure 4. <https://szkolajenkinsa.pl/2021/02/07/czym-jest-jenkins/>

Pipeline

W Jenkinsie możemy definiować joby na różne sposoby. Obecnie najefektywniejszy sposób to pipeline. Dzięki temu możemy łatwo go zmienić, powielić i utrzymywać go w repozytorium kodu. Jest to plik najczęściej nazywa się **Jenkinsfile**, który jest napisany w Groovym. Cała dokumentacja znajduje się [tutaj](#).

Przykładowy pipeline

```
pipeline {
  agent none
  stages {
    stage('Example Build') {
      agent { docker 'maven:3.9.0-eclipse-temurin-11' }
      steps {
        echo 'Hello, Maven'
        sh 'mvn --version'
      }
    }
    stage('Example Test') {
      agent { docker 'openjdk:8-jre' }
      steps {
        echo 'Hello, JDK'
        sh 'java -version'
      }
    }
  }
}
```

```
}
```

Zadania

Instalacja i konfiguracja

Zadanie 1. Instalacja i uruchomienie Jenkinsa bezpośrednio w systemie Linux

1. Zainstaluj i uruchom Jenkinsa

Instalacja Jenkinsa w systemie Windows

1. Zainstaluj za pomocą tutoriala z oficjalnej strony <https://www.jenkins.io/doc/book/installing/windows/>

Instalacja Jenkinsa w systemie MacOS

1. Zainstaluj za pomocą tutoriala z oficjalnej strony <https://www.jenkins.io/doc/book/installing/macos/>

Instalacja Jenkinsa w systemie Linux

1. Zainstaluj Javę za pomocą komend

```
sudo apt-get update  
sudo apt-get install openjdk-17-jre
```

2. Sprawdź instalację Javy za pomocą polecenia `java -version`. Oczekiwany rezultat

```
openjdk version "17.0.7" 2023-04-18  
OpenJDK Runtime Environment (build 17.0.7+7-Debian-1deb11u1)  
OpenJDK 64-Bit Server VM (build 17.0.7+7-Debian-1deb11u1, mixed mode, sharing)
```

3. Zainstaluj Jenkinsa za pomocą komend

```
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee \  
  /usr/share/keyrings/jenkins-keyring.asc > /dev/null  
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \  
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \  
  /etc/apt/sources.list.d/jenkins.list > /dev/null  
sudo apt-get update  
sudo apt-get install jenkins
```

4. Uruchom Jenkinsa i ustaw automatyczne uruchamianie przy restarcie maszyny

```
sudo systemctl enable jenkins
sudo systemctl start jenkins
```

5. Sprawdź działanie Jenkinsa za pomocą komendy `sudo systemctl status jenkins`. Oczekiwany rezultat

```

❏ jenkins.service - Jenkins Continuous Integration Server
   Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor prese>
   Active: active (running) since Thu 2023-06-08 12:14:44 CEST; 2h 22min ago

```

6. Wejdź na adres: <http://localhost:8080/>

7. Powinienesz otrzymać taką stronę

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

`/var/jenkins_home/secrets/initialAdminPassword`

Please copy the password from either location and paste it below.

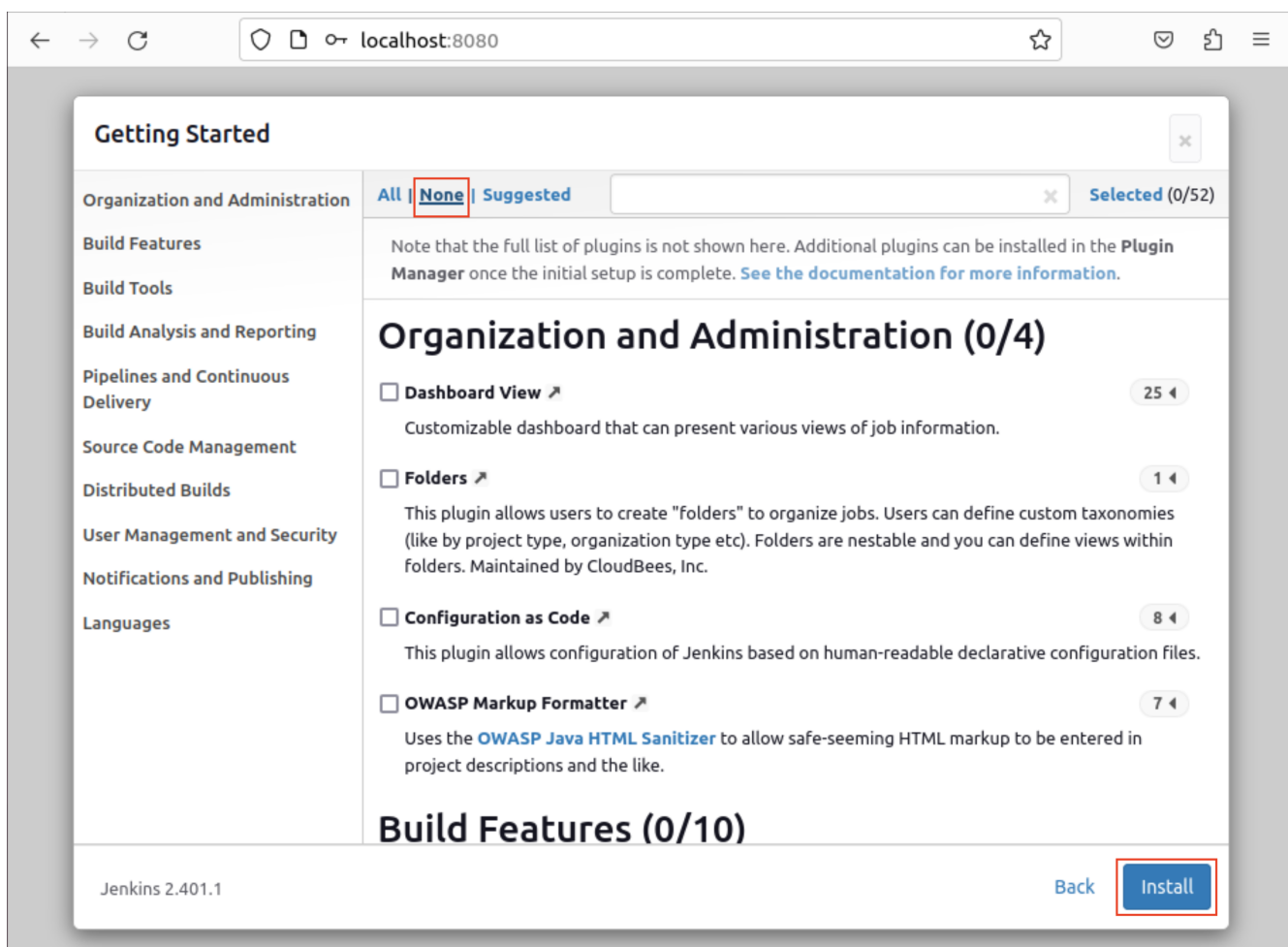
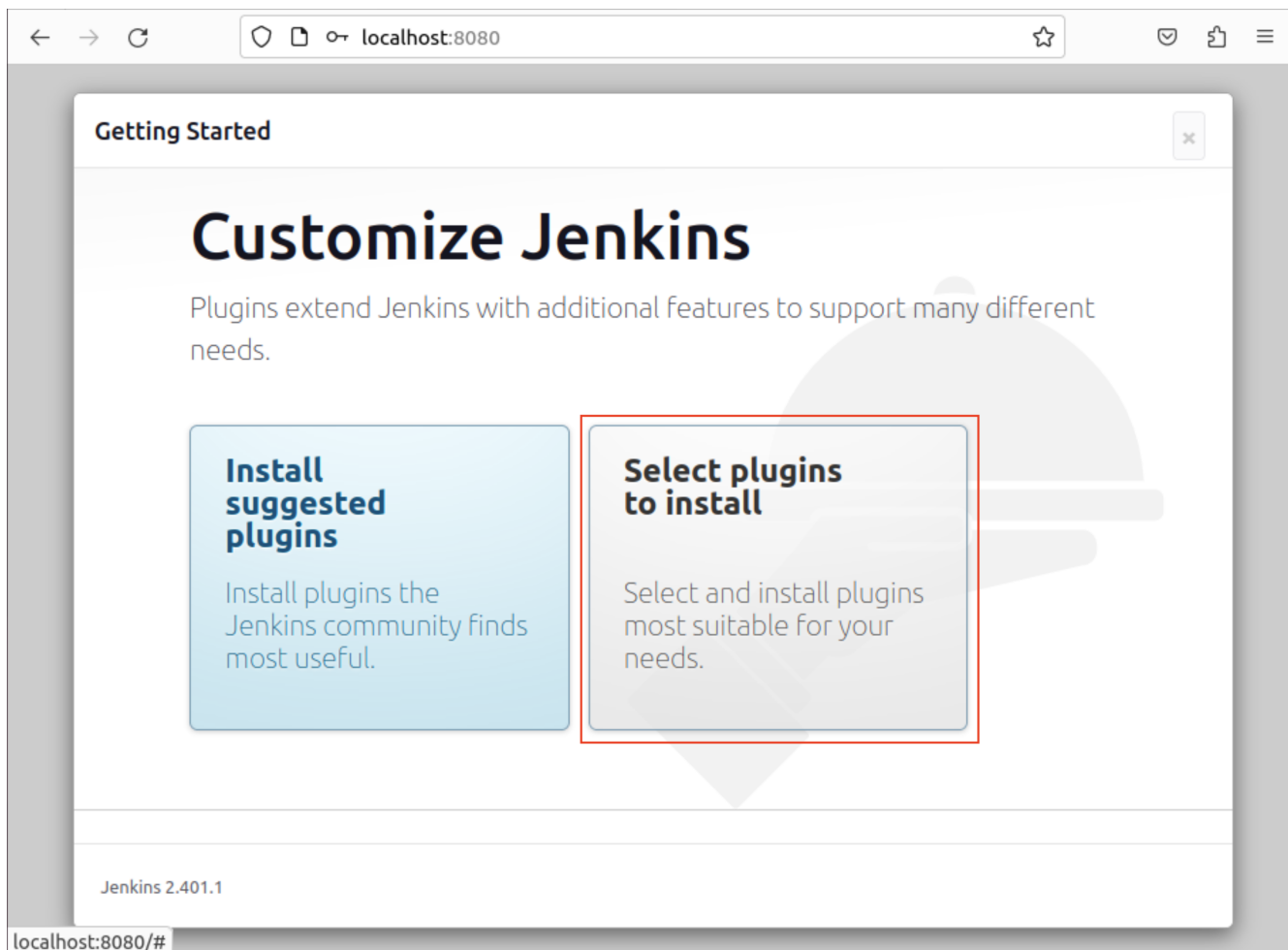
Administrator password



Continue

8. Znajdź hasło administratora komendą `sudo cat /var/lib/jenkins/secrets/initialAdminPassword`

Pierwsza konfiguracja



← → ↺

localhost:8080

☆

🔒 📄 ☰

Getting Started

Username

juliusz_marciniak

Password

.....

Confirm password

.....

Full name

Jenkins 2.401.1

Skip and continue as admin

Save and Continue

Getting Started

Instance Configuration

Jenkins URL:

http://localhost:8080/

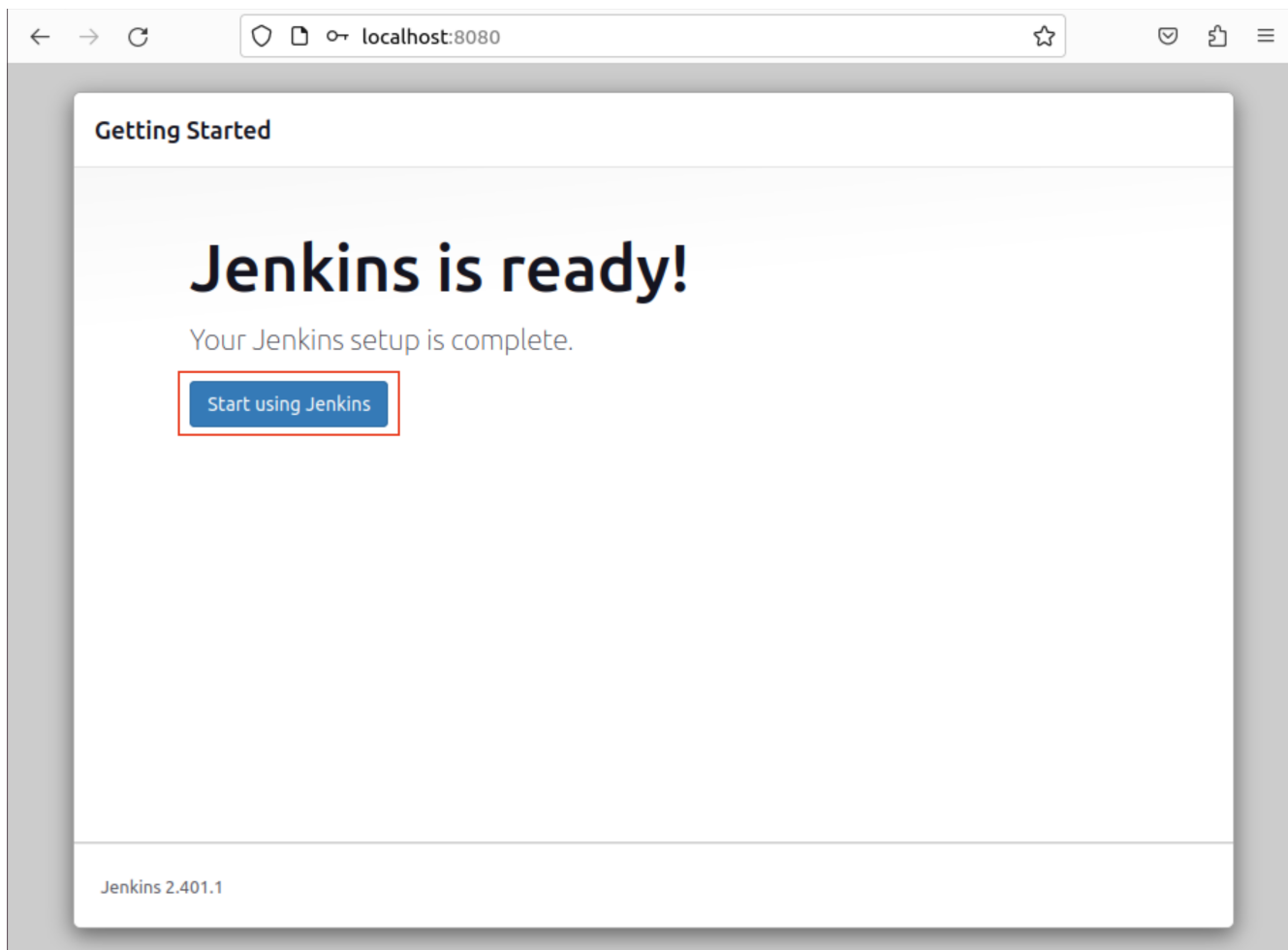
The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the BUILD_URL environment variable provided to build steps.

The proposed default value shown is **not saved yet** and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

Jenkins 2.401.1

Not now

Save and Finish



Zadanie 2. Instalacja i konfiguracja pluginów

1. Zainstaluj pluginy:
 - Git
 - Eclipse Temurin installer
 - Pipeline
 - Workspace Cleanup
2. Skonfiguruj narzędzia
 - Dodaj instaler JDK w wersji `jdk-17.0.8.1+1` i nazwij go `jdk-17`
 - Dodaj instaler Mavena w wersji `3.9.4` i nazwij go `maven-3`

Integracja z Git

Zadanie 1. Pobranie repozytorium

1. Stwórz zadanie o nazwie `github-fetch`
2. Pobierz jakiekolwiek repozytorium z GitHub, np. `pet-clinic`
3. Sprawdź workspace czy faktycznie został pobrany kod

Zadanie 2. Zmiana brancha

1. Stwórz nowe zadanie o nazwie **change-branch**
2. Pobierz jakiegokolwiek repozytorium z GitHub, np. [pet-clinic](#)
3. Zmień brancha na inny
4. Sprawdź workspace czy faktycznie został pobrany kod z odpowiedniego brancha

Integracja z Maven

Zadanie 1. Aplikacja mavenowa

1. Stwórz nowe repozytorium na GitHub
2. Wrzuć kod przykładowy projekt ze strony <https://start.spring.io>

Zadanie 2. Budowanie aplikacji mavenowej

1. Stwórz zadanie o nazwie **maven-spring-base**
2. Ściągnij kod z poprzedniego repozytorium
3. Wykonaj budowanie projektu za pomocą maven

Zadanie 3. Releasowanie aplikacji mavenowej

1. Stwórz zadanie o nazwie **maven-spring-base-release**
2. Ściągnij kod z poprzedniego repozytorium
3. Wykonaj releasowanie projektu za pomocą maven

Podstawowy Pipeline

Zadanie 1. Uruchomienie pierwszego Pipeline

1. Stwórz nowy pipeline o nazwie **base-echo**
2. Jako pipeline wklej

```
pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                sh 'echo Hello World!'
            }
        }
    }
}
```

3. Uruchom pipeline

4. Sprawdź logi wykonania pipeline i sprawdź, czy pojawił się napis `Hello World!`

Zadanie 2. Generowanie plików

1. Dodaj w pipeline z poprzedniego zadania utworzenie pliku z losową nazwą, np. `sh "uuidgen | xargs touch"`
2. Uruchom pipeline dwukrotnie
3. Sprawdź co znajduje się w workspace

Zadanie 3. Wyczyszczenie Workspace

1. Zmodyfikuj pipeline z poprzedniego zadania, dodając krok czyszczący Workspace, używając pluginu Workspace Cleanup
2. Uruchom pipeline dwukrotnie
3. Sprawdź co znajduje się w workspace

Zadanie 4. Budowanie aplikacji Spring Boot

1. Stwórz nowy pipeline o nazwie `pet-clinic-build`
2. Na podstawie pipeline poprzedniego zadania stwórz pipeline, który za pomocą komendy `mvn clean verify` zbuduje projekt `pet-clinic`

Zadanie 5. Testy

1. Zmodyfikuj pipeline z poprzedniego zadania dodając do niego obsługę JUnit
2. Uruchom pipeline
3. Obejrzyj jak wyglądają testy JUnit

Zadanie 6. Podział na stage

1. Napisany przez siebie pipeline z poprzedniego zadania podziel na 3 stage: ['Clean', 'Checkout', 'Build']

Zadanie 7. Notyfikacja na Slacku

1. Rozszerz napisany przez siebie pipeline z poprzedniego zadania o wysłanie notyfikacji na Slacku
2. Notyfikacje mają się wysyłać w momencie pozytywnego buildu/negatywnego/niestabilnego
3. Dodaj kanał na slacku o nazwie `jenkins-imie-nazwisko`
4. Uruchom pipeline
5. Sprawdź, czy notyfikacja została wysłana
6. Zmodyfikuj pipeline w taki sposób, żeby komunikat na Slacka zawierał parametry `BUILD_TAG` i `BUILD_URL`

Zadanie 8. Zmienne środowiskowe

1. Stwórz nowy pipeline o nazwie `base-curl-with-credentials`
2. Wywołaj w nim `sh "curl -i https://httpbin.org/basic-auth/foo/bar"`
3. Uruchom pipeline
4. Sprawdź wynik
5. Dodaj credentiale user: `foo`, password: `bar`
6. Użyj credentiali jako zmienne środowiskowe
7. Adres również umieść w zmiennych środowiskowych
8. Wywołaj `curla` z autoryzacją `sh "curl -i -u 'user:pass' https://httpbin.org/basic-auth/foo/bar"`
9. Uruchom pipeline
10. Sprawdź logi
11. Doprowadź do sytuacji, kiedy Jenkins nie będzie zwracał warna
12. Zmień credentiale ze zmiennych środowiskowych na dyrektywę `withCredentials`
13. Uruchom pipeline

Zadanie 9. Parametry wejściowe

1. Stwórz nowy pipeline o nazwie `base-echo-with-parameters`
2. Dodaj parametr wejściowy do pipeline
3. Wyświetl parametr wejściowy w konsoli (jak w Zadaniu 1)

Pipeline wewnątrz repozytorium

Zadania 1. Dodanie Pipeline do repozytorium

1. Zrób fork `repozytorium`
2. Dodaj plik `Jenkinsfile` do repozytorium z zawartością z Zadania 7 z poprzedniego modułu
3. Stwórz pipeline o nazwie `example-build`
4. Uruchom pipeline

Zadanie 2. Pipeline w osobnym repozytorium

1. Stwórz repozytorium na GitHub
2. Dodaj plik `Jenkinsfile` do repozytorium z zawartością z poprzedniego zadania
3. Stwórz pipeline o nazwie `example-external-build`, w którym konfiguracja będzie brana z repozytorium, które stworzyłeś, ale będzie budowany projekt, z poprzedniego zadania

Zadanie 3. Warunkowe wykonanie kroków

1. Dodaj do pipeline klauzulę **when**, żeby wszystkie kroki były pomijane w momencie, jeśli ostatni commit zaczyna się od **[ci skip]**
2. Zrób commita zaczynającego się od **[ci skip]**
3. Uruchom pipeline
4. Sprawdź w logach czy kroki się wykonały

Zadanie 4. SCM Skip Plugin

1. Zrób to samo co w poprzednim zadaniu, ale z użyciem pluginu SCM Skip

Zadanie 5. Triggery

1. Dodaj do któregoś z poprzednich pipeline trigger, który będzie wywoływał ten pipeline co 1 minutę
2. Sprawdź efekty

Zadanie 6. Active Choices i Scriptler

1. Zainstaluj plugin **Active Choices**
2. Zainstaluj plugin **Scriptler**
3. Stwórz aktywne pola wyboru
4. Możesz wzorować się na <https://plugins.jenkins.io/uno-choice/#plugin-content-example>, ale proponuję zrobić własne pola wyboru

Połączmy to z Dockerem

Zadanie 1. Zbuduj obraz Dockerowy za pomocą mavena

1. Stwórz nowy pipeline o nazwie **pet-clinic-build-docker**
2. Użyj pipeline z modułu Podstawowy Pipeline Zadanie 4 i zmień komendę z **mvn clean verify** na **mvn clean spring-boot:build-image**
3. Uruchom pipeline
4. Spójrz w logi spróbuj doprowadzić go do działania

Zadanie 2. Zbuduj obraz Dockerowy za pomocą pluginu

1. Stwórz pipeline o nazwie **docker-simple-build**
2. Użyj Dockerfile z <https://github.com/rechandler12/szkolenie-ci-jenkins> katalog **docker-pipeline**
3. Zbuduj obraz dockerowy za pomocą pluginu **Docker Pipeline**

Zadanie 3. Użyj Docker Agenta do zbudowania aplikacji pet-clinic

1. Stwórz pipeline o nazwie **pet-clinic-build-docker-without-dind**

2. Użyj pipeline z modułu Podstawowy Pipeline Zadanie 4
3. Usuń sekcję tools
4. Zmień sekcję agents, żeby użyć obrazu: `maven:3.9.2-eclipse-temurin-17`

Zadanie 4. Użyj Docker Agenta do zbudowania obrazu Dockerowego aplikacji pet-clinic

1. Stwórz pipeline o nazwie `pet-clinic-build-docker-with-dind`
2. Użyj pipeline z poprzedniego zadania
3. Zmień komendę `mvn clean verify` na `mvn clean spring-boot:build-image`
4. Spróbuj doprowadzić do tego, żeby obraz się zbudował

Konfiguracja dodatkowego Workera

Zadanie 1. Skonfigurowanie 2 dodatkowych Workerów

1. Zainstaluj plugin `Command Agent Launcher`
2. Dodaj 2 dodatkowe workery
 - Liczba executorów: 1
 - Labelki: `workerA` i `workerB`
 - Remote root directory: `/home/jenkins/agentA` i `/home/jenkins/agentB`
 - Launch method: Launch agent via execution of command on the master
 - Launch command (X do podmiany): `docker run -i --rm --name agentX --init jenkins/agent java -jar /usr/share/jenkins/agent.jar`

Zadanie 2. Wyłącz executory na masterach

1. Ustaw liczbę executorów na masterze na 0
2. Puść pipeline, które masz już skonfigurowane
3. Sprawdź, jaki jest efekt

Zadanie 3. Uruchamianie budowania na konkretnym workerze

1. Znajdź pipeline, który zakończył się sukcesem
2. Ustaw sekcję `agent` na wybranie workera po odpowiedniej labelce
3. Uruchom pipeline
4. Sprawdź, czy faktycznie się uruchomił na danym workerze
5. Usuń worker z tą labelką
6. Ponownie uruchom pipeline
7. Jaki efekt otrzymałeś?