Szkolenie CI/CD z wykorzystaniem Jenkinsa i GitLaba

Szkolenie z podstaw CI/CD. Po szkoleniu będziecie wiedzieć, czym jest CI/CD. Nauczycie się definiować pipeline w narzędziu Jenkins i GitLab.

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. Podstawy teoretyczne
2. Narzędzia do CI/CD — dlaczego wybieramy konkretne rozwiązania, GitLab/Bamboo/Jenkins
3. Jenkins
    a. Podstawy teoretyczne
    b. Instalacja i konfiguracja
    c. Pluginy
    d. Deklarowanie pipeline

Dzień 2

1. GitLab Runner

    a. Instalacja i konfiguracja
    b. Porównanie z Jenkinsem
    c. Deklarowanie pipeline

Dzień 3

1. Nexus
    a. Podstawy teoretyczne
    b. Instalacja i konfiguracja
2. Integracja narzędzi w spójny sposób
3. Wstęp do CI/CD w chmurze

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 produkcje, a ciągłe wdrażanie wszystkie kroki ma automatyczne.

Zarówno jedno, jak i drugie polega na jak najczęstszym wydawaniu aplikacje 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.

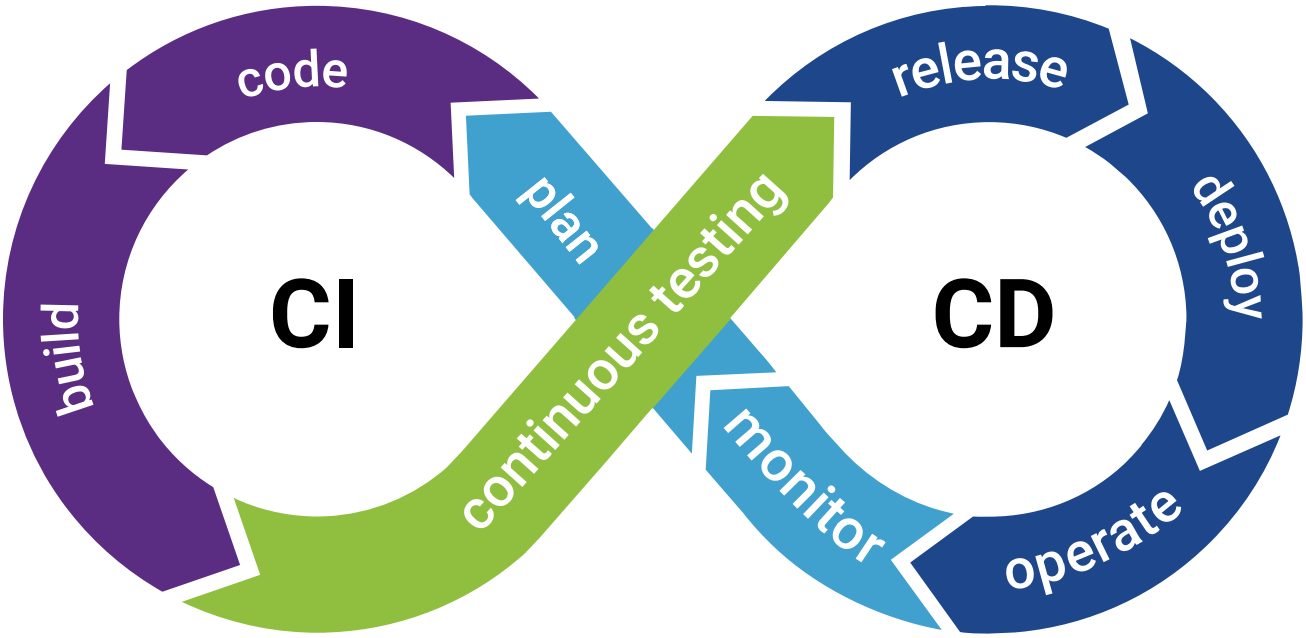


Figure 1. https://www.synopsys.com/glossary/what-is-cicd.html

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 pipelinów 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

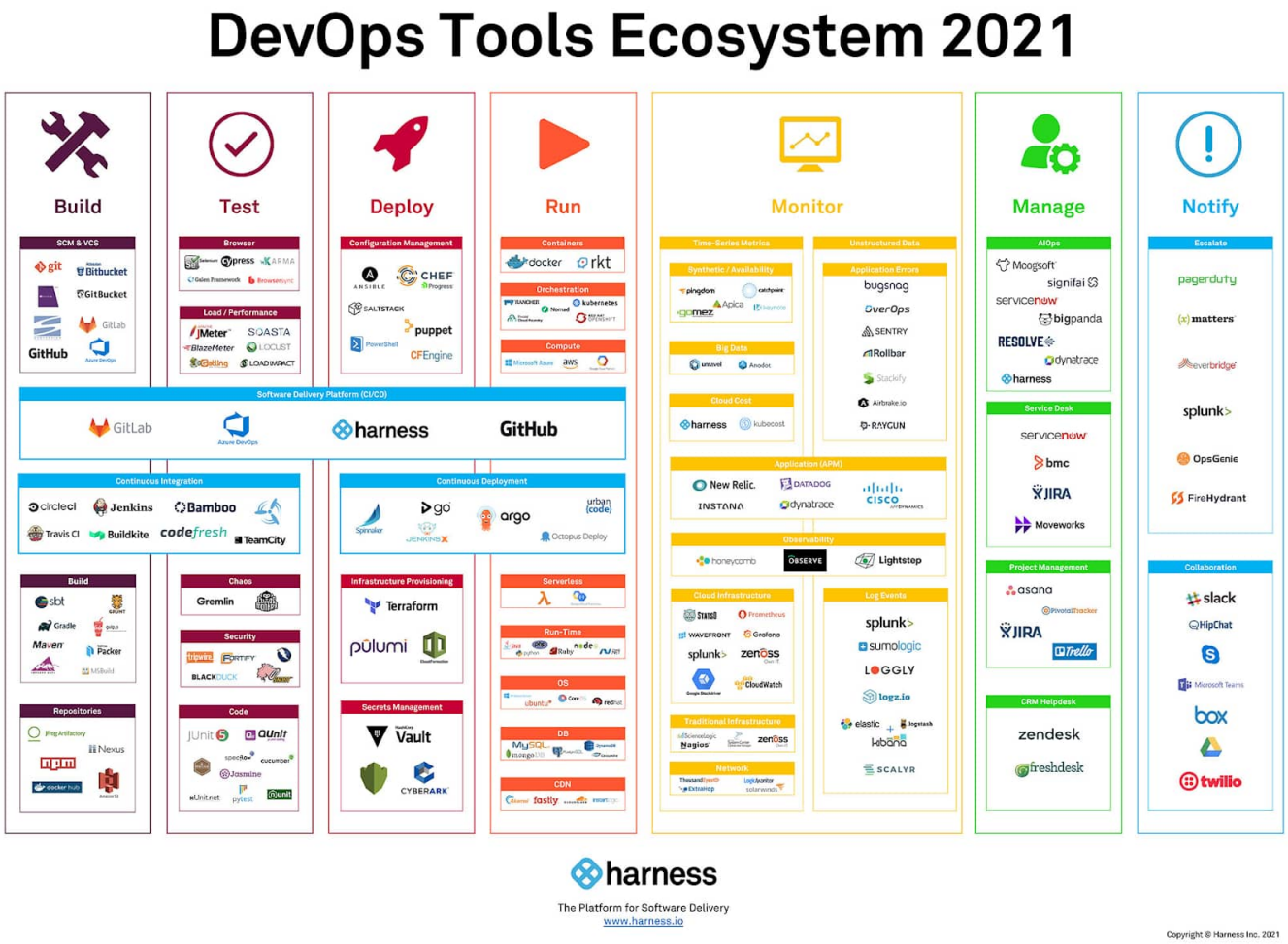


Figure 3. https://www.harness.io/blog/continuous-delivery-tools

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.

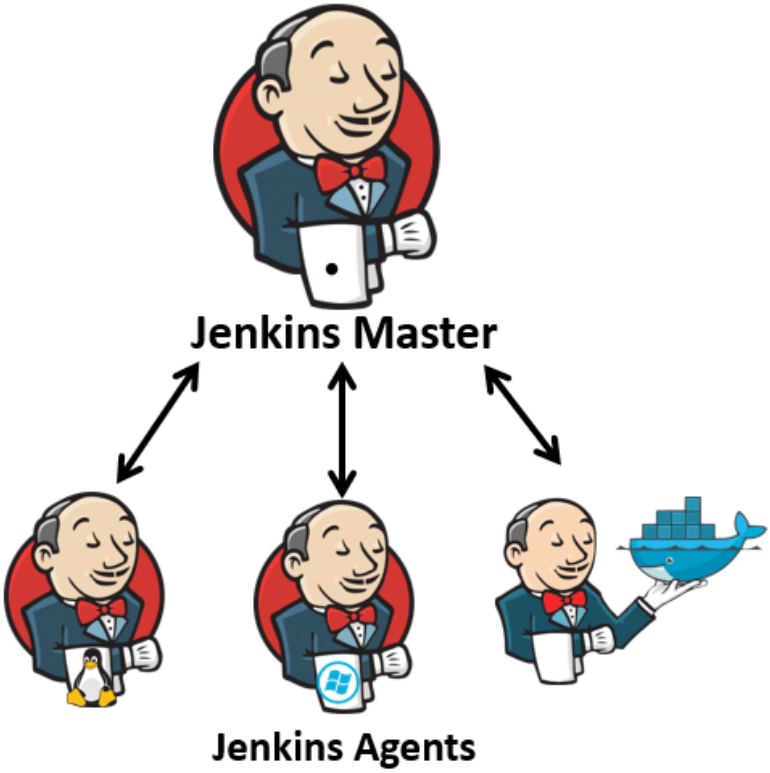


Figure 4. https://szkolajenkinsa.pl/2021/02/07/czym-jest-jenkins/

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
```

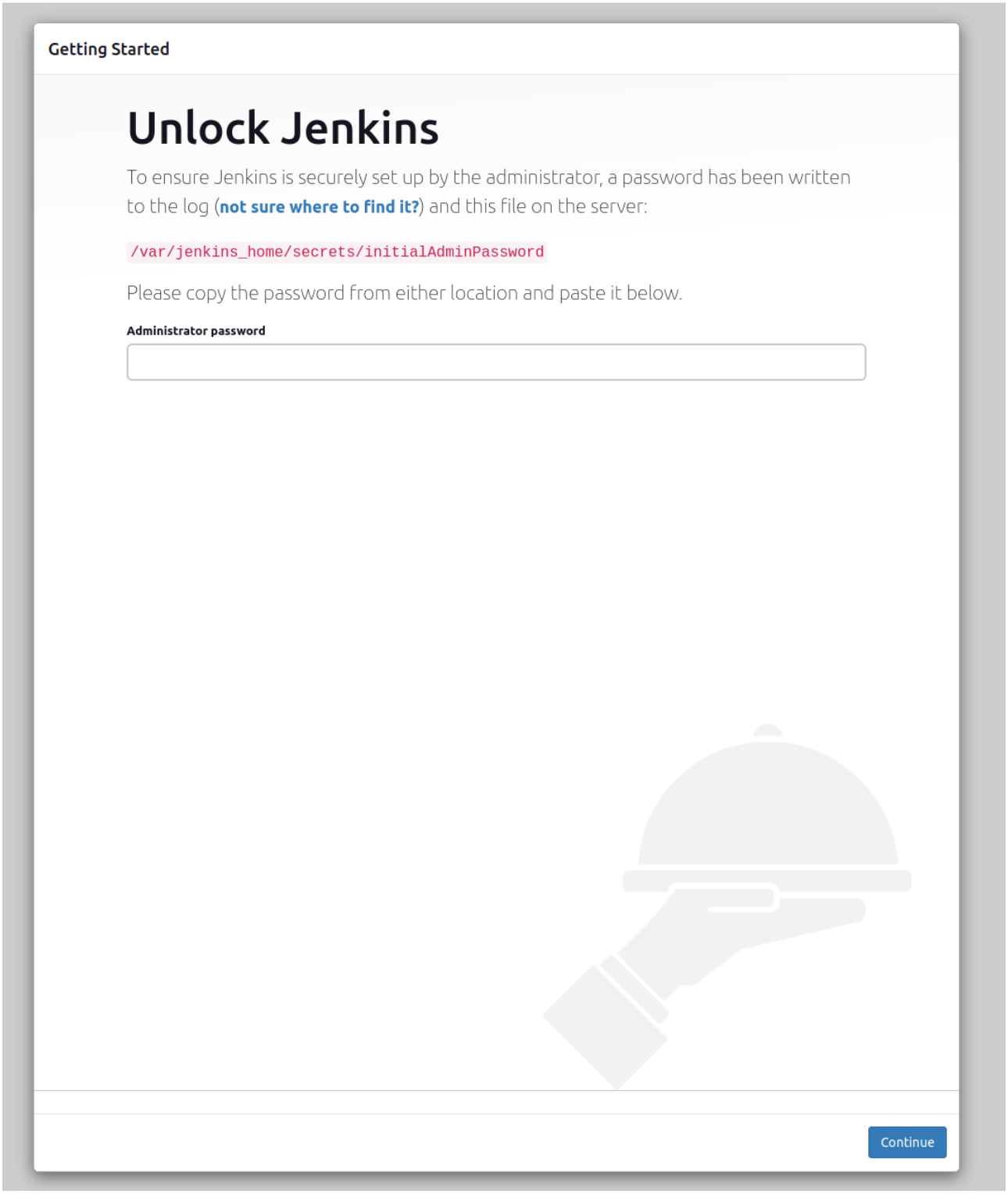
4. Uruchom Jenkinsa i ustaw automatyczne uruchamianie przy restarcie maszyny

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

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

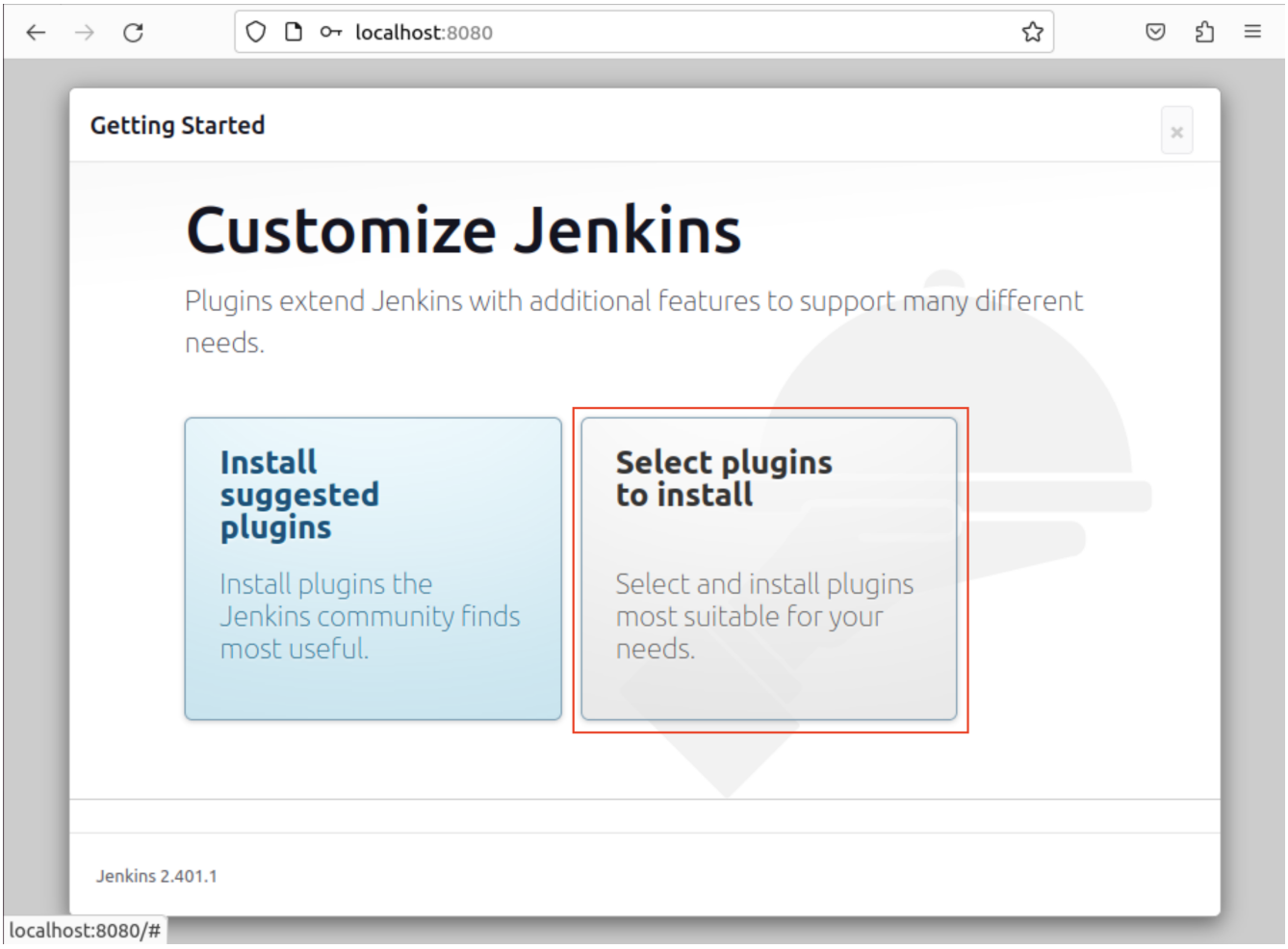
```
● jenkins.service - Jenkins Continuous Integration Server
     Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor prese>
     Active: active (running) since Thu 2023-06-08 12:14:44 CEST; 2h 22min ago
```

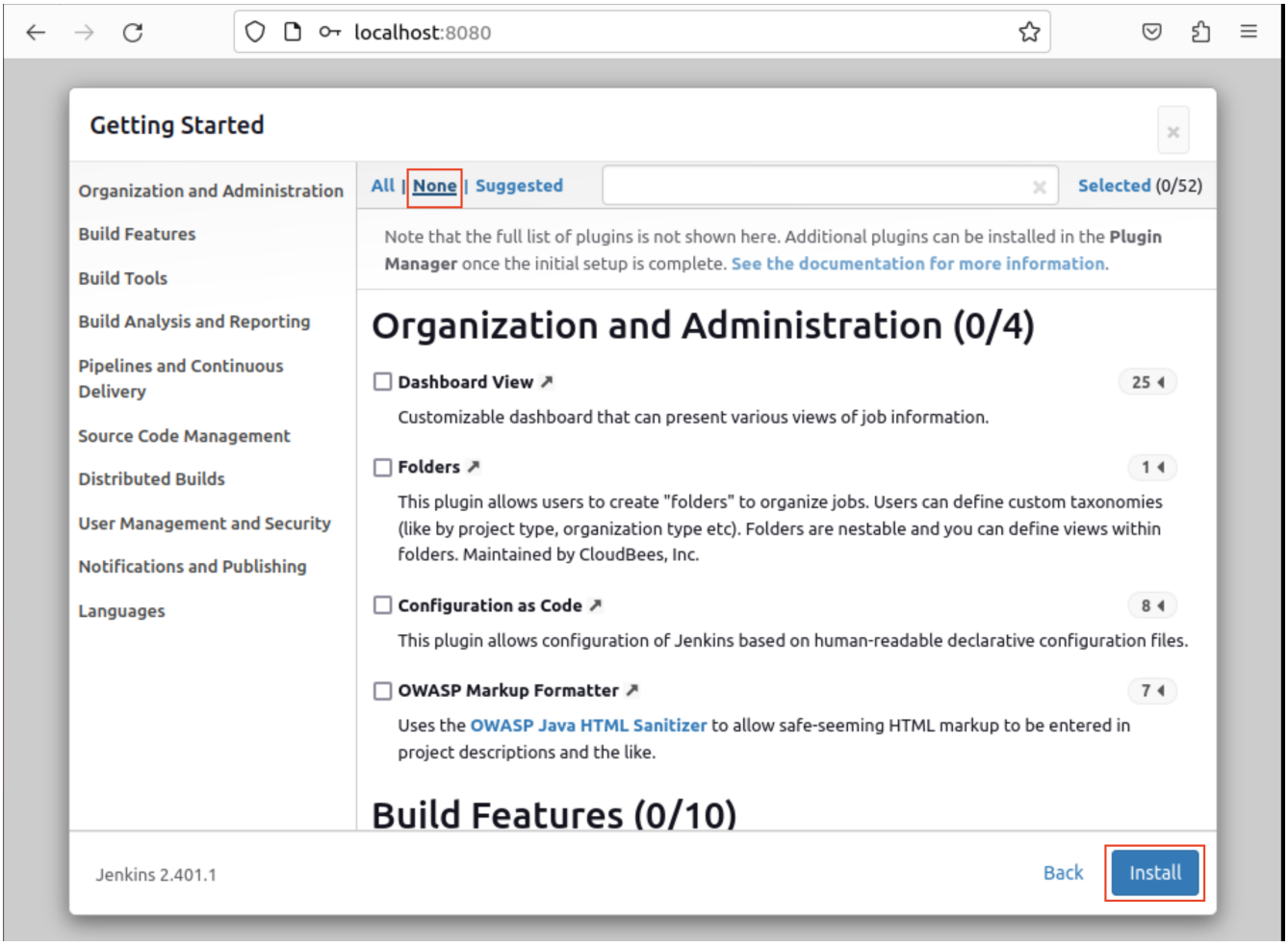
6. Wejdź na adres: http://localhost:8080/
7. Powinieneś otrzymać taką stronę

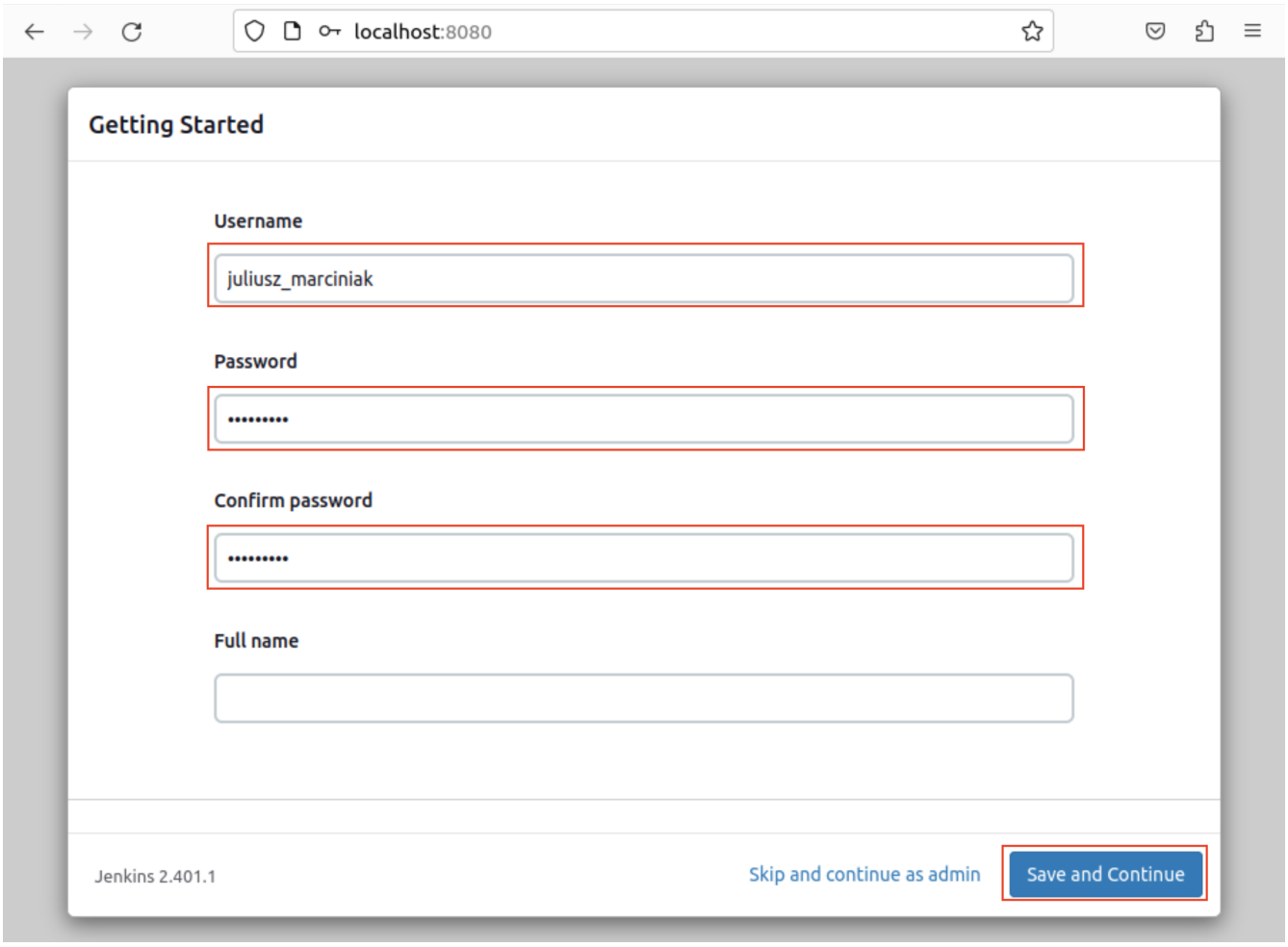


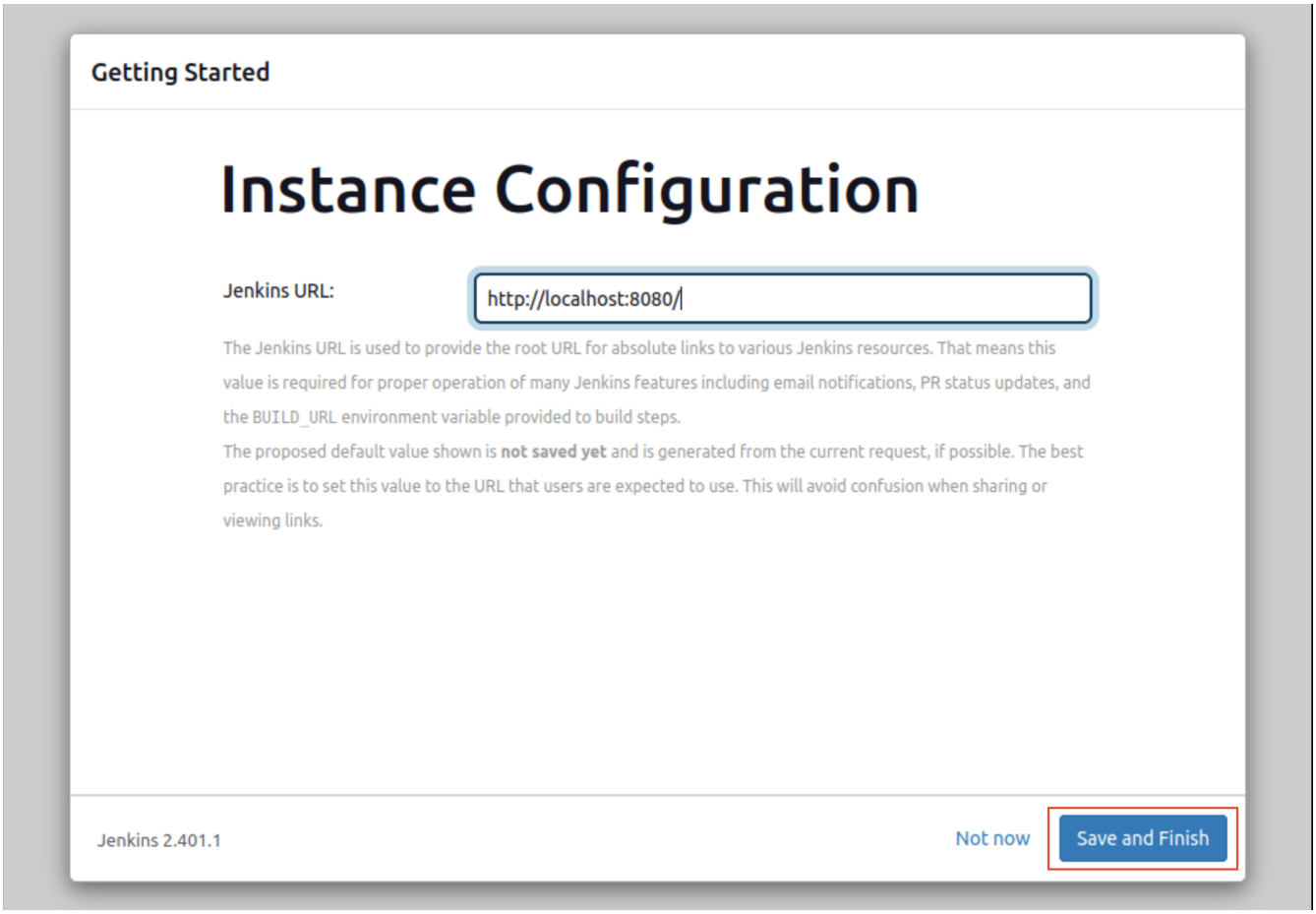
8. Znajdź hasło administratora komendą `sudo cat /var/lib/jenkins/secrets/initialAdminPassword`

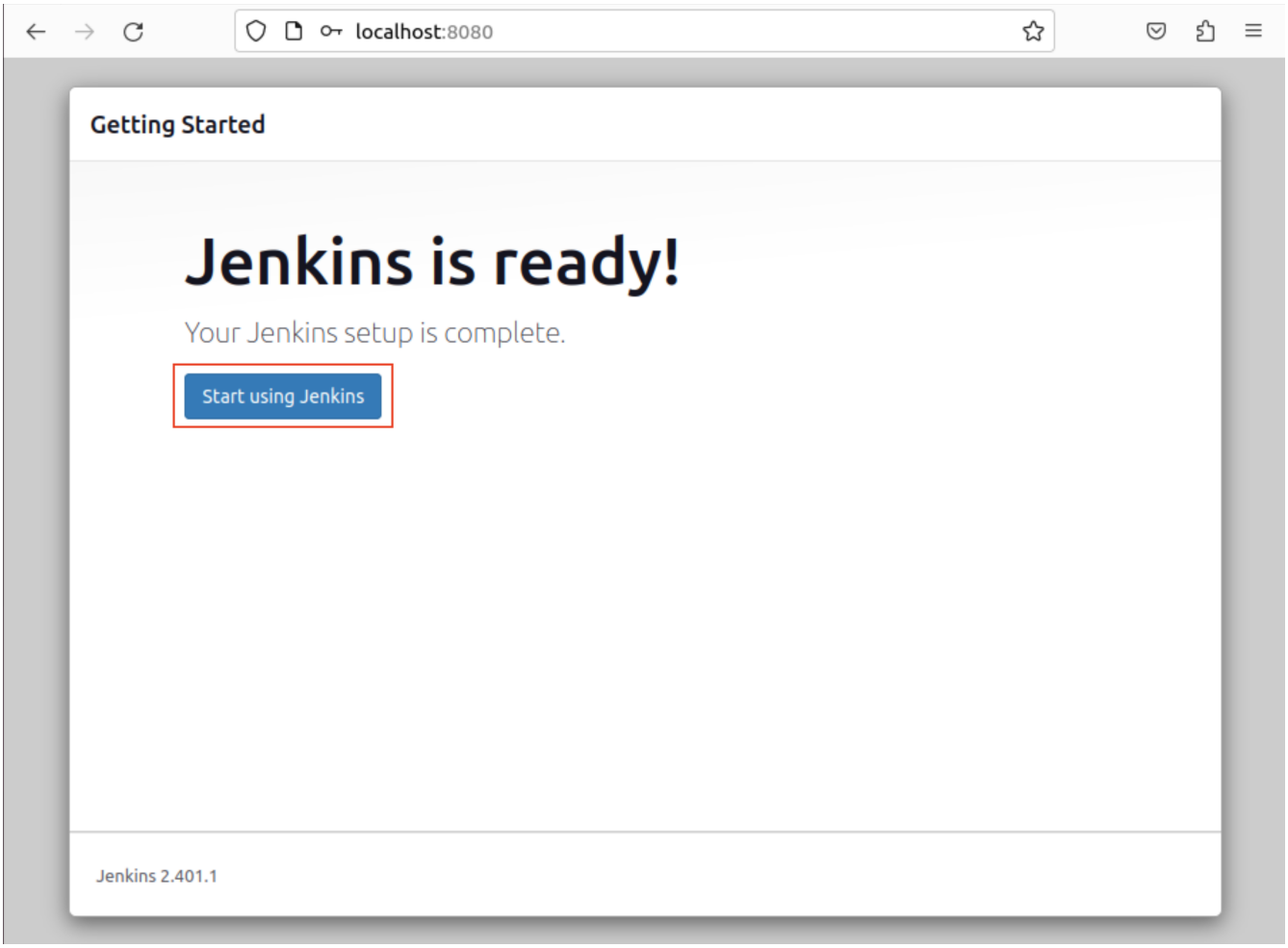
Pierwsza konfiguracja


localhost:8080
Getting Started
Customize Jenkins
Plugins extend Jenkins with additional features to support many different needs.
Install suggested plugins
Install plugins the Jenkins community finds most useful.
Select plugins to install
Select and install plugins most suitable for your needs.
Jenkins 2.401.1
localhost:8080/#


localhost:8080
Getting Started
Organization and Administration
Build Features
Build Tools
Build Analysis and Reporting
Pipelines and Continuous Delivery
Source Code Management
Distributed Builds
User Management and Security
Notifications and Publishing
Languages
All | None | Suggested
Selected (0/52)
Note that the full list of plugins is not shown here. Additional plugins can be installed in the Plugin Manager once the initial setup is complete. See the documentation for more information.
Organization and Administration (0/4)
Dashboard View
25
Customizable dashboard that can present various views of job information.
Folders
1
This plugin allows users to create "folders" to organize jobs. Users can define custom taxonomies (like by project type, organization type etc). Folders are nestable and you can define views within folders. Maintained by CloudBees, Inc.
Configuration as Code
8
This plugin allows configuration of Jenkins based on human-readable declarative configuration files.
OWASP Markup Formatter
7
Uses the OWASP Java HTML Sanitizer to allow safe-seeming HTML markup to be entered in project descriptions and the like.
Build Features (0/10)
Jenkins 2.401.1
Back
Install


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 jakiekolwiek 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 zadnie 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 zadnie 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']

Zdanie 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 <code>sh "curl -i https://httpbin.org/basic-auth/foo/bar"</code>
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 <code>curla</code> z autoryzacją <code>sh "curl -i -u 'user:pass' https://httpbin.org/basic-auth/foo/bar"</code>
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-cicd-jenkins-gitlab 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/agent`
   - Launch method: Launch agent via execution of command on the master
   - Launch command: `docker run -i --rm --name agent --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 sekcje `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?

GitLab

Opis

GitLab to opensourcowa platforma z wbudowaną kontrolą wersji do wytwarzania oprogramowania. Posiada takie funkcje jak narzędzia do code review, CI/CD, moduł zgłaszania błędów i wiele innych. Może być hostowana we własnym datacenter, w cloud albo używana u dostawcy. Wspiera również konteneryzacje.

Różnice względem Jenkinsa

- Jenkins jest jedynie narzędzie do CI/CD, a GitLab to cała platforma, od repozytorium kodu zaczynając
- Jenkins może być rozszerzany pluginami
- GitLab do pipeline wykorzystuje YAML, a Jenkins Grooviego
- GitLab jest prostszy i prawdopodobnie tańszy we wdrożeniu

Pipeline

Pipeline w GitLabie używają YAMLa, są zintegrowane z całym ekosystemem, więc za ich pomocą można robić naprawdę bardzo wiele rzeczy.

Przykładowy Pipeline Maven

```
# To contribute improvements to CI/CD templates, please follow the Development guide
at:
# https://docs.gitlab.com/ee/development/cicd/templates.html
# This specific template is located at:
# https://gitlab.com/gitlab-org/gitlab/-
/blob/master/lib/gitlab/ci/templates/Maven.gitlab-ci.yml

# Build JAVA applications using Apache Maven (http://maven.apache.org)
# For docker image tags see https://hub.docker.com/_/maven/
#
# For general lifecycle information see
https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html

# This template will build and test your projects
# * Caches downloaded dependencies and plugins between invocation.
# * Verify but don't deploy merge requests.
# * Deploy built artifacts from master branch only.

variables:
  # `showDateTime` will show the passed time in milliseconds. You need to specify `--
batch-mode` to make this work.
  MAVEN_OPTS: >-
    -Dhttps.protocols=TLSv1.2
```

```
    -Dmaven.repo.local=$CI_PROJECT_DIR/.m2/repository
    -Dorg.slf4j.simpleLogger.showDateTime=true
    -Djava.awt.headless=true

  # As of Maven 3.3.0 instead of this you MAY define these options in
`.mvn/maven.config` so the same config is used
  # when running from the command line.
  # As of Maven 3.6.1, the use of `--no-tranfer-progress` (or `-ntp`) suppresses
download and upload messages. The use
  # of the `Slf4jMavenTransferListener` is no longer necessary.
  # `installAtEnd` and `deployAtEnd` are only effective with recent version of the
corresponding plugins.
  MAVEN_CLI_OPTS: >-
    --batch-mode
    --errors
    --fail-at-end
    --show-version
    --no-transfer-progress
    -DinstallAtEnd=true
    -DdeployAtEnd=true

# This template uses the latest Maven 3 release, e.g., 3.8.6, and OpenJDK 8 (LTS)
# for verifying and deploying images
# Maven 3.8.x REQUIRES HTTPS repositories.
# See https://maven.apache.org/docs/3.8.1/release-notes.html#how-to-fix-when-i-get-a-
http-repository-blocked for more.
image: maven:3-openjdk-8

# Cache downloaded dependencies and plugins between builds.
# To keep cache across branches add 'key: "$CI_JOB_NAME"'
# Be aware that `mvn deploy` will install the built jar into this repository. If you
notice your cache size
# increasing, consider adding `-Dmaven.install.skip=true` to `MAVEN_OPTS` or in
`.mvn/maven.config`
cache:
  paths:
    - .m2/repository

# For merge requests do not `deploy` but only run `verify`.
# See https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html
.verify:
  stage: test
  script:
    - 'mvn $MAVEN_CLI_OPTS verify'
  except:
    variables:
      - $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH

# Verify merge requests using JDK8
verify:jdk8:
  extends:
```

```
    - .verify

# To deploy packages from CI, create a `ci_settings.xml` file
# For deploying packages to GitLab's Maven Repository: See
https://docs.gitlab.com/ee/user/packages/maven_repository/index.html#create-maven-
packages-with-gitlab-cicd for more details.
# Please note: The GitLab Maven Repository is currently only available in GitLab
Premium / Ultimate.
# For `master` or `main` branch run `mvn deploy` automatically.
deploy:jdk8:
  stage: deploy
  script:
    - if [ ! -f ci_settings.xml ]; then
        echo "CI settings missing\! If deploying to GitLab Maven Repository, please
see https://docs.gitlab.com/ee/user/packages/maven_repository/index.html#create-maven-
packages-with-gitlab-cicd for instructions.";
      fi
    - 'mvn $MAVEN_CLI_OPTS deploy --settings ci_settings.xml'
  only:
    variables:
      - $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH
```

Przykładowy Pipeline Docker

```
# To contribute improvements to CI/CD templates, please follow the Development guide
at:
# https://docs.gitlab.com/ee/development/cicd/templates.html
# This specific template is located at:
# https://gitlab.com/gitlab-org/gitlab/-
/blob/master/lib/gitlab/ci/templates/Docker.gitlab-ci.yml

# Build a Docker image with CI/CD and push to the GitLab registry.
# Docker-in-Docker documentation:
https://docs.gitlab.com/ee/ci/docker/using_docker_build.html
#
# This template uses one generic job with conditional builds
# for the default branch and all other (MR) branches.

docker-build:
  # Use the official docker image.
  image: docker:latest
  stage: build
  services:
    - docker:dind
  before_script:
    - docker login -u "$CI_REGISTRY_USER" -p "$CI_REGISTRY_PASSWORD" $CI_REGISTRY
  # Default branch leaves tag empty (= latest tag)
  # All other branches are tagged with the escaped branch name (commit ref slug)
  script:
    - |
```

```
      if [[ "$CI_COMMIT_BRANCH" == "$CI_DEFAULT_BRANCH" ]]; then
        tag=""
        echo "Running on default branch '$CI_DEFAULT_BRANCH': tag = 'latest'"
      else
        tag=":$CI_COMMIT_REF_SLUG"
        echo "Running on branch '$CI_COMMIT_BRANCH': tag = $tag"
      fi
    - docker build --pull -t "$CI_REGISTRY_IMAGE${tag}" .
    - docker push "$CI_REGISTRY_IMAGE${tag}"
  # Run this job in a branch where a Dockerfile exists
  rules:
    - if: $CI_COMMIT_BRANCH
      exists:
        - Dockerfile
```

Zadania

Podstawy obsługi

Zadanie 1. Stworzenie repozytorium

1. Załóż konto na https://gitlab.com
2. Stwórz repozytorium kodu
3. Pobierz repozytorium do lokalnego katalogu

Zadanie 2. Dodanie MR

1. Stwórz nowego brancha
2. Wypchnij go do repozytorium
3. Utwórz MR

Zadanie 3. Code review

1. Dodaj uczestników szkolenia do projektu jako developerów
2. Poproś uczestników o code review

Zadanie 4. Integracja ze Slack

1. Dodaj aplikacje GitLab do Slacka
2. Skonfiguruj aplikację w taki sposób, żeby wszystko wysyłała na publiczny kanał stworzony przez Ciebie

Zadanie 5. Dodanie projektu aplikacji

1. Wypchnij do repozytorium jakikolwiek projekt aplikacji (w taki języku, w jakim czujesz się swobodnie)

Zadanie 6. Opcje mergowania MR

1. Stwórz kilka nowych MR
2. Spróbuj je zmergować z różnymi opcjami
3. Jaki efekt uzyskałeś?

Instalacja i konfiguracja

Zadanie 1. Zainstaluj GitLab Runnera

1. Wykonaj następujące polecenia

```
curl -L "https://packages.gitlab.com/install/repositories/runner/gitlab-
runner/script.deb.sh" | sudo bash
sudo apt-get install gitlab-runner
```

Zadanie 2. Skonfiguruj 2 runnery

IMPORTANT	Zainstalowanie runnerów jako root (z `sudo` lub po zalogowaniu jako user root, np. `sudo -Hi`) nie wymaga ich uruchomienia poleceniem `gitlab-runner run`

1. Załóż konto na stronie gitlab.com
2. Utwórz repozytorium
3. Przejdź do konfiguracji repozytorium
4. Przejdź do zakładki CI/CD
5. Przejdź do Runners
6. Wyłącz shared runnery i dodaj 2 nowe runnery
    - Operating systems: Linux
    - Runner 1:
        - Tags: linux, shared
        - Run untagged jobs: true
    - Runner 2:
        - Tags: linux, docker
7. Postępuj zgodnie z instrukcją na ekranie
    - Runnera 1 uruchom jako `shell`
    - Runnera 2 uruchom jako `docker`

Pipeline

Zadanie 1. Uruchomienie pierwszego Pipeline

1. Stwórz w edytorze wizualnym plik `.gitlab-ci.yml`
2. Dodaj pierwszy pipeline

```
stages:
  - build

build-job:
  stage: build
  script:
    - echo "Hello World!"
```

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

Zadanie 2. Budowanie aplikacji Spring Boot

1. Ściągnij przykładowy projekt ze strony start.spring.io (Maven)
2. Zacommituj go do repozytorium
3. Zmień wywołanie `echo` na `mvn clean verify`
4. Sprawdź, czy pipeline wykonał się poprawnie

Zadanie 3. Budowanie aplikacji na konkretnym obrazie Dockerowym

1. Popraw poprzedni pipeline wskazując, z jakiego obrazu Dockerowego ma korzystać runner (możesz wskazać: `maven:3.9.2-eclipse-temurin-17`)
2. Otaguj `build-job`, żeby korzystał z Runnera dockerowego

Zadanie 4. Testy

1. Zmodyfikuj pipeline z poprzedniego zadania dodając do niego obsługę JUnit
2. Uruchom pipeline
3. Obejrzyj jak wyglądają testy JUnit
4. Dodaj test, który failuje
5. Uruchom pipeline
6. Zobacz, jak wyglądają wyniki w momencie, kiedy część testów nie przechodzi

Zadanie 5. Warunkowe wykonanie kroków

1. Dopisz krok, który wykona się tylko na branchu `dev`
2. Stwórz brancha `dev`
3. Sprawdź, czy krok się wykonał
4. Użyj instrukcji `when: manual`

5. Sprawdź efekt

Zdanie 6. Merge request

1. Stwórz krok, który wykona się w momencie stworzenia merge requesta — użyj do tego dokumentacji https://docs.gitlab.com/ee/ci/pipelines/merge_request_pipelines.html

Zadanie 7. Dodanie cache

1. Dopisz cachowanie folderu `.m2/repository`
2. Uruchom pipeline
3. Sprawdź, czy znowu były ściągane artefakty

Zadanie 8. Zapisanie artefaktu

1. Dopisz zapisywanie artefaktu
2. Sprawdź, czy artefakt się zapisał

Zadanie 9. Wypchnięcie obrazu dockerowego do registry

1. Zbuduj obraz dockerowy za pomocą `mvn clean spring-boot:build-image`
2. Wypchnij obraz do registry dockerowego zawartego w GitLab

Zadanie 10. Reużywanie jobów

1. Wynieś część joba do ukrytego joba
2. Używając `extend` doprowadź do takiego samego działania jak przed wyniesieniem

Zadanie 11. Remote pipeline

1. Wynieś cały pipeline do innego repozytorium
2. Za pomocą `include` wykonaj pipeline w pierwotnym repozytorium

Zadanie 12. Docker runner DIND

1. Używając tagów wymuś budowanie na runnerze dockerowym
2. Spróbuj zbudować pipeline z Zadania 9.

Nexus

Opis

Nexus – centralne miejsce do przechowywania artefaktów, które zostały zbudowane. Możemy przechowywać tam praktycznie każdy rodzaj oprogramowania, między innymi artefakty mavenowe, obrazy dockerowe, paczki Nugetowe.

Zastosowanie w CI/CD

Nexus będzie znakomicie się sprawdzał jako uzupełnienie narzędzia do budowania kodu, jakim jest Jenkins. Dzięki połączeniu tych narzędzi możemy wgrywać zbudowane przez Jenkinsa paczki do Nexusa. Dzięki temu mamy centralne miejsce z artefaktami. Następnie z Nexusa możemy pobierać obrazy/paczki w trakcie wdrażenia, czy to standalone, czy też na Kubernetesa.

Alternatywy

- JFrog Artifactory
- GitLab

Zadania

Instalacja i konfiguracja

Zadanie 1. Instalacja i uruchomienie Nexusa

1. Zainstaluj i uruchom Nexusa

Instalacja Nexusa w systemie Linux

1. Zainstaluj Javę za pomocą komend

```
sudo apt-get update
sudo apt-get install temurin-8-jdk
```

3. Zainstaluj Nexusa za pomocą komend

```
wget https://download.sonatype.com/nexus/3/nexus-3.55.0-01-unix.tar.gz
sudo tar -xvf nexus-3.55.0-01-unix.tar.gz
sudo mv nexus-3.55.0-01 /opt/nexus
sudo adduser nexus
sudo chown -R nexus:nexus /opt/nexus
sudo mkdir -p /opt/sonatype-work
sudo chown -R nexus:nexus /opt/sonatype-work
```

4. W pliku `/opt/nexus/bin/nexus.rc` zmień `#run_as_user=""` na `run_as_user="nexus"`
5. Utwórz plik `/etc/systemd/system/nexus.service` z zawartością

/etc/systemd/system/nexus.service

```
[Unit]
Description=nexus service
After=network.target
```

```
[Service]
Type=forking
LimitNOFILE=65536
User=nexus
Group=nexus
ExecStart=/opt/nexus/bin/nexus start
ExecStop=/opt/nexus/bin/nexus stop
User=nexus
Restart=on-abort
[Install]
WantedBy=multi-user.target
```

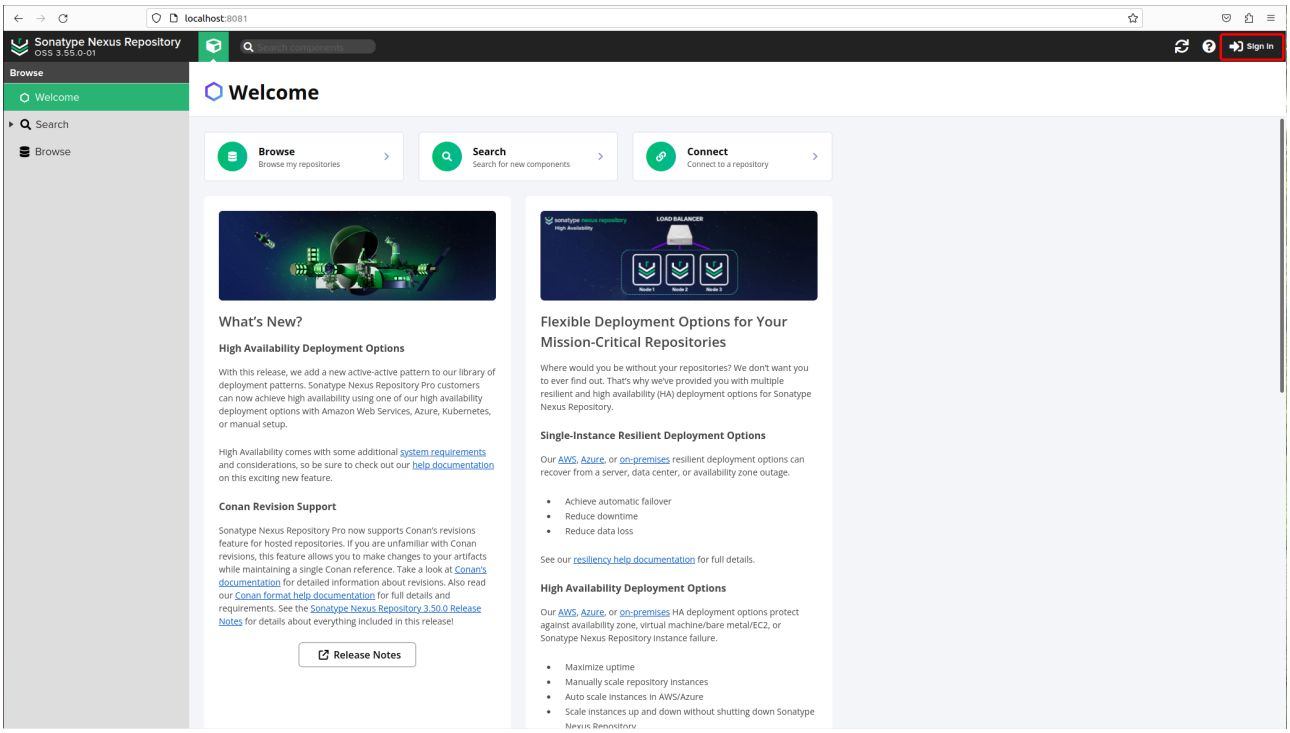
6. Uruchom Nexusa i ustaw automatyczne uruchamianie przy restarcie maszyny

```
sudo systemctl enable nexus
sudo systemctl start nexus
```

7. Sprawdź działanie Nexusa za pomocą komendy `sudo systemctl status nexus`. Oczekiwany rezultat

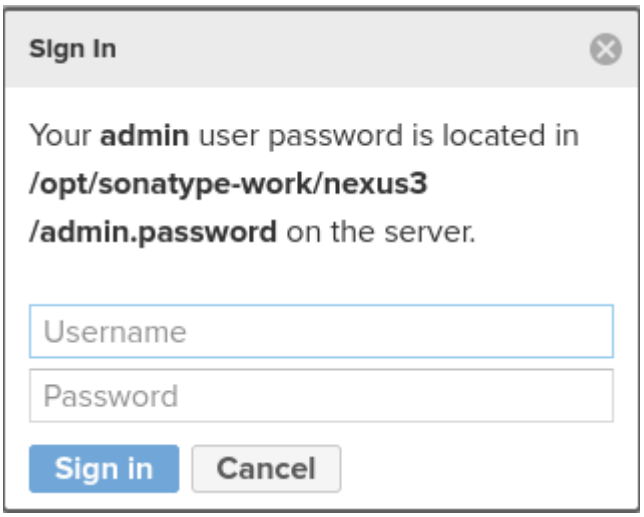
```
● nexus.service - nexus service
     Loaded: loaded (/etc/systemd/system/nexus.service; enabled; vendor preset:
enabled)
     Active: active (running) since Sun 2023-06-08 20:58:34 CEST; 18s ago
```

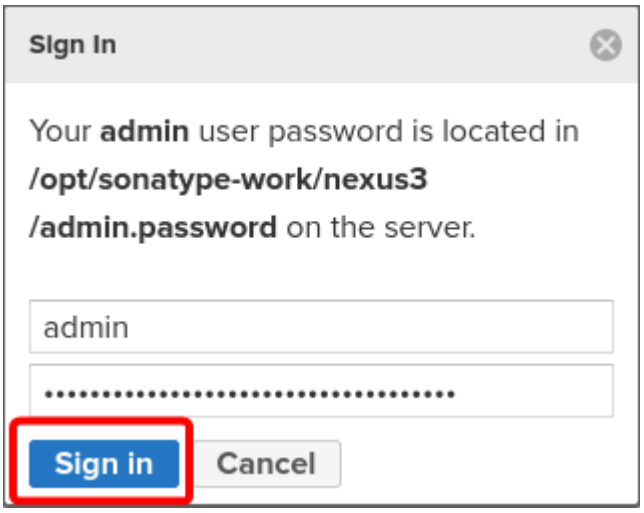
6. Wejdź na adres: http://localhost:8081/
7. Powinieneś otrzymać taką stronę

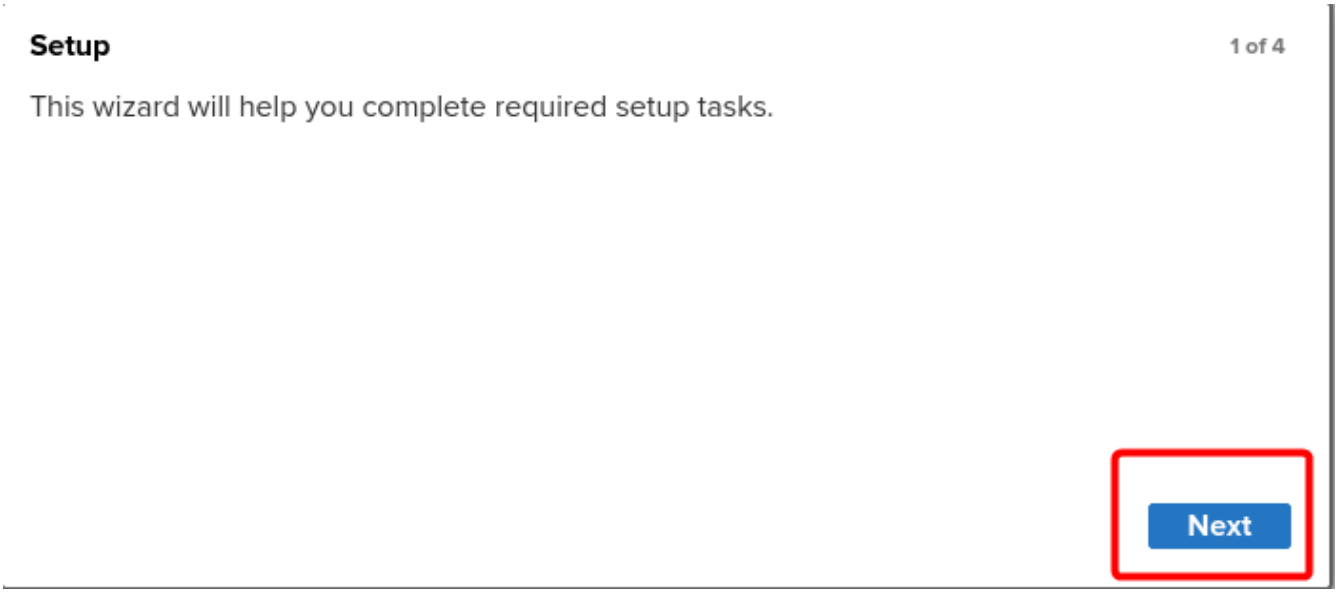


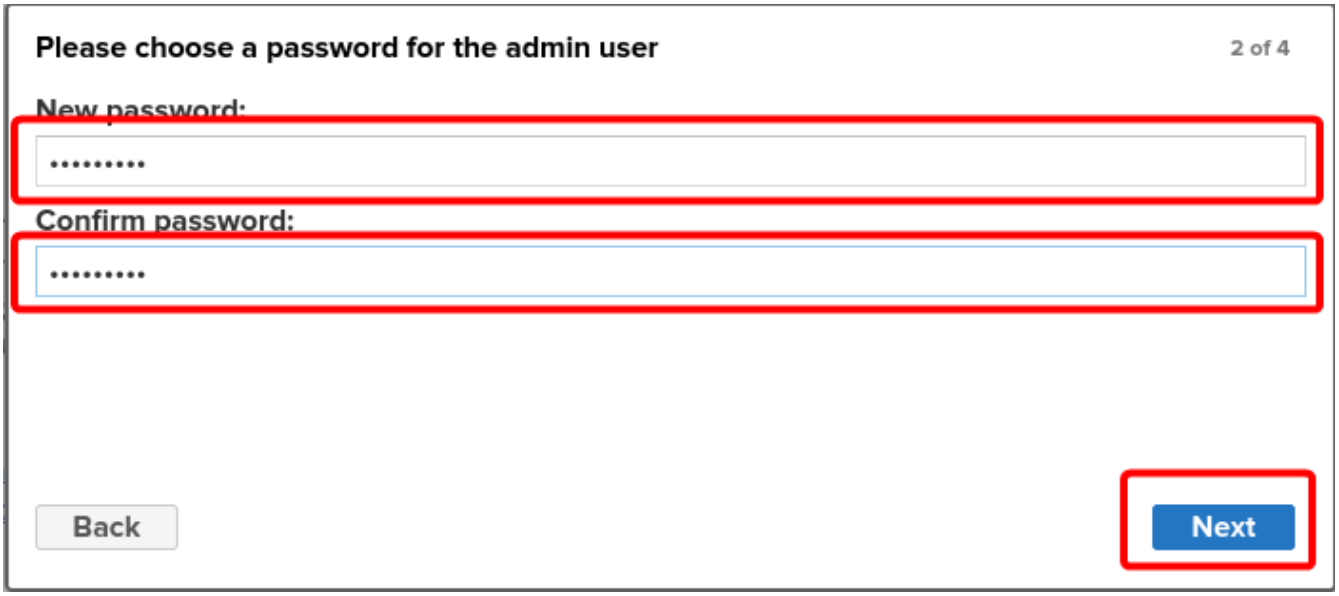
8. Znajdź hasło administratora komendą `sudo cat /opt/sonatype-work/nexus3/admin.password`

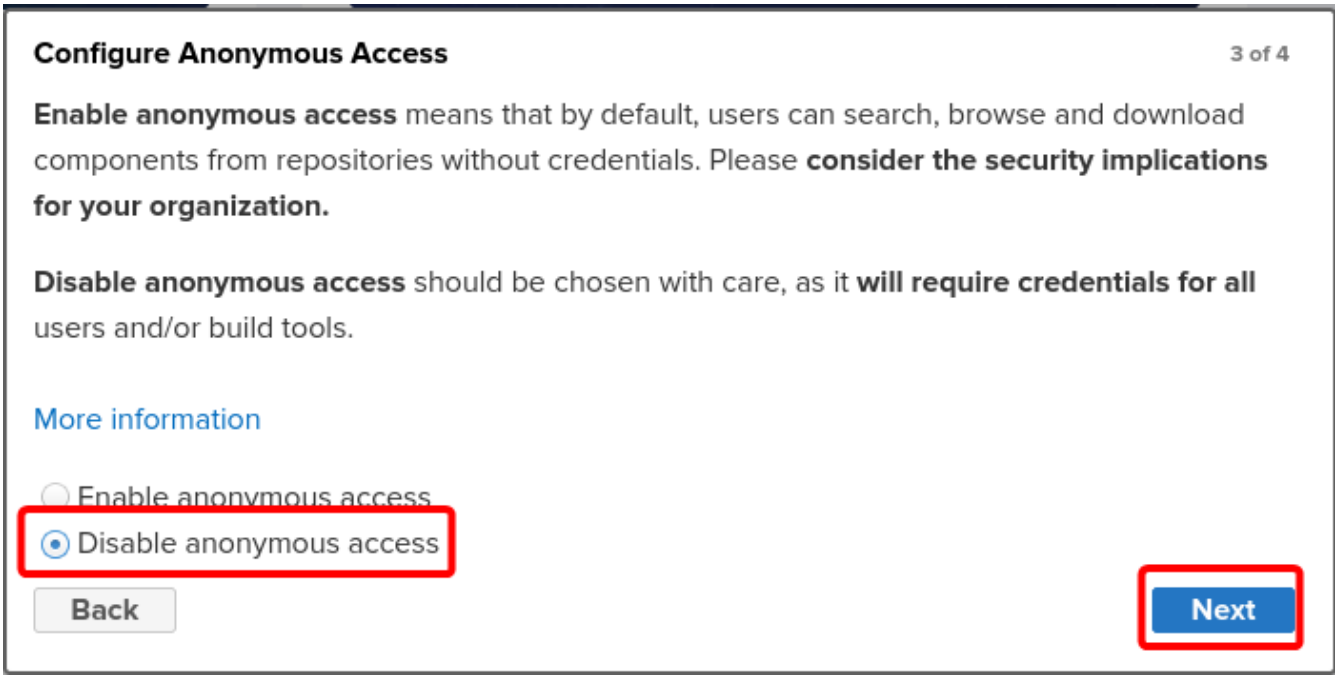
Pierwsza konfiguracja

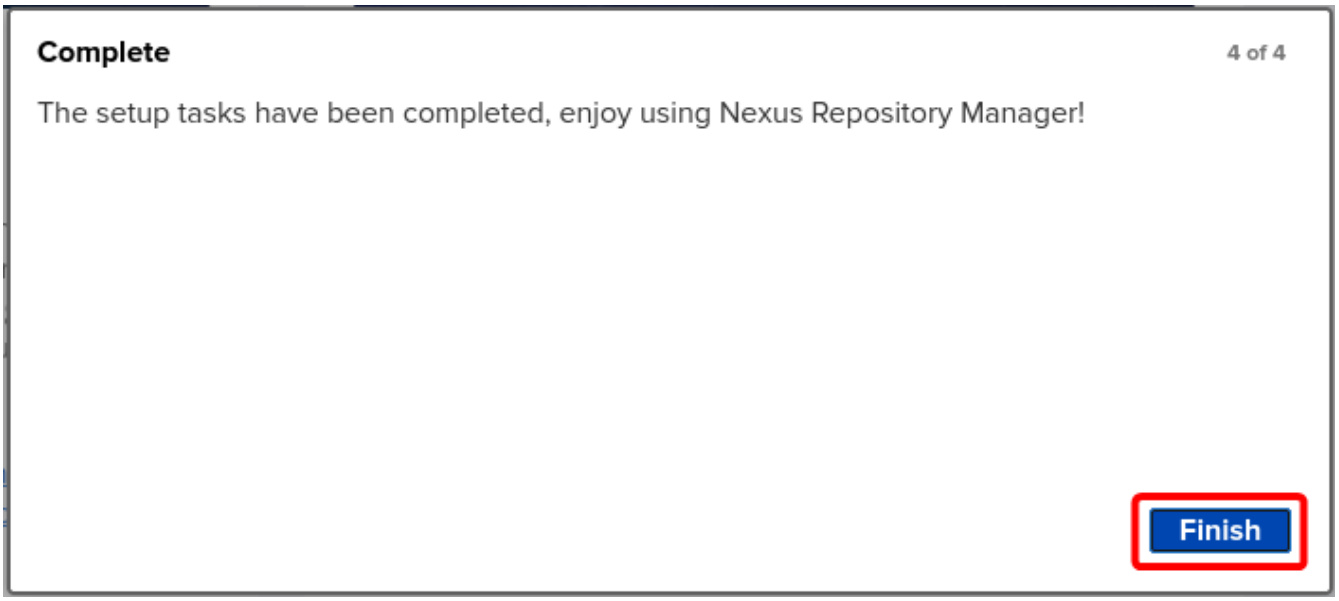












2. Stwórz użytkownika, którego będzie używał Jenkins, możesz dać mu rolę `nx-admin`

3. Za pomocą credentiali zapisz poświadczenia w Jenkinsie
4. Stwórz plik konfiguracyjny do Nexusa w Jenkinsie (settings.xml z Mavena)

Zadanie 2. Stwórz repozytoria w Nexus

1. Stwórz hostowane repozytorium Dockerowe
2. Stwórz hostowane repozytorium Mavena dla Snapshotów
3. Stwórz hostowane repozytorium Mavena dla Releasów

Zadanie 3. Wrzucenie obrazu Dockerowego do Nexusa

1. Zmodyfikuj pipeline z modułu Połączmy to z Dockerem Zadanie 2, żeby nie tylko obraz był budowany, ale również wrzucany do Nexusa
2. Użyj tego samego pluginu
3. Użyj instrukcji `withCredentials` do przekazania credentiali

Zadanie 4. Wydanie wersji aplikacji mavenowej

1. Będziemy bazować na forku, który stworzyłeś w ramach modułu Pipeline wewnątrz repozytorium Zadanie 1
2. Stwórz pipeline o nazwie `example-release`
3. Spróbuj napisać pipeline, który wyda wersje aplikacji mavenowej. Musisz pamiętać o:
    - Zmianie pom.xml (dodanie sekcji `scm` i `distributionManagement`)
    - Daniu uprawnień do pushowania do GitHuba Jenkinsowi
    - Dodaniu 2 stagów w pipeline:
        - Start release (`mvn -B release:prepare`)
        - Finish release(`mvn -B release:perform`)

Zadanie 5. Wydanie wersji aplikacji frontendowej

1. Utwórz podstawową aplikację Frontendową (np. Angularową)
2. Wrzuć ją do repozytorium
3. Stwórz od 0 pipeline do wydawania wersji
4. Dodaj parametr, w którym będzie można nadpisać wersję automatyczną