

# Laboratorium 5 - projekt, Tomasz Ziobrowski

---

## 1. Przygotowanie zadania

### Przygotowanie pipeline'a oraz SCM


Do pracy wykorzystano instancję Jenkins działającą jako kontener Dockera przygotowaną podczas poprzedniego zadania.

W panelu Jenkinsa utworzono nowy pipeline, który nazwano "ruru".

### Enter an item name


nazwa-

» Required field



#### Freestyle project

This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.



#### Pipeline

Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

W zaawansowanej sekcji ustawień pipeline'a dodano pobieranie skryptu (oraz innych plików) z SCM (Source Control Management), który w tym przypadku było sforkowane repozytrium projektu na [githubie](#). Ustawiono również gałąź, z której były zaciągane zmiany.

**Pipeline**

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

https://github.com/Ziobrowsky/ava

Credentials ?

- none - Add

Advanced...

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?

\*/main

Należało również podać ścieżkę do pliku Jenkinsfike, który również znajdował się w tym repozytorium.

Script Path ?

Jenkinsfile

☒ Lightweight checkout ?

[Pipeline Syntax](#)

## Zarys Jenkinsfile

W pliku jenkinsfile przygotowano etapy w bloku **steps**, które zostaną uruchomione podczas budowania ruociagu, oraz w bloku **parameters** będą znajdowały się parametry builda.

 Jenkinsfile  

```
pipeline {  
    agent any  
    parameters { ...  
    }  
    stages {  
        stage("Build") { ...  
        }  
        stage("Test") { ...  
        }  
        stage("Publish") { ...  
        }  
    }  
}
```

## Parametry builda

Każdy z parametrów budowania ma swój własny typ taki jak boolean, string, choice czy również password. W przypadku tego pipeline'a wybrano jedynie parametry typu bool, oraz string. Były one wykorzystane następnie do ustawień takich akcji jak publikowanie, numer wersji builda (major oraz minor) czy token do poratlu NPM. Każdy z parametrów ma format następujący:

```
<typ_parametru>(name: <nazwa_parametru>, defaultValue: <wartosc_domyślna>,  
description: <opis_parametru>)
```

Pole description jest opcjonalne i w przypadku nie podania go, zostanie wykorzystany jako label nazwa parametru.

```
parameters {  
    booleanParam(name: "PUBLISH", defaultValue: true, description: "Check  
to publish build to npm")  
    string(name: "VERSION_MAJOR", defaultValue: "1", description: "Major  
version of build to be published")  
}
```

```
string(name: "VERSION_MINOR", defaultValue: "0", description: "Minor
version of build to be published")
string(name: "TOKEN", defaultValue: "", description: "Provide npm
access token, else one in .env file is used")
}
```

Okno wprowadzania parametrów nieznacznie różni się w zależności od tego czy jest wykorzystywany interfejs Blueocean czy Jenkinsa standardowy do odpalenia budowania.

## Pipeline rurru

This build requires parameters:

☐ PUBLISH

Check to publish build to npm

VERSION\_MAJOR

1

Major version of build to be published

VERSION\_MINOR

0

Minor version of build to be published

TOKEN

Provide npm access token, else one in .env file is used

Build

## Input required

Check to publish build to npm

☐ PUBLISH

Major version of build to be published

1

Minor version of build to be published

0

Provide npm access token, else one in .env file is used

Run

Cancel

Przykładowe wykorzystanie parametru w kodzie wygląda następująco:

```
pipeline {
  parameters {
    string(name: "PARAM_1", defaultValue: "test")
  }
  stages {
    stage("Stage 1") {
      steps {
        echo "${params.PARAM_1}"
      }
    }
  }
}
```

## 2. Stage "Build"

Celem tego etapu było przygotowanie środowiska programu oraz jego instalacja.

W tym celu wykorzystano plik **Dockerfile**, który znajdował się również w repozytorium projektu a w Jenkinsie pod ścieżką **/ava/Dockerfile**.

Do projektu wykorzystano multi-stage Dockerfile, aby móc uniknąć tworzenia wielu plików dockera.

```
FROM node:latest AS build
RUN git clone https://github.com/Ziobrowskyy/ava.git
WORKDIR /ava
RUN npm install

FROM build AS test
WORKDIR /ava
CMD npm run test
```

Jak można zauważyć, jest to bardzo prosty skrypt, który nie wymaga zbyt wiele kroków.

W pliku **Jenkinsfile** jedynym krokiem było utworzenie obrazu dockera poleceniem **docker build --target build -t ava/build:latest .**, który przygotowywał obraz gotowego środowiska wraz z zainstalowanym programem. Obraz ten był później wykorzystany do kroku testowania oraz publish.

```
pipeline {
  (...)
  stages {
    stage("Build") {
      steps {
        sh "docker build --target build -t ava/build:latest ."
      }
    }
    (...)
  }
}
```

```
}
}
```

## Działający stage Build

Build - 2s

 [Restart Build](#)  

```
✓ Check out from version control <1s
1 The recommended git tool is: git
2 No credentials specified
3 > git rev-parse --resolve-git-dir /var/jenkins_home/workspace/ruru/.git # timeout=10
4 Fetching changes from the remote Git repository
5 > git config remote.origin.url https://github.com/Ziobrowskyy/ava # timeout=10
6 Fetching upstream changes from https://github.com/Ziobrowskyy/ava
7 > git --version # timeout=10
8 > git --version # 'git version 2.30.2'
9 > git fetch --tags --force --progress -- https://github.com/Ziobrowskyy/ava
+refs/heads/*:refs/remotes/origin/* # timeout=10
10 > git rev-parse refs/remotes/origin/main^{commit} # timeout=10
11 Checking out Revision 92d35862c5af6cdfff9e59613298a0f1b4588f8d (refs/remotes/origin/main)
12 > git config core.sparsecheckout # timeout=10
13 > git checkout -f 92d35862c5af6cdfff9e59613298a0f1b4588f8d # timeout=10
14 Commit message: "Fix wrong env variable name"
15 > git rev-list --no-walk b0f670f67b81ff8fb471ba6d91e019b9e32da78c # timeout=10

✓ docker build --target build -t ava/build:latest . -- Shell Script 2s
1 + docker build --target build -t ava/build:latest .
2 Sending build context to Docker daemon 742.9MB
3 Step 1/4 : FROM node:latest AS build
4 ----> 45aa5693e034
5 Step 2/4 : RUN git clone https://github.com/Ziobrowskyy/ava.git
6 ----> Using cache
7 ----> ae7a7bd46167
8 Step 3/4 : WORKDIR /ava
9 ----> Using cache
10 ----> 3b5d448b7726
11 Step 4/4 : RUN npm install
12 ----> Using cache
13 ----> 2080804c70df
14 Successfully built 2080804c70df
15 Successfully tagged ava/build:latest
```

## 3. Stage "Test"

Kod Jenkinsfile:

```
(...)
    stage("Test") {
        steps {
            sh "docker build --target test -t ava/test:latest ."
            sh "docker run ava/test:latest"
        }
    }
(...)
```



Kod Dockerfile:

```
FROM build AS test
WORKDIR /ava
```

```
CMD npm run test
```

W tym kroku wykorzystany jest obraz z kroku build. W fazie budowania następuje jedynie ustawienie odpowiedniego katalogu roboczego, zaś podczas `docker run` następuje wywołanie skryptu testującego aplikację poprzez `npm run test`.

## Działający stage Test

Test - 29s Restart Test  

✓ `docker build --target test -t ava/test:latest . -- Shell Script` 2s

```
1 + docker build --target test -t ava/test:latest .
2 Sending build context to Docker daemon 742.9MB
3 Step 1/7 : FROM node:latest AS build
4 ----> 45aa5693e034
5 Step 2/7 : RUN git clone https://github.com/Ziobrowskyy/ava.git
6 ----> Using cache
7 ----> ae7a7bd46167
8 Step 3/7 : WORKDIR /ava
9 ----> Using cache
10 ----> 3b5d448b7726
11 Step 4/7 : RUN npm install
12 ----> Using cache
13 ----> 2080804c70df
14 Step 5/7 : FROM build AS test
15 ----> 2080804c70df
16 Step 6/7 : WORKDIR /ava
17 ----> Using cache
18 ----> edbd75111129
19 Step 7/7 : CMD npm run test
20 ----> Using cache
21 ----> fb531616b8ec
22 Successfully built fb531616b8ec
23 Successfully tagged ava/test:latest
```

✓ `> docker run ava/test:latest -- Shell Script` 27s

```
1 + docker run ava/test:latest
2
3 > @ziobrowskyy/ava@1.0.0 test
4 > xo && tsd
```

## 4. Stage "Deploy"

Aplikacja wykorzystana do tego projektu, tj. Ava jest biblioteką, którą wykorzystuje się w projektach npm i służy do testowania aplikacji. Z tego względu nie ma możliwości uruchomienia jej w kroku `Deploy`, gdyż musiałaby być najpierw zainstalowana i dociągnięta jako dependency do innego projektu, a dopiero potem uruchomiona.

Z tego też względu w pipeline pominięto krok `Deploy`.

## 5. Stage "Publish"

Pełny kod Jenkinsfile dla tego kroku:

```
(...)
stage("Publish") {
```

```

        when {
            environment(name: "PUBLISH", value: "true")
        }
        agent {
            docker {
                image "ava/build:latest"
                args "-u root"
            }
        }
        steps {
            sh "git config user.email thomas.ziobrowski@gmail.com"
            sh "git config user.name Ziobrowskyy"
            script {
                if(params.TOKEN.equals("")) {
                    echo "Using .env file token"
                    load "$JENKINS_HOME/.env"
                    sh "echo
'//registry.npmjs.org/:_authToken=${NPM_TOKEN}' >> ~/.npmrc"
                } else {
                    echo "Using param token"
                    sh "echo
'//registry.npmjs.org/:_authToken=${params.TOKEN}' >> ~/.npmrc"
                }
            }
            sh "npm version
${params.VERSION_MAJOR}.${params.VERSION_MINOR}.${BUILD_NUMBER}"
            sh "npm publish --access public"
        }
    }
}
(...)

```

Do warunkowego publikowania stage wykorzystano parametr pipeline typu boolean **PUBLISH** oraz klauzulę **when**.

```

when {
    environment(name: "PUBLISH", value: "true")
}

```

Działa ona w ten sposób, że w momencie gdy jej zawartość zostanie zewaluowana na true, to tylko wtedy dany Stage jest uruchamiany. W tym przypadku sprawdzana jest wartość zmiennej **PUBLISH** czy jest równa **TRUE**.

Następnie jako **agent**, czyli środowisko na rzecz którego będą wykonywane komendy w **bloku steps** wykorzystano kontener dockera utworzony w kroku **Build**.

```

agent {
    docker {
        image "ava/build:latest"
        args "-u root"
    }
}

```



```
}
}
```

```
steps {
  // Zalogowanie się go konta git – potrzebne do kroku npm version
  sh "git config user.email thomas.ziobrowski@gmail.com"
  sh "git config user.name Ziobrowsky"
  // Sprawdzenie czy token npm został podany jako parametr pipeline
  // orz przygotowanie pliku .npmrc, potrzebnego do autoryzacji npm
  publish
  script {
    if(params.TOKEN.equals("")) {
      // Zaciągnięcie wartości zmiennej środowiskowej z pliku .env
      echo "Using .env file token"
      load "$JENKINS_HOME/.env"
      sh "echo '//registry.npmjs.org/:_authToken=${NPM_TOKEN}' >>
~/.npmrc"
    } else {
      // Wykorzystanie tokenu podanego jako parametr
      echo "Using param token"
      sh "echo '//registry.npmjs.org/:_authToken=${params.TOKEN}' >>
~/.npmrc"
    }
  }
  // Ustawienie wybranej wersji artefaktu
  sh "npm version
${params.VERSION_MAJOR}.${params.VERSION_MINOR}.${BUILD_NUMBER}"
  // Opublikowanie programu do npm
  sh "npm publish --access public"
}
```

Większość kroków, które wykonano w kolejnych krokach stage'a po krótko opisano jako komentarze powyżej. Całą procedurę publikowania do npm można podzielić na kilka etapów:

- ustawienie nazwy użytkownika oraz email git (`git config user.[email|name] <wartość>`), które są potrzebne do następnego ustawienia wersji aplikacji komendą `npm version`
- przygotowanie pliku `.npmrc`, który jest wymagany do autoryzacji użytkownika w npm. W ten sposób wiadomo, kto jest autorem danego repo do opublikowania. W celu autoryzacji wykorzystywany jest token npm.
  - aby wygenerować token, należy wejść w ustawienia konta na stronie (npm)[npmjs.com], a następnie wejść w zakładkę **Access tokens**. Wyświetlają się tam wygenerowane tokeny, oraz

opcja utworzenia nowego

## Access Tokens

[Generate New Token](#)[Delete Selected Tokens](#)

<input type="checkbox"/> Token	Type	Name	Created	Delete
<input type="checkbox"/> npm_6kz4.....kNlK	Publish	jenkins	20 hours ago	<input type="button" value="x"/>
<input type="checkbox"/> npm_0foT.....7kuu	Publish		20 hours ago	<input type="button" value="x"/>

- przy generowaniu tokenu należy podać nazwę, która umożliwi późniejszą jego identyfikację oraz wybrać prawa, które będą przyznane po zalogowaniu z wykorzystaniem tego tokenu. W naszym przypadku należy wybrać opcję **publish**, ponieważ będziemy chcieli opublikować nasz projekt do npm.

## New Access Token

Access tokens can be used instead of your password when using the npm CLI to download or publish packages.

Name

Select type

The type of access token defines its permissions. [Read more about types of access tokens.](#)

☐ Read-only

A read-only token can download public or private packages from the npm registry.

☐ Automation

An automation token will **bypass** two-factor authentication when publishing. If you have two-factor authentication enabled, you will not be prompted when using an automation token, making it suitable for CI/CD workflows.

☒ Publish

A publish token can read **and** publish packages to the npm registry. If you have two-factor authentication (2FA) enabled, it **will** be required when using this token.

[Generate Token](#)

- zakładając, że token jest dostępny jako zmienna należy przygotować plik **.npmrc** komendą **echo '//registry.npmjs.org/:\_authToken=\${NPM\_TOKEN}' >> ~/.npmrc**, która umieści adres registry npm oraz token autoryzacyjny w tym pliku
  - Oprócz przekazania tokenu jako parametru pipeline' jest możliwość użycia pliku **.env** w katalogu głównym Jenkinsa. Przygotowujemy w następujący sposób plik:

```
ava > ⚙️ .env
1 NPM_TOKEN="<token_npm>"
```

- następnie należy go wystać do kontenera Jenkinsa

```
anon@Tomaszs-MacBook-Air-2 ava % file .env
.env: ASCII text, with no line terminators
anon@Tomaszs-MacBook-Air-2 ava % docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
44b636997315   docker:dind                        "dockerd-entrypoint..." 20 hours ago   Up 20 hours   2375/tcp, 0.0.0.0:2376->2376/tcp
jenkins-docker  a4d68510e8f4   myjenkins-blueocean:2.332.3-1 "/sbin/tini -- /usr/"    20 hours ago   Up 20 hours   0.0.0.0:8080->8080/tcp, 0.0.0.0:50000->50000/tcp
jenkins-blueocean
anon@Tomaszs-MacBook-Air-2 ava % docker cp .env a4d:/var/jenkins_home/
```

- można go odczytać z pliku Jenkinsfile wykorzystując komendę `load <ściezka_do_pliku>`, a następnie wykorzystać jako zmienną środowiskową jak tu:

```
load "$JENKINS_HOME/.env" //zawartość pliku .env: NPM_TOKEN="<token>"
sh "echo '///registry.npmjs.org/:_authToken=${NPM_TOKEN}' >> ~/.npmrc"
```

- kolejnym krokiem będzie ustawienie odpowiedniej wersji programu do publikacji wykorzystując komendę `npm version <nr_wersji>`, gdzie numer wersji musi zawierać 3 cyfry oddzielone kropkami w formacie `<MAJOR>.<MINOR>.<FIX>`.
  - numer wersji major oraz minor pobrano jako parametry pipeline'a a jako ostatnią wartość użyto numer builda Jenkinsa jako `npm version ${params.MAJOR}.${params.MINOR}.${BUILD_NUMBER}`
- opublikowanie projektu do repozytorium npm komendą `npm publish --access public`
  - zakładając że token npm podany wcześniej jest prawidłowy oraz nastąpiła poprawnie autoryzacja użytkownika krok ten powinien przebiec prawidłowo
  - numer wersji opublikowanego programu nie może się różnić, ponieważ inaczej może nastąpić błąd
  - posiadając jedynie darmowe konto na npm należy publikować projekty jako publiczne z flagą `--access public`, inaczej wystąpi błąd w tym kroku

Działający stage Publish

Publish - 6s

 [Restart Publish](#)  

✓	> Check out from version control	<1s
✓	> Checks if running on a Unix-like node	<1s
✓	> docker inspect -f. "\$JD_TO_RUN" — Shell Script	<1s
✓	> git config user.email thomas.ziobrowski@gmail.com — Shell Script	<1s
✓	> git config user.name Ziobrowskyy — Shell Script	<1s
✓	> Using .env file token — Print Message	<1s
✓	> echo '//registry.npmjs.org/:_authToken=npm_6kz40Zr2avZOPwoJxHOYG1Kn9ZnCs1vkNlK' >> ~/.npmrc — Shell Script	<1s
✓	> npm version 1.0.85 — Shell Script	<1s
✓	> npm publish --access public — Shell Script	2s

Publish - 6s

 [Restart Publish](#)  

✓	✓ Check out from version control	<1s
	<pre> 1 The recommended git tool is: git 2 No credentials specified 3 &gt; git rev-parse --resolve-git-dir /var/jenkins_home/workspace/ruru@2/.git # timeout=10 4 Fetching changes from the remote Git repository 5 &gt; git config remote.origin.url https://github.com/Ziobrowskyy/ava # timeout=10 6 Fetching upstream changes from https://github.com/Ziobrowskyy/ava 7 &gt; git --version # timeout=10 8 &gt; git --version # 'git version 2.30.2' 9 &gt; git fetch --tags --force --progress -- https://github.com/Ziobrowskyy/ava +refs/heads/*:refs/remotes/origin/* # timeout=10 10 &gt; git rev-parse refs/remotes/origin/main^{commit} # timeout=10 11 Checking out Revision 92d35862c5af6cdfff9e59613298a0f1b4588f8d (refs/remotes/origin/main) 12 &gt; git config core.sparsecheckout # timeout=10 13 &gt; git checkout -f 92d35862c5af6cdfff9e59613298a0f1b4588f8d # timeout=10 14 Commit message: "Fix wrong env variable name"</pre>	
✓	✓ Checks if running on a Unix-like node	<1s
✓	✓ docker inspect -f. "\$JD_TO_RUN" — Shell Script	<1s
	<pre> 1 + docker inspect -f . ava/build:latest 2 .</pre>	
✓	✓ git config user.email thomas.ziobrowski@gmail.com — Shell Script	<1s
	<pre> 1 + git config user.email thomas.ziobrowski@gmail.com</pre>	
✓	✓ git config user.name Ziobrowskyy — Shell Script	<1s
	<pre> 1 + git config user.name Ziobrowskyy</pre>	
✓	✓ Using .env file token — Print Message	<1s
	<pre> 1 Using .env file token</pre>	
✓	✓ echo '//registry.npmjs.org/:_authToken=npm_6kz40Zr2avZOPwoJxHOYG1Kn9ZnCs1vkNlK' >> ~/.npmrc — Shell Script	<1s
	<pre> 1 + echo '//registry.npmjs.org/:_authToken=npm_6kz40Zr2avZOPwoJxHOYG1Kn9ZnCs1vkNlK'</pre>	
✓	✓ npm version 1.0.85 — Shell Script	<1s
	<pre> 1 + npm version 1.0.85 2 v1.0.85</pre>	

```

✓  npm publish --access public  — Shell Script 2s

1  + npm publish --access public
2  npm notice
3  npm notice package: @ziobrowskyy/ava@1.0.85
4  npm notice === Tarball Contents ===
5  npm notice 61B    entrypoints/cli.mjs
6  npm notice 3.1kB  entrypoints/eslint-plugin-helper.cjs
7  npm notice 66B    entrypoints/main.cjs
8  npm notice 48B    entrypoints/main.mjs
9  npm notice 68B    entrypoints/plugin.cjs
10 npm notice 123B   entrypoints/plugin.mjs
11 npm notice 386B   index.d.ts
12 npm notice 10.2kB lib/api.js
13 npm notice 24.8kB lib/assert.js
14 npm notice 351B   lib/chalk.js
15 npm notice 14.5kB lib/cli.js
16 npm notice 1.3kB  lib/code-excerpt.js

70 npm notice 1.5kB  types/try-fn.d.ts
71 npm notice === Tarball Details ===
72 npm notice name:    @ziobrowskyy/ava
73 npm notice version: 1.0.85
74 npm notice filename: @ziobrowskyy/ava-1.0.85.tgz
75 npm notice package size: 68.2 kB
76 npm notice unpacked size: 261.7 kB
77 npm notice shasum:   66b78936c541077acdd5e5f15064eecf2451ec21
78 npm notice integrity: sha512-J86esJG14EmZP[...]aSjQl0y7ekmpg==
79 npm notice total files: 66
80 npm notice
81 npm notice Publishing to https://registry.npmjs.org/
82 + @ziobrowskyy/ava@1.0.85

```

## Działający build pipeline'a

Gotowy plik Jenkinsfile:

```

pipeline {
    agent any
    parameters {
        booleanParam(name: "PUBLISH", defaultValue: true, description:
"Check to publish build to npm")
        string(name: "VERSION_MAJOR", defaultValue: "1", description:
"Major version of build to be published")
        string(name: "VERSION_MINOR", defaultValue: "0", description:
"Minor version of build to be published")
        string(name: "TOKEN", defaultValue: "", description: "Provide npm
access token, else one in .env file is used")
    }
    stages {
        stage("Build") {
            steps {
                sh "docker build --target build -t ava/build:latest ."
            }
        }
        stage("Test") {
            steps {
                sh "docker build --target test -t ava/test:latest ."
                sh "docker run ava/test:latest"
            }
        }
    }
}

```

```
stage("Publish") {

    when {
        environment(name: "PUBLISH", value: "true")
    }
    agent {
        docker {
            image "ava/build:latest"
            args "-u root"
        }
    }
    steps {
        sh "git config user.email thomas.ziobrowski@gmail.com"
        sh "git config user.name Ziobrowskyy"
        script {
            if(params.TOKEN.equals("")) {
                echo "Using .env file token"
                load "$JENKINS_HOME/.env"
                sh "echo
'//registry.npmjs.org/:_authToken=${NPM_TOKEN}' >> ~/.npmrc"
            } else {
                echo "Using param token"
                sh "echo
'//registry.npmjs.org/:_authToken=${params.TOKEN}' >> ~/.npmrc"
            }
        }
        sh "npm version
${params.VERSION_MAJOR}.${params.VERSION_MINOR}.${params.BUILD_NUMBER}"
        sh "npm publish --access public"
    }
}
}
```

- wraz z publish

✓ **ruru** < 85

Pipeline

Changes

Tests

Artifacts

Logout

Branch: —

39s

Changes by Tomasz Ziobrowski

Commit: —

14 hours ago

Started by user Tomasz Ziobrowski



Publish - 6s

[Restart Publish](#)

✓	> Check out from version control	<1s
✓	> Checks if running on a Unix-like node	<1s
✓	> docker inspect -f . "\$JD_TO_RUN" — Shell Script	<1s
✓	> git config user.email thomas.ziobrowski@gmail.com — Shell Script	<1s
✓	> git config user.name Ziobrowskyy — Shell Script	<1s
✓	> Using .env file token — Print Message	<1s
✓	> echo '//registry.npmjs.org/:_authToken=npm_6kz40Zr2avZOPwoJxHOYG1Kn9ZnCs1vkNIK' >> ~/.npmrc — Shell Script	<1s
✓	> npm version 1.0.85 — Shell Script	<1s
✓	> npm publish --access public — Shell Script	2s

- bez publish



## Input required

Check to publish build to npm

☐ PUBLISH

Major version of build to be published

1

Minor version of build to be published

0

Provide npm access token, else one in .env file is used

Run

Cancel



✓ ruru < 86 >
Pipeline
Changes
Tests
Artifacts
↺
⚙️
🔗
Logout
✕

Branch: — 43s Changes by Tomasz Ziobrowski  
Commit: — a few seconds ago Started by user Tomasz Ziobrowski



Publish

[Restart Test](#)
🔗
⬇️

No steps This stage has no steps

```

1 Started by user Tomasz Ziobrowski
2 Obtained Jenkinsfile from git https://github.com/Ziobrowskyy/ava
3 [Pipeline] Start of Pipeline
4 [Pipeline] node
5 Running on Jenkins in /var/jenkins_home/workspace/ruru
6 [Pipeline] {
7 [Pipeline] stage
8 [Pipeline] { (Declarative: Checkout SCM)
9 [Pipeline] checkout
10 The recommended git tool is: git
11 No credentials specified
12 > git rev-parse --resolve-git-dir /var/jenkins_home/workspace/ruru/.git # timeout=10
13 Fetching changes from the remote Git repository
14 > git config remote.origin.url https://github.com/Ziobrowskyy/ava # timeout=10
15 Fetching upstream changes from https://github.com/Ziobrowskyy/ava
16 > git --version # timeout=10
  
```

## 6. Napotkane problemy

- Jednym z natrętnie mnie spotykających problemów podczas realizacji tego projektu była konieczność pamiętania, czy podczas uruchomienia Jenkinsa, działa również kontener Docker. Error'y, które się w przeciwnym wypadku pojawiały były bardzo niepomocne i sprawiały wiele problemów, ponieważ ciężko było je wyszukać w internecie :^( Dopiero w momencie poddania się i reinsatacji Jenkinsa na nowo odkryłem, że właśnie dockerowy dind był źródłem kłopotów i frustracji.
  - zmarnowany czas: 4-5h
- W przypadku Jenkinsa i paramterów budowania, od roku 2018 nie potrafią naprawić aktualizacji tychże paramterów. Kilukrotne odpalenie pipeline'a nie rozwiązywało problemu z cachowaniem poprzednich parametów. Tak samo uruchamianie ponowne Jenkinsa nie przyniosło spodziewanych rezultatów. Rozwiązanie było takie, że należało w interfejsu Blueocean przejść do standardowego, tam odpalić jeszcze raz build (a czaem nie trzeba było) i magicznie pojawiały się już zaktualizowane parametry (???). Po odpaleniu builda ze standardowego UI i przejściu z powrotem na Blueocean tam również były one zaktualizowane.
  - zmarnowany czas: 2h