

Projekt pipeline Jenkins

Wszystkie pliki Jenkinsfile, Dockerfile oraz logi Jenkinsa znajdują się w repozytorium wraz z tym plikiem sprawozdania.

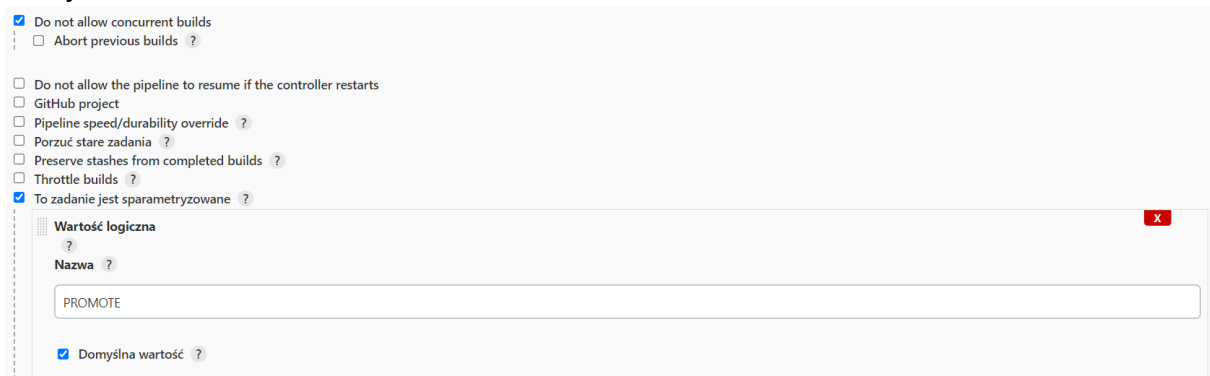
Podczas realizowania projektu z Jenkinsa pojawiły się problemy z poprzednio wybranym repozytorium, dlatego zdecydowano się na zmianę projektu na inny również wykorzystujący Node. Również pojawiły się problemy związane z Jenkinsem, który nagle napotykał błędy podczas fetchowania plików Jenkinsfile i Dockerfile z Gita co uniemożliwiało poprawne działanie pipeline'u i finalne opóźnienia związane z wysłaniem tego projektu.

1. Cel projektu

Celem projektu było utworzenie w pełni działającego automatycznego pipeline'u, który buduje dany kod, testuje go, a następnie odtwarza i publikuje. Wszystkie te etapy zostały przeprowadzone za pomocą konteneryzacji Dockerem, którego obsługiwał Jenkins.

2. Przygotowanie pipeline'u

Utworzony pipeline brał pliki Jenkinsfile oraz pozostałe Dockerfile z prywatnego repozytorium na Githubie. Pipeline miał zaznaczone opcje „Do not allow concurrent builds” oraz „To zadanie jest sparametryzowane” zawierającą parametr *PROMOTE* będący wartością logiczną, który służył do określenia czy dana kompilacja ma zostać promowana do nowej wersji.



☒ Do not allow concurrent builds
☐ Abort previous builds ?

☐ Do not allow the pipeline to resume if the controller restarts
☐ GitHub project
☐ Pipeline speed/durability override ?
☐ Porzuć stare zadania ?
☐ Preserve stashes from completed builds ?
☐ Throttle builds ?
☒ To zadanie jest sparametryzowane ?

Wartość logiczna ?
Nazwa ?
PROMOTE

☒ Domyslna wartość ?

3. Kroki pipeline'u

- Ustalenie parametrów oraz Prebuild

```
agent any
parameters {
    booleanParam(name: 'PROMOTE', defaultValue: true, description: '')
}
stages {
    stage("Prebuild") {
        steps {
            script {
                sh 'mkdir -p shared_dir'
            }
        }
        post {
            success {
                sh 'echo Prebuild success'
            }
            failure {
                sh 'echo Prebuild failure'
            }
        }
    }
}
```

Jenkinsfile Prebuild

Podany fragment Jenkinsfile ustanawia parametr PROMOTE jako parametr typu logicznego oraz nadaje mu wartość domyślną na prawdę. Następnie rozpoczynają się już kroki naszego pipeline'u – rozpoczynamy od etapu Prebuild służącego do utworzenia folderu który posłuży jako wolumin wyjściowy i wejściowy dla kontenerów. Dla każdego etapu wypisywany jest również *success* w przypadku powodzenia kroku lub *failure* w przypadku niepowodzenia.

- Build

```
stage('Build') {
    steps {
        script {
            sh '''
            echo Build

            docker build -f dockerBuilder -t nodejs.org . --no-cache
            docker run -v \$(pwd)/shared_dir:/docker_volume nodejs.org
            '''
        }
    }
    post {
        success {
            sh 'echo Build success'
        }
        failure {
            sh 'echo Build failure'
        }
    }
}
```

Jenkinsfile Build

```
1 FROM node
2 RUN git clone https://github.com/nodejs/nodejs.org.git
3 WORKDIR /nodejs.org/
4 RUN npm install && npm run build
5 RUN npm pack
6 RUN mkdir /artifacts
7 RUN cp nodejs.org-*.tgz /artifacts
8 CMD cp nodejs.org-*.tgz /docker_volume
```

Dockerfile dockerBuilder

Kolejnym etapem jest Build, który buduje obraz za pomocą Dockera i poprzednio przygotowanego Dockerfile'a. Budowanie zawiera opcje no-cache, aby każdy etap tworzenia obrazu był uruchamiany za każdym uruchomieniem pipeline'a. Obraz jest tworzony poprzez obraz Node, następnie klonowane jest na niego repozytorium z wybranym projektem, później instalowane są dependencje oraz uruchamiany jest build. W przypadku powodzenia zbudowany kod jest pakowany do pliku .tgz poprzez polecenie *npm pack*. Następnie artefakt kopiowany jest do specjalnego folderu oraz przy uruchomieniu kontenera kopiowany jest do folderu do którego powinien być podpięty wolumin. Po utworzeniu obrazu uruchamiany jest kontener wraz z woluminem, którego folder został utworzony podczas etapu Prebuild.

- Test

```
stage('Test') {
    steps {
        script {
            sh '''
                echo Test

                docker build . -f dockerTester -t nodejs.org-test
                docker run nodejs.org-test
            '''
        }
    }
    post {
        success {
            sh 'echo Tests success'
        }
        failure {
            sh 'echo Tests failure'
        }
    }
}
```

Jenkinsfile Test

```
1 FROM nodejs.org:latest
2 WORKDIR /nodejs.org/
3 RUN npm run test
```

Dockerfile dockerTester

Etap Test budował obraz na podstawie obrazu Build, a następnie wykonywał testy za pomocą *npm run test*. W przypadku powodzenia testów wypisany był *success*, a w przeciwnym wypadku *failure*.

- Deploy

```
stage('Deploy') {
    steps {
        script {
            sh '''
                echo Deploy

                docker build . -f dockerDeploy -t nodejs.org-deploy
                docker run -d -v \$(pwd)/shared_dir:/docker_volume nodejs.org-deploy
            '''
        }
    }
    post {
        success {
            sh 'echo Deploy success'
        }
        failure {
            sh 'echo Deploy failure'
        }
    }
}
```

Jenkinsfile Deploy

```
1 FROM nodejs.org:latest
2 RUN mkdir /nodejs.org-deploy
3 RUN cp /artifacts/nodejs.org-*.tgz /nodejs.org-deploy/
4 RUN tar -xzf /nodejs.org-deploy/nodejs.org-*.tgz -C /nodejs.org-deploy
5 WORKDIR /nodejs.org-deploy/package/
6 RUN npm install
7 EXPOSE 8080
8 CMD ["npm", "start"]
```

Dockerfile dockerDeploy

Po poprawnym wykonaniu Build oraz Test dana aplikacja została uruchamiana. Budowany był obraz na podstawie obrazu Build, który w celu sprawdzenia poprawności utworzonego artefaktu rozpakowywał go i na jego podstawie instalował dependencje. Następnie publikowany był port 8080 na którym działa aplikacja. Podczas uruchomienia kontenera wykonywane było polecenie *npm start* uruchamiające aplikację.

```
• Publish
stage('Publish') {
  when {
    {
      expression {return params.PROMOTE}
    }
  }
  steps {
    script {
      archiveArtifacts artifacts: 'shared_dir/nodejs.org-*.tgz', fingerprint: true
    }
  }
}
```

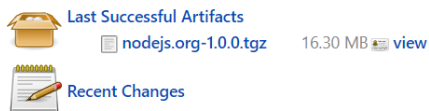
Jenkinsfile Publish

Podczas kroku Publish w przypadku powodzenia wszystkich poprzednich oraz gdy parametr *PROMOTE* miał wartość logiczną prawdę, wtedy publikowany był artefakt zbudowany podczas kroku Build. W przypadku gdy parametr *PROMOTE* miał wartość logiczną fałsz to dany krok się nie wykonywał.

4. Wynik działania pipeline'u

Poniżej widać wynik uruchomienia pipeline'u – dla wersji 2 z parametrem *PROMOTE* ustawionym na prawdę, a w przypadku 3 na fałsz co możemy zauważyć przez niewykonanie kroku Publish. Dla wersji drugiej mamy możliwość pobrania artefaktu zbudowanego podczas uruchomienia tej wersji pipeline'u, ponieważ krok Publish został wykonany.

Pipeline projekt_devops



Stage View

		Declarative: Checkout SCM	Prebuild	Build	Test	Deploy	Publish
Average stage times: (Average full run time: ~6min 20s)		1s	900ms	3min 7s	1min 25s	1min 40s	1s
#3 Jun 03 18:16 No Changes		736ms	901ms	1min 43s	1min 23s	1min 26s	
#2 Jun 03 18:11 1 commit		1s	882ms	1min 53s	1min 18s	1min 17s	956ms

Diagram aktywności:

