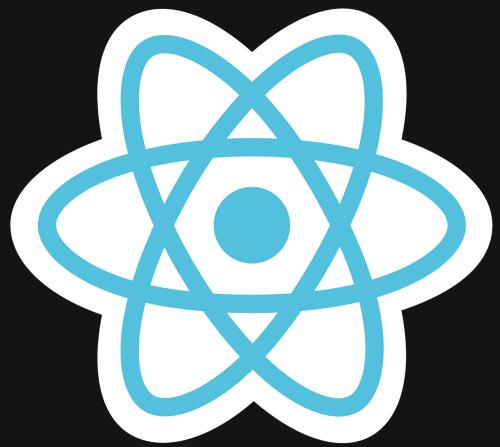


# Deployment aplikacji na serwer zdalny

28 DZIEŃ REACT'A



# Deployment

## WDRAŻANIE APLIKACJI

Deployment (wdrażanie) aplikacji oznacza tyle, że wszystkie żądania (requesty) przeglądarki użytkownika muszą być dostępne dla przeglądarki - wszystkie pliki javascript, wszelkie niestandardowe czcionki, obrazy, arkusze stylów itp., których używamy w naszej aplikacji, muszą być dostępne na publicznie dostępnym serwerze, aby aplikacja poprawnie (o ile w ogóle) działała.



# Build aplikacji

Ostatnio używany Webpack zajmuje się build'owaniem i spakowaniem całej naszej aplikacji, w celu wrzucenia na chmurę i jej poprawnego działania.

Obejmuje to wszelkie tokeny klienta i naszą konfigurację produkcyjną.

Jedynie co potrzebujemy do wysłania na zdalny serwer to zawartość pakietu dystrybucyjnego Webpack, czyli katalog build.

Pakujemy naszą aplikację komendą npm run build w terminalu.

```
npm run build
```

DEPLOYMENT

3

# Przykładowe strony umożliwiające hosting

My skupimy się na Surge, Github pages oraz Heroku.

4

DEPLOYMENT

SURGE.SH

PAGES.GITHUB.COM

WWW.HEROKU.COM

AWS.AMAZON.COM/S3

GETFORGE.COM

APP.NETLIFY.COM

WWW.PANCAKE.IO

# Surge.sh

Surge is a static web publishing for Front-End Developers and permitted you publish your project without leaving the command line.

## INSTALLATION

Instalujemy surge komendą

```
npm install --global surge
```

Musimy stworzyć/zalogować się na surge -> wpisujemy w terminalu `surge` i postępujemy według komunikatów.

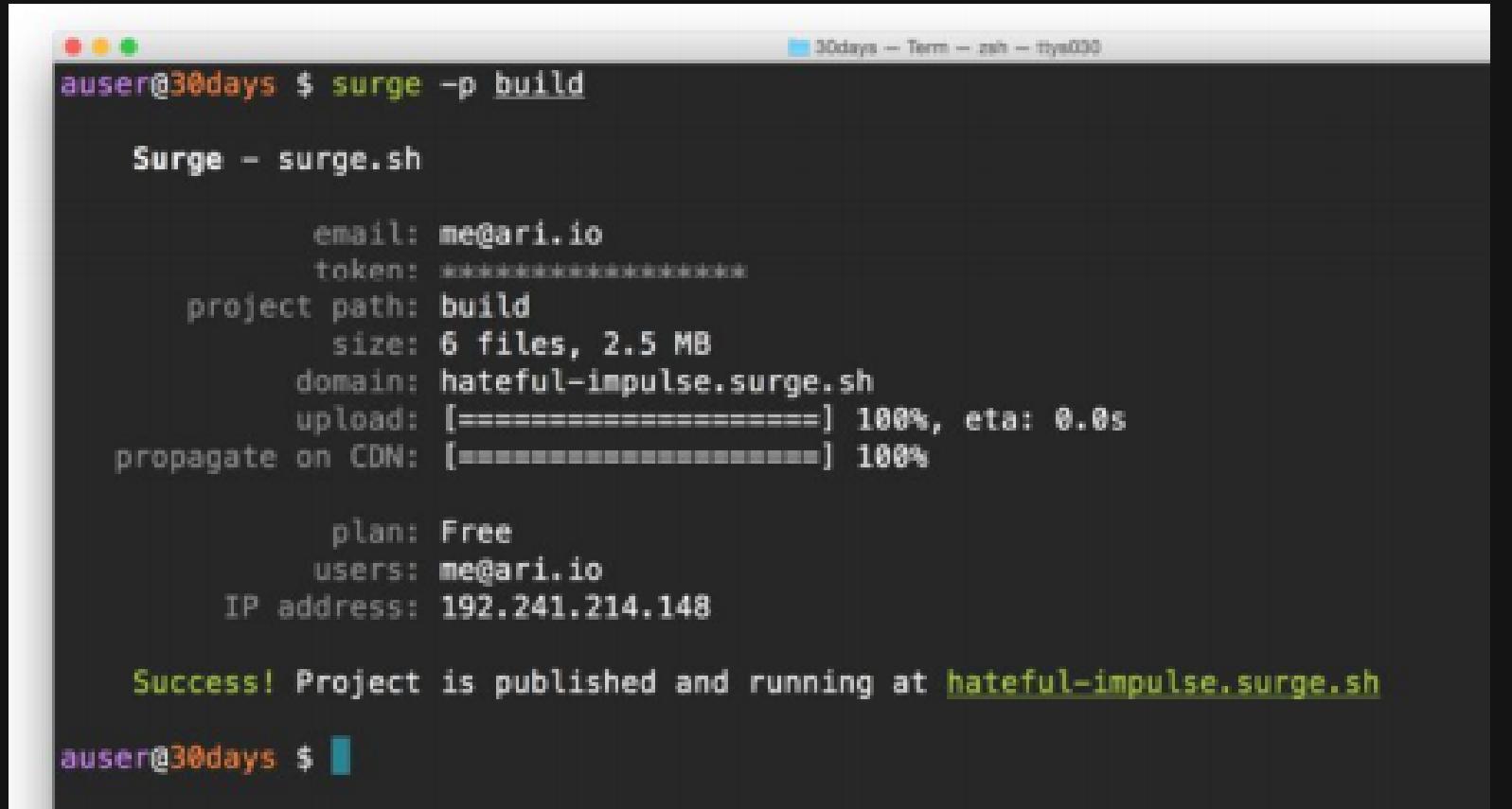


DEPLOYMENT

Po zalogowaniu przechodzimy w terminalu do naszego katalogu build i wpisujemy komendę

```
surge -p build
```

potrzebną do wrzucenia plików już na stronę



```
auser@30days $ surge -p build

Surge - surge.sh

email: me@ari.io
token: *****
project path: build
size: 6 files, 2.5 MB
domain: hateful-impulse.surge.sh
upload: [=====] 100%, eta: 0.0s
propagate on CDN: [=====] 100%

plan: Free
users: me@ari.io
IP address: 192.241.214.148

Success! Project is published and running at hateful-impulse.surge.sh
auser@30days $
```

Powinniśmy dostać wygenerowany link do strony z naszą aplikacją (lub możliwością wpisania własnej nazwy).

# GitHub pages

Github pages to kolejna łatwa usługa do wdrażania naszych plików statycznych.

Musimy tylko posiadać konto github do hostowania naszych plików git, ale jest kolejnym łatwym w użyciu środowiskiem hostingowym dla aplikacji jednostronnicowych.



Jeśli nie mamy jeszcze repozytorium z naszą aplikacją, którą chcemy wrzucić na hosting - inicjujemy repo.

```
git init  
git add -A .  
git commit -am "Initial commit"
```

W głównym katalogu dodajemy remote'a:

```
git remote add github [your git url here]
```

Przechodzimy na branch gh-pages (github sam tworzy).  
Możemy sprawdzić czy gałąź istnieje oraz pokazujemy gitowi, że głównym katalogiem na tym branchu jest katalog build

```
npm run build  
git checkout -B gh-pages  
git add -f build  
git commit -am "Rebuild website"  
git filter-branch -f --prune-empty --subdirectory-filter build  
git checkout -
```

## DEPLOYMENT

Github pages nie obsługuje bezpośrednio z głównego katalogu, więc musimy dodać konfigurację do `package.json`

Dodajemy klucz homepage z url naszego gita

```
{  
  "name": "30days",  
  "version": "0.0.1",  
  "private": true,  
  "homepage": "http://auser.github.io/30-days-of-react-demo  
(http://auser.github.io/30-days-of-react-demo)",  
  // ...  
}
```

W książce proponują zrobić to za pomocą programu jq (lejki program do edytowania JSON'a z wiersza poleceń).  
Jeśli mamy zainstalowany, wpisujemy

```
jq '.homepage = \  
  "http://auser.github.io/30-days-of-react-demo  
(http://auser.github.io/30-days-of-react-demo)"' \  
 > package.json
```

DEPLOYMENT

Po wniesieniu zmiany do pliku musimy zrobić nowego build'a i push'nąć zmiany na githuba z katalogu build

```
git push -f github gh-pages
```

Strona powinna już być dostępna na github pages.

# Heroku

Usługa hostingowa, która pozwala nam hostować zarówno statyczne, jak i niestatyczne strony internetowe.

Instalujemy

```
npm install -g heroku
```

Przechodzimy w terminalu do folderu z projektem i logujemy się do Heroku z terminala

```
heroku login
```

Powinno też przekierować na stronę internetową do zalogowania się/stworzenia konta.

Tworzymy nową aplikację na Heroku poprzez  
heroku create apps-name

```
heroku apps:create
```

Musimy pokazać Heroku, że mamy pakiet statyczny (static-file buildpack) do plików statycznych, aby mógł obsługiwać naszą aplikację jako plik statyczny. Instalujemy plugin

```
heroku plugins:install heroku-cli-static
```

Możemy teraz dodać pakiet statyczny

```
heroku buildpacks:set https://github.com/hone/heroku-buildpack-static  
(https://github.com/hone/heroku-buildpack-static)
```

Po jakiekolwiek zmianie konfiguracji - musimy wygenerować wymagany plik static.json

```
heroku static:init
```

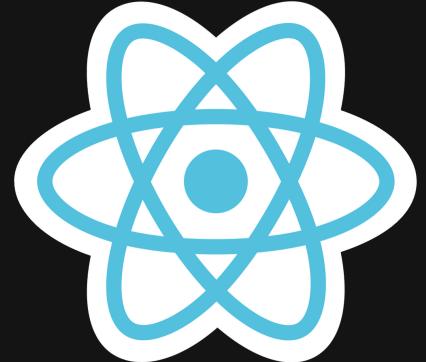
Dopiero możemy wrzucić aplikację na heroku używając gita

```
git push heroku master  
# or from a branch, such as the heroku branch  
git push heroku heroku:master
```

# Continuous Integration

## Ciągła integracja

29 DZIEŃ REACT'A



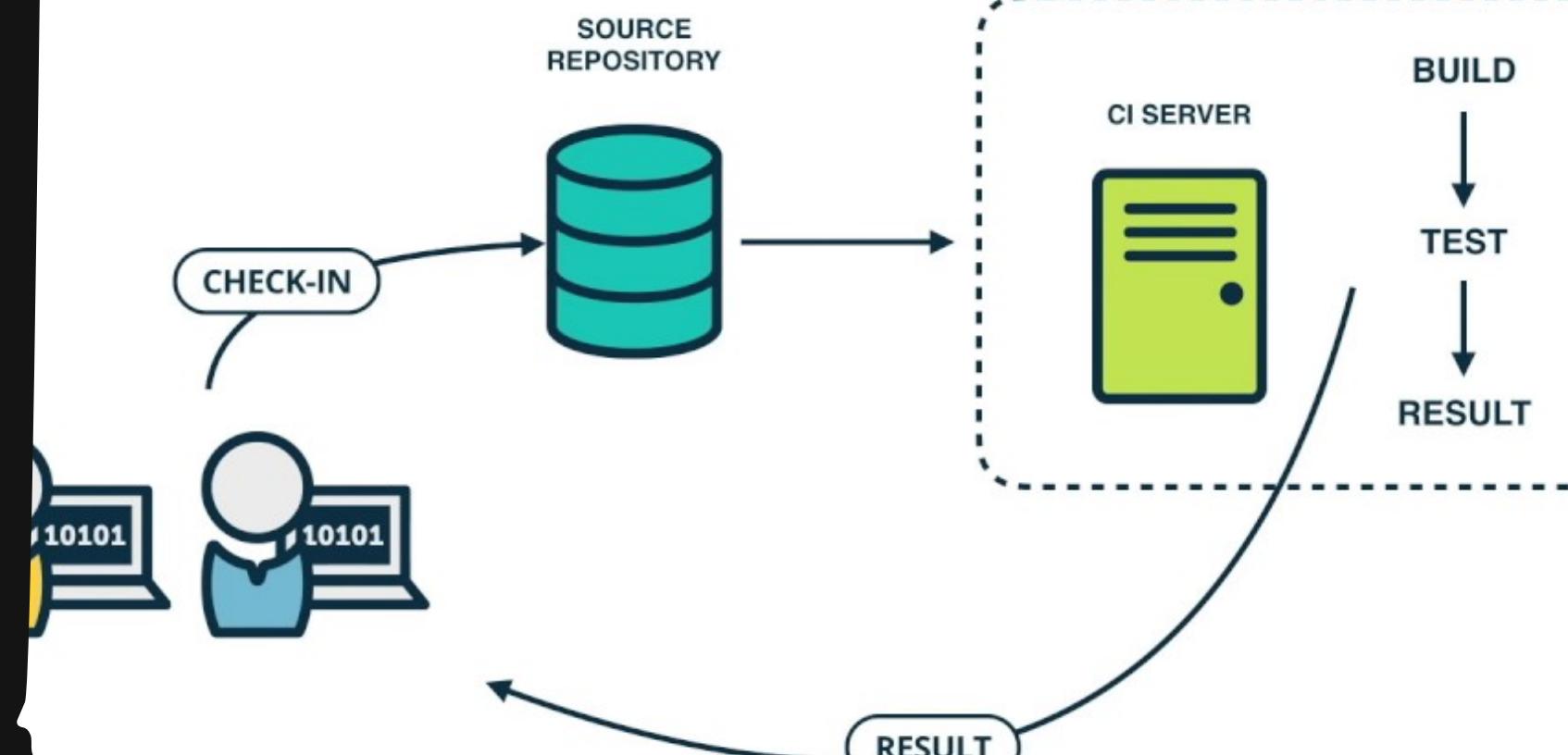
# Continuous Integration

## CIĄGŁA INTEGRACJA

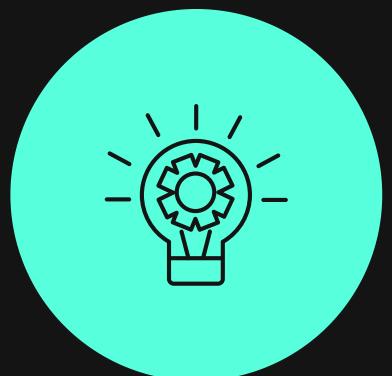
Praktyka stosowana w trakcie rozwoju oprogramowania, polegająca na częstym, regularnym włączaniu (integracji) bieżących zmian w kodzie do głównego repozytorium i każdorazowej weryfikacji zmian, poprzez zbudowanie projektu (jeśli jest taka potrzeba) oraz wykonanie testów jednostkowych.

CONTINUOUS  
INTEGRATION

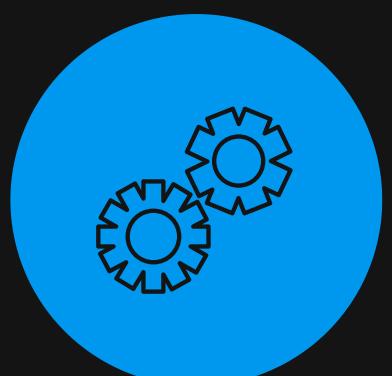
## Continuous Integration (CI)



# Testing then deployment



Stworzymy test



Dodamy aplikację na  
continuous integration server,  
który uruchomi nasz test



Dopiero po pozytywnym  
przejściu testu



Wrzuci aplikację na  
chmurę

Po wrzuceniu aplikacji na zdalny  
serwer, chcemy być pewni, że będzie  
działać poprawnie - dlatego

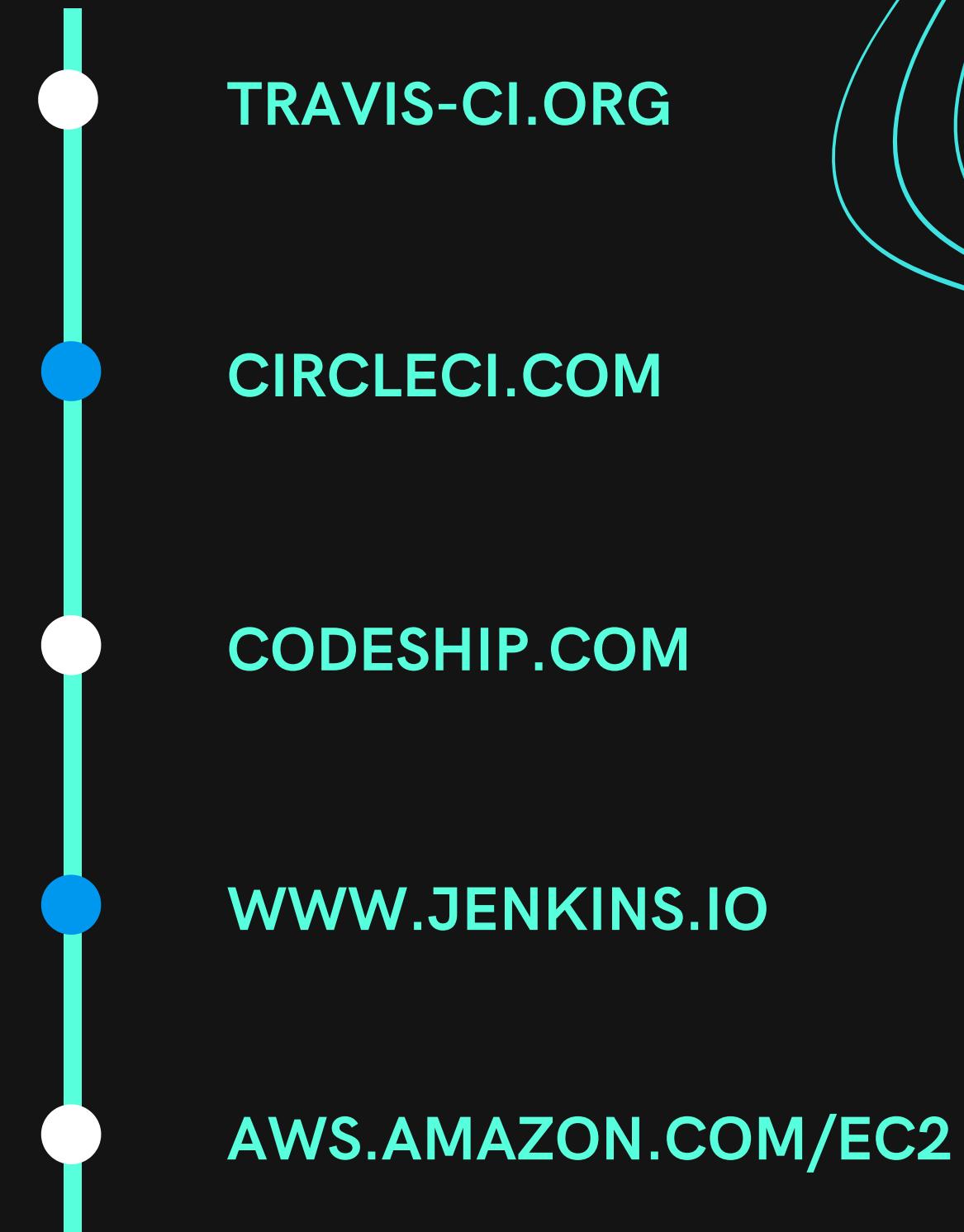


CONTINUOUS  
INTEGRATION



Jedne z popularnych  
stron, zapewniających  
taką usługę

Skupimy się na travis



# KONCEPT

Bez angażowania dodatkowych serwisów, które zapewniają z reguły płatne strony, możemy napisać skrypt, aby który wykona nasze testy przed wdrożeniem.

Utwórzmy skrypt, który najpierw wykonuje proces wdrażania.

Skorzystajmy z poprzedniego przykładu - surge.sh

Dodajemy skrypt o nazwie `deploy.sh` w katalogu `scripts/`

```
touch scripts/deploy.sh  
chmod u+x scripts/deploy.sh
```

Dodajemy skrypt wdrażania surge

```
#!/usr/bin/env bash  
surge -p build --domain hateful-impulse.surge.sh
```

CONTINUOUS  
INTEGRATION

Dodajemy skrypt release - aby się wykonał dodajemy go do package.json

```
{  
  // ...  
  "scripts": {  
    "start": "node ./scripts/start.js",  
    "build": "node ./scripts/build.js",  
    "release": "node ./scripts/release.js",  
    "test": "node ./scripts/test.js"  
  },  
}
```

Tworzymy plik scripts/release.js - z poziomu głównego katalogu

```
touch scripts/release.js
```

CONTINUOUS  
INTEGRATION

20

Wewnątrz pliku będziemy uruchamiać kilka skryptów wiersza polecenia:



1 Build



2 Odpalenia testów



3 Finalnie wdrożenia aplikacji,  
jeśli testy będą pozytywne

W pliku node ustawiamy, aby NODE\_ENV był testowany ->

```
process.env.NODE_ENV = "test";  
--
```

```
process.env.NODE_ENV = "test";  
process.env.CI = true;  
  
var chalk = require("chalk");  
const exec = require("child_process").exec;  
  
var output = [];  
function runCmd(cmd) {  
  return new Promise((resolve, reject) => {  
    const testProcess = exec(cmd, { stdio: [0, 1, 2] });  
  
    testProcess.stdout.on("data", msg => output.push(msg));  
    testProcess.stderr.on("data", msg => output.push(msg));  
    testProcess.on("close", code => (code === 0 ? resolve() :  
      reject()));  
  });  
}
```

Dodamy również skrypt do uruchomienia komendy z wiersza poleceń z node script i zapisania wszystkich danych wyjściowych w tablicy.

Po wywołaniu - funkcja `runCmd()` zwróci promise'a, który zostanie rozwiązany, gdy zakończy się pomyślnie lub odrzuci w przypadku błędu.

Nasz skrypt będzie teraz wykonywał następujące zadania:

- 1 build
- 2 testowanie
- 3 deploy
- 4 zraportowanie ewentualnych błędów

Moglibyśmy wyobrazić sobie tego pipeline'a, jako:

```
build()  
  .then(runTests)  
  .then(deploy)  
  .catch(error);
```

CONTINUOUS  
INTEGRATION

Napiszmy te funkcje, które użyją `runCmd()`

```
function build() {  
  console.log(chalk.cyan("Building app"));  
  return runCmd("npm run build");  
}  
  
function runTests() {  
  console.log(chalk.cyan("Running tests..."));  
  return runCmd("npm test");  
}  
  
function deploy() {  
  console.log(chalk.green("Deploying..."));  
  return runCmd(`sh -c "${__dirname}/deploy.sh" `);  
}  
  
function error() {  
  console.log(chalk.red("There was an error"));  
  output.forEach(msg => process.stdout.write(msg));  
}  
  
build()  
  .then(runTests)  
  .then(deploy)  
  .catch(error);
```

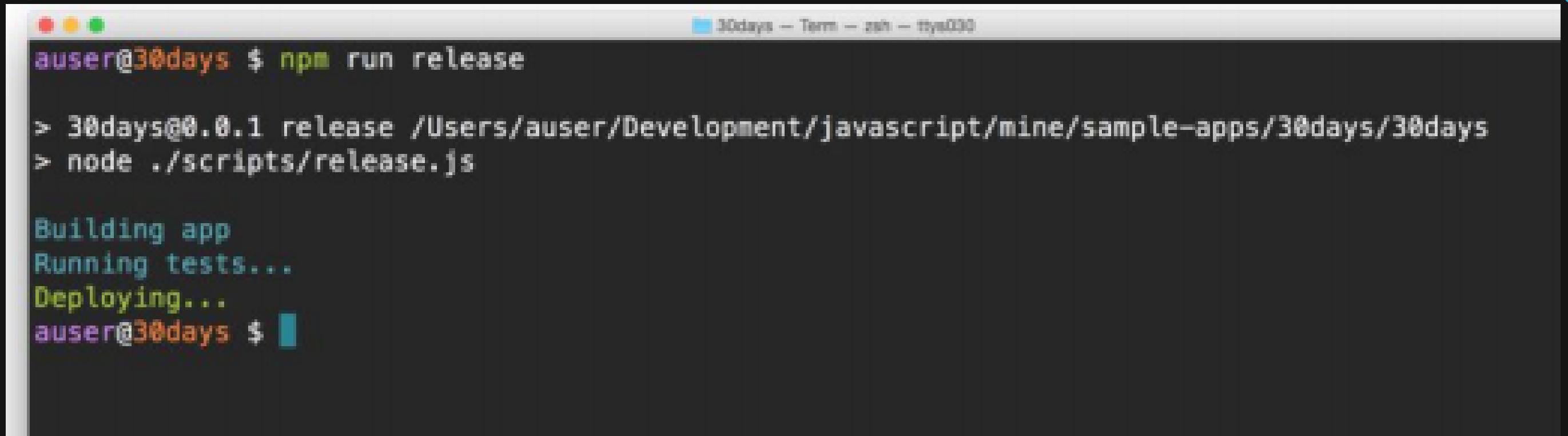


CONTINUOUS  
INTEGRATION

Po skonfigurowaniu odpowiednio pliku scripts/release.js wywołujemy komendę

```
npm run release
```

Teraz aplikacja powinna przejść testy pozytywnie i zostać wrzucona na dany serwer



```
auser@30days $ npm run release

> 30days@0.0.1 release /Users/auser/Development/javascript/mine/sample-apps/30days/30days
> node ./scripts/release.js

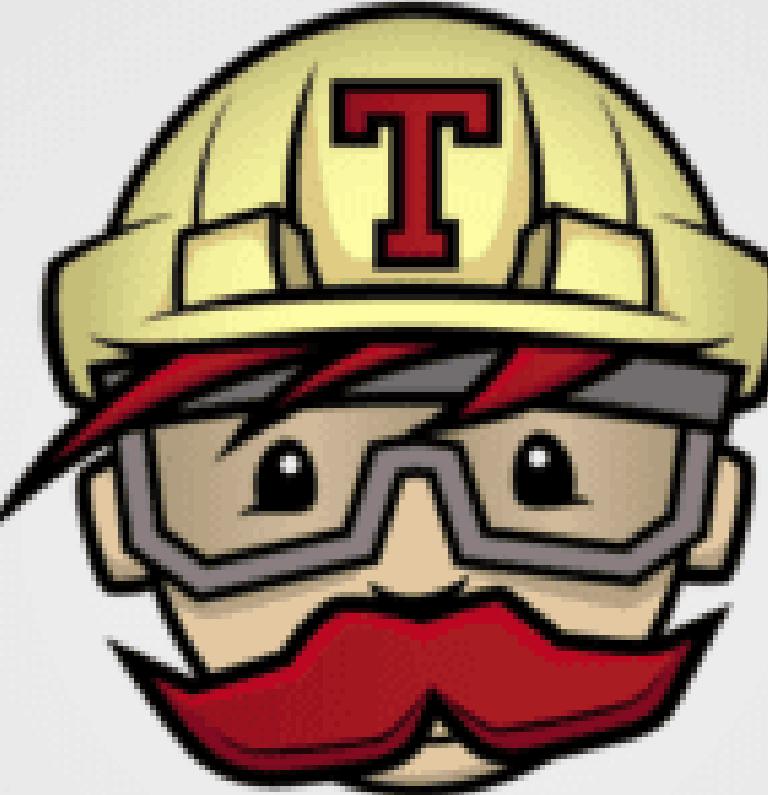
Building app
Running tests...
Deploying...
auser@30days $
```

CONTINUOUS  
INTEGRATION

# Travis CI

## HOSTED CONTINUOUS INTEGRATION ENVIRONMENT

Easily sync your projects with Travis CI and you'll be testing your code in minutes!



TRAVIS

CONTINUOUS  
INTEGRATION

Ponieważ zrobiliśmy push do github, skorzystajmy z tego i skonfigurujmy travis za pomocą konta github.

Przechodzimy do strony i aktywujemy nasze repozytorium gtihub

The screenshot shows the Travis CI dashboard. At the top, there is a navigation bar with links: Dashboard, Changelog, Documentation, and Help. Below the navigation bar is a search bar labeled "Search all repositories" with a magnifying glass icon. Underneath the search bar, there is a section titled "My Repositories" with a plus sign (+) to its right. A horizontal teal line runs across the screen below this section. In the center, there is a button labeled "Add New Repository". Below the button, the text "No repositories found" is displayed. The background of the dashboard is white, and the overall interface is clean and modern.

CONTINUOUS  
INTEGRATION



Aby umożliwić Travis CI automatyczne logowanie się podczas wdrażania, musimy dodać zmienne środowiskowe

**SURGE\_LOGIN** i **SURGE\_TOKEN**

Otwórz menu "Więcej opcji", dalej Ustawienia.

W obszarze zmiennych środowiskowych utwórz zmienną o nazwie **SURGE\_LOGIN** i ustaw ją na adres e-mail używany w Surge.

Następnie dodaj kolejną zmienną o nazwie **SURGE\_TOKEN** i ustaw ją na swój token Surge.

Możesz wyświetlić swój token surge, wpisując 'surge token' w swoim terminalu. Ponieważ używamy surge do wdrażania, powinniśmy go również załadować do naszych devDependencies w package.json.

Uruchom **npm install surge --save-dev**, aby go dodać

CONTINUOUS  
INTEGRATION



Musimy skonfigurować travis, aby uruchamiał nasze skrypty testowe, a następnie wdrażał naszą aplikację.

Aby skonfigurować travis, musimy utworzyć plik `.travis.yml` w katalogu głównym naszej aplikacji.

```
touch .travis.yml
```

CONTINUOUS  
INTEGRATION

Dodajmy następującą treść, aby ustawić język w node v. 10.15.0

```
language: node_js  
node_js:  
  - "10.15.0"
```

Teraz musimy dodać plik **.travis.yml** do git i przesłać zmiany repo do github

```
git add .travis.yml  
git commit -am "Added travis-ci configuration file"  
git push github master
```

Teraz travis wykona nasze testy w oparciu o domyślny skrypt testu npm.

CONTINUOUS  
INTEGRATION

Chcemy, aby travis wdrożył dla nas naszą aplikację. Ponieważ mamy już skrypt `scripts/deploy.sh`, który wdroży naszą aplikację, możemy użyć tego do wdrożenia z travis - dodajemy `deploy`key do `.travis.yml`, aby travis uruchomił nasz skrypt deploy.sh.

Musimy również zbudować naszą aplikację przed wdrożeniem, stąd `before_deploy`

Zaktualizujmy konfigurację yml, (aby nakazała uruchomienie naszego skryptu wdrażania):

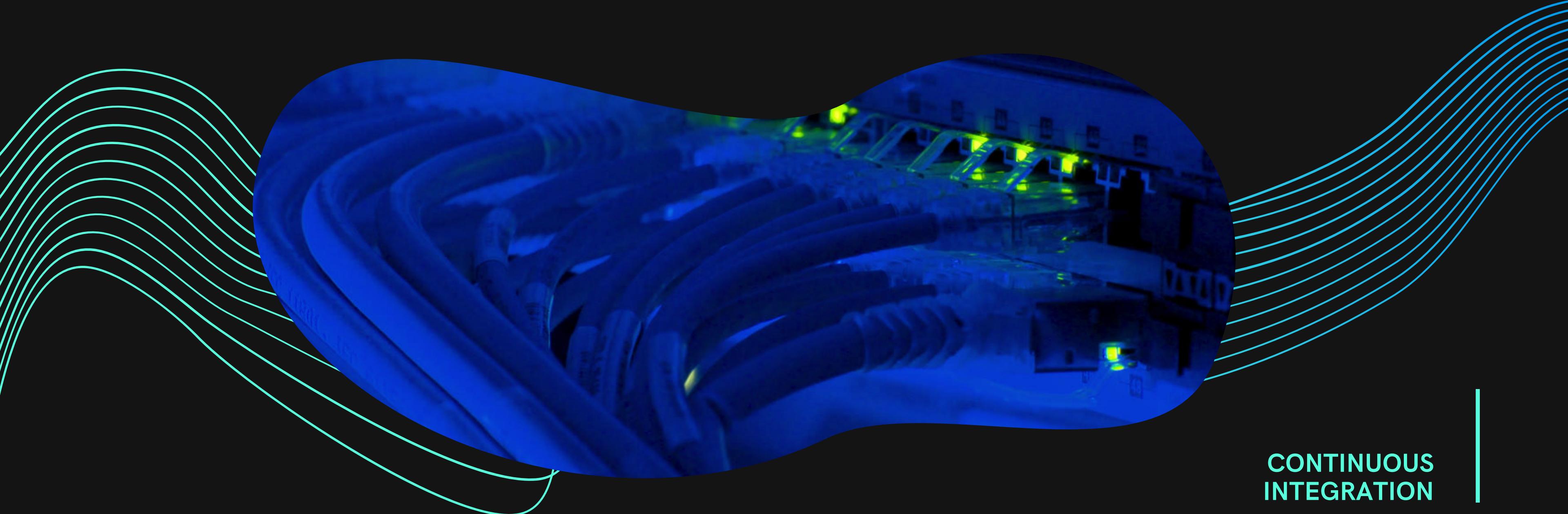
```
language: node_js
node_js:
  - "10.15.0"
before_deploy:
  - npm run build
deploy:
  provider: script
  skip_cleanup: true
  script: sh scripts/deploy.sh
on:
  branch: master
```

CONTINUOUS  
INTEGRATION



# DONE

Następnym razem, gdy będziemy push'ować - travis przejmie kontrolę i zrobi push do surge (lub gdziekolwiek `scripts/deploy.sh` każą mu się wdrożyć).



CONTINUOUS  
INTEGRATION

## DANE DO UWIERZYTELNIENIA

Aby zrobić deployment na stronach github, musimy dodać token do skryptu - pomocny link:  
<https://gist.github.com/domenic/ec8b0fc8ab45f39403dd>

# Linki z poradnikami krok po kroku wrzucania aplikacji react na hosting

CONTINUOUS  
INTEGRATION

32

[HTTPS://LEVELUP.GITCONNECTED.COM/HOW-TO-DEPLOY-  
REACT-APP-TO-PRODUCTION-B79645CB12E9](https://levelup.gitconnected.com/how-to-deploy-react-app-to-production-b79645cb12e9)

deploying React App to Heroku, now.sh, and surge.sh

**DEPLOY REACT APPS ON NETLIFY IN LESS THAN 30 SECONDS**

Deploy React Apps on Netlify in less than 30 Seconds

[HTTPS://WWW.NETLIFY.COM/BLOG/2016/07/22/DEPLOY-  
REACT-APPS-IN-LESS-THAN-30-SECONDS/](https://www.netlify.com/blog/2016/07/22/deploy-react-apps-in-less-than-30-seconds/)

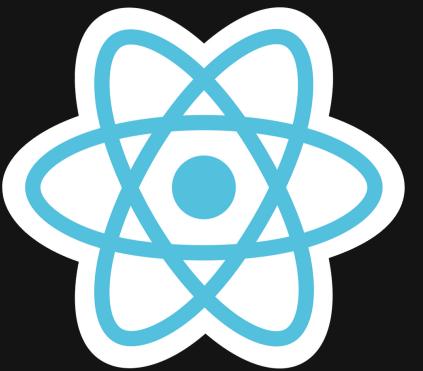
Deploy your react app in 3 minutes with surge

[HTTPS://MEDIUM.COM/@VINICIUSGULARTE/DEPLOY-YOUR-  
REACT-APP-IN-3-MINUTES-WITH-SURGE-11BEBE96B871](https://medium.com/@viniciusgularte/deploy-your-react-app-in-3-minutes-with-surge-11bebe96b871)

[HTTPS://BULLDOGJOB.PL/NEWS/747-GITHUB-PAGES-I-  
REACT-JAK-STWORZYC-W-TEN-SPOSOB-STRONE](https://bulldogjob.pl/news/747-github-pages-i-react-jak-stworzyc-w-ten-sposob-strone)

# Summary

30 DZIEŃ REACT'A



# The high-level topics we discussed in our first 30 days:

1. JSX and what it is, from the ground up.
2. Building components
  - a. Static
  - b. Data-driven components
  - c. Stateful and stateless components
  - d. Pure components
  - e. The inherent tree-based structure of the virtual DOM
3. The React component lifecycle
4. How to build reusable and self-documenting components
5. How to make our components stylish using native React prototypes as well as third party libraries
6. Adding interaction to our components
7. How to use create-react-app to bootstrap our apps
8. How to integrate data from an API server, including a look at promises
  9. We worked through the Flux architecture
10. Integrated Redux in our application, including how middleware works
11. We integrated testing strategies in our app
  - a. Unit testing
  - b. End-to-end testing
  - c. Functional testing
12. We discussed deployment and extending our application to support multi-environment deployments
13. We added continuous integration in our deployment chain.
14. Client-side routing

SUMMARY

34

