

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МО ЭВМ**

**ОТЧЕТ**  
**по научно-исследовательской работе**  
**Тема: Непрерывная интеграция, непрерывная доставка и**  
**развертывание приложения, написанного на Node.js**

Студентка гр. 5304

\_\_\_\_\_

Орлова В.В.

Преподаватель

\_\_\_\_\_

Заславский М.М.

Санкт-Петербург

2020

**ЗАДАНИЕ**  
**НА НАУЧНО-ИССЛЕДОВАТЕЛЬСКУЮ РАБОТУ**

Студентка Орлова В.В.

Группа 5304

Тема работы: Непрерывная интеграция, непрерывная доставка и развертывание приложения, написанного на Node.js

Содержание пояснительной записки:

«Содержание», «Определения, обозначения и сокращения», «Введение»,  
«Постановка задачи», «Выводы», «Список литературы»

Предполагаемый объем пояснительной записки:

Не менее 20 страниц.

Сроки выполнения НИР: 01.09.2020 – 20.12.2020

Дата сдачи отчета: 20.12.2020

Дата защиты отчета: 20.12.2020

Студентка

\_\_\_\_\_

Орлова В.В.

Преподаватель

\_\_\_\_\_

Заславский М.М.

## **АННОТАЦИЯ**

В данной работе представлена идея и реализация настройки CI/CD реального приложения, с использованием пайплайна с помощью GithubActions и Docker. Пайплайн в итоге прогоняет тесты, запускает проверку кода и выгружает image приложения в DockerHub, после чего выгружает приложение на сервер с помощью AWS Elastic Beanstalk.

## **SUMMARY**

This work presents the idea and implementation of CI / CD settings for a real application, using a pipeline of GithubActions and Docker. As a result, the pipeline runs the tests, runs code review and uploads the application image to DockerHub, and then uploads the application to the server using AWS Elastic Beanstalk.

## СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ .....	5
ВВЕДЕНИЕ .....	7
1. ПОСТАНОВКА ЗАДАЧИ .....	9
2. РЕЗУЛЬТАТЫ РАБОТЫ В ВЕСЕННЕМ СЕМЕСТРЕ .....	15
3. ПЛАН РАБОТЫ НА ОСЕННИЙ СЕМЕСТР .....	27
ВЫВОДЫ.....	28
СПИСОК ЛИТЕРАТУРЫ .....	29

## ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

**Непрерывная интеграция (Continuous Integration – CI)** — это практика разработки программного обеспечения, которая заключается в слиянии рабочих копий в общую основную ветвь разработки несколько раз в день и выполнении частых автоматизированных сборок проекта для скорейшего выявления потенциальных дефектов и решения интеграционных проблем [5];

**Непрерывная доставка (Continuous delivery – CD)** – является продолжением непрерывной интеграции, дающая возможность быстро и эффективно выпускать новые изменения для своих клиентов. Это означает, что помимо автоматизации тестирования также возможность автоматизировать процесс выпуска и возможность в любой момент развернуть приложение, нажав кнопку [2];

**Непрерывное развертывание (Continuous deployment – CD)** – идет на один шаг дальше, чем непрерывная доставка. С этой практикой каждое изменение, которое проходит все стадии производственного процесса, передается клиентам. Вмешательство человека не требуется, и только неудачный тест не позволит внедрить новое изменение в производство [1];

**Node.js** – это платформа для разработки приложений на языке JavaScript, позволяющая быстро создавать приложения, работающие в сети [7];

**Jenkins** — программная система с открытым исходным кодом на Java, предназначенная для обеспечения процесса непрерывной интеграции программного обеспечения [1];

**Docker** — программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации. Позволяет «упаковать» приложение со всем его окружением и зависимостями в контейнер, который может быть перенесён на Linux-систему, а также предоставляет среду по управлению контейнерами [2];

**GitHub** — крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки [2];

**Github Actions** — функциональность, позволяющая автоматизировать рабочий процесс;

**DockerHub** — открытый репозиторий образов контейнеров Docker;

**Amazon Web Services (AWS)** — коммерческое публичное облако, поддерживаемое и развиваемое компанией Amazon с 2006 года. Предоставляет подписчикам услуги как по инфраструктурной модели (виртуальные серверы, ресурсы хранения), так и платформенного уровня (облачные базы данных, облачное связующее программное обеспечение, облачные бессерверные вычисления, средства разработки);

**Elastic Beanstalk** — сервис оркестрации, предлагаемый Amazon Web Services для развертывания приложений, которые управляют различными сервисами AWS, включая S3, CloudWatch и др.

## ВВЕДЕНИЕ

Непрерывная интеграция и непрерывное развертывание — это две современные практики разработки программного обеспечения. Непрерывная интеграция (CI) — это процесс автоматизации сборки и тестирования кода каждый раз, когда член команды вносит изменения в управление версиями. Непрерывное развертывание (CD) можно рассматривать как расширение непрерывной интеграции, и оно представляет собой процесс автоматического развертывания приложения после успешного завершения CI. Цель состоит в том, чтобы минимизировать время выполнения заказа - время между написанием одной новой строки кода во время разработки и началом использования этого нового кода живыми пользователями в производстве.

Есть много преимуществ для практики CI / CD. В этой работе не будет подробно рассказываться о каждом преимуществе, но необходимо выделить некоторые из них:

- Преимущества непрерывной интеграции:
  - Меньше ошибок
  - Меньше переключения контекста, так как разработчики получают предупреждение, как только они ломают сборку
  - Затраты на тестирование снижены
  - Команда QA тратит меньше времени на тестирование
- Преимущества непрерывного развертывания:
  - Релизы менее рискованны
  - Легкий выпуск
  - Клиенты видят непрерывный поток улучшений
  - Ускорение разработки, так как нет необходимости приостанавливать разработку для выпусков

Именно поэтому данная работа ставит перед собой задачи исследования процессов непрерывной интеграции и доставки, а также настройки CI/CD для

автоматизации процессов тестирования, построения и развертывания приложения.



## 1. ПОСТАНОВКА ЗАДАЧИ

Целью работы является исследование современных CI/CD инструментов, позволяющих работать с NodeJs-приложениями и внедрение в конкретное приложение на примере Github Actions.

Проблема, которой обусловлена эта работа – это то, что ежедневная жизнь разработчика заполнена однообразными и повторяющимися задачами, которые отлично автоматизируются, такие как прогон тестов, проверка стиля кода и выгрузка готового image в DockerHub и развертка с помощью AWS Elastic Beanstalk. Именно этим обусловлена актуальность настройки CI/CD приложения.

Поставлены следующие задачи:

- обзор аналогичных методов и продуктов;
- реализация в web-проекте;
- тестирование в реальных условиях;
- разработка итогового решения.

## 2. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ

### 2.1. Обзор аналогов в предметной области

Принцип отбора аналогов: возможность организации непрерывной интеграции, непрерывной доставки и развертывания приложения, написанного на Node.js.

- Jenkins

Jenkins — это проект с открытым исходным кодом, написанный на Java, который работает на Windows, macOS и других Unix-подобных операционных системах.

В Jenkins доступна обширная библиотека плагинов. Плагины Jenkins охватывают пять областей: платформы, пользовательский интерфейс, администрирование, управление исходным кодом и, наиболее часто, управление сборкой. Хотя другие инструменты CI предоставляют аналогичные функции, им не хватает полной интеграции плагинов, которую имеет Jenkins. Более того, сообщество Jenkins поощряет своих пользователей расширять функциональность новыми функциями, предоставляя учебные ресурсы.

- Travis CI

Как одно из старейших решений CI, Travis завоевал доверие многих пользователей. Он имеет большое и полезное сообщество, которое приветствует новых пользователей и предоставляет множество учебных пособий.

Travis CI может тестировать на Linux и MacOS. Предлагая множество автоматических опций CI, Travis устраняет необходимость в выделенном сервере, так как он размещен в облаке.

**Ценовая политика.** Первые 100 сборок бесплатны. В противном случае существует четыре тарифных плана для хобби-проектов (69 долларов в месяц) и для небольших, растущих и более крупных команд (от 129 до 489 долларов в месяц).

- TeamCity

TeamCity от JetBrains - это надежный и качественный CI-сервер. Команды часто выбирают TeamCity для большого количества функций аутентификации, развертывания и тестирования, а также поддержки Docker. TeamCity работает сразу после установки, никаких дополнительных настроек или настроек не требуется. Он обладает рядом уникальных функций, таких как подробные хронологические отчеты, мгновенные отзывы о неудачных тестах и повторное использование настроек, поэтому вам не нужно дублировать код.

**Ценовая политика** TeamCity предлагает бесплатную версию с полным доступом ко всем функциям продукта, но она ограничена 100 конфигурациями сборки и тремя агентами сборки. Добавление еще одного агента сборки и 10 конфигураций сборки в настоящее время стоит \$ 299.

- GitHub Actions

GitHub Actions — это CI/CD система, интегрированная с GitHub. В первом приближении можно думать о ней, как об аналоге TeamCity или Jenkins, предоставляемом в виде сервиса. Сервис бесплатен для открытых проектов, и даже для закрытых, если ваши билды собираются не слишком долго и/или не слишком часто.

Первый и главный плюс GitHub Actions - это простота настройки. Действия GitHub работают в облаке, но у вас также есть возможность запустить его локально.

По этой причине сервер Jenkins требует установки, тогда как GitHub Actions не нуждается в этом. Следовательно, процесс настройки в GitHub Actions очень удобен. Кроме того, GitHub Actions - это серия запусков докеров. Для этого требуется только сборка докера и запуск докера. Это упрощает запуск и отладку.

- Gitlab CI

Gitlab предоставляет уникальную возможность получить одновременно бесплатный приватный репозиторий и бесплатный CI/CD из коробки в том же месте. Для сохранения баланса он конфигурируется не менее уникальным способом через конфиг-файл.

## **2.2. Критерии сравнения аналогов**

- **Облачный хостинг**

Размещение возможно на физическом сервере или в облаке, аналог должен иметь в возможностях второй вариант.

Обоснование критерия: облачные инструменты размещаются на стороне поставщика, требуют минимальных настроек и могут быть скорректированы по требованию в зависимости от ваших потребностей.

Размещенные решения избавляют от трудностей установки, предлагая большую масштабируемость. Этот вариант более подходит для тестируемого приложения т.к. не требует поддержки физического сервера.

- **Актуальные интерфейс и документация**

Обоснование критерия: некоторые инструменты могут сделать процесс сборки намного проще, чем другие, благодаря их четкому и понятному графическому интерфейсу и UX. Тщательно продуманный интерфейс и подробная документация могут сэкономить ваше время на этапе адаптации.

- **Экономичность**

Обоснование критерия: поскольку проект не коммерческий и поддерживается силами исключительно автора работы, цена является одним из главных критериев по понятным причинам. Поскольку упор работы сделан на изучение именно непрерывной интеграции, в дальнейшем предполагается многократный запуск пайплайнов, поэтому оптимальным будет бесплатное решение.

Сравнение аналогов по критериям представлено в таблице 2.1

Таблица 2.1. – Сравнения аналогов по критериям

	Облачный хостинг	Актуальные интерфейс и документация	Экономичность
Jenkins	Физический сервер, -	Устаревшие, -	Бесплатный +
TeamCity	Физический сервер или облако +	+	Дорогой – (от 299\$)
Travis CI	+	+	Дорогой – (от 69\$/month)
GitHub Actions	+	+	Бесплатный +
Gitlab CI	+	+	Дешевый +- (от 15\$/месяц)

### 2.3. Выводы по итогам сравнения

Облачный хостинг, как и актуальные интерфейс и документация, являются крайне важными критериями, поэтому решение с использованием Jenkins, несмотря на огромную популярность, не подходит для решения поставленной задачи. Технология осваивается автором практически с нуля, поэтому документация и tutorиалы должны содержать актуальную информацию, а интерфейс – быть интуитивно понятным.

Физический сервер был бы приемлем, если бы у студентов была возможность постоянно его поддерживать, но, поскольку аналоги предлагают облачные решения, проще обратиться к ним и избежать, помимо прочего, сложностей настройки сервера.

Экономичность – другой важнейший критерий, т.к. проект не коммерческий и поддерживается силами исключительно автора работы. Большинство решений предлагают учебные версии, но, поскольку планируется продолжительная разработка, пробные периоды, а также ограничение по количеству задач/времени крайне нежелательны. Поэтому дорогие решения TeamCity и Travis CI рассматриваться дальше не будут. Gitlab CI предлагает относительно дешевое использование, но проигрывает в сравнении с бесплатными Jenkins и GitHub Actions.

## **2.4. Выбор метода решения**

Готовым решением будет служить сформированный после анализа исходного пула возможных решений технология непрерывной интеграции. Выбранная технология должна соответствовать следующим требованиям:

- Облачный хостинг – для размещения на стороне поставщика, минимальных затрат сил для начальных и последующих настроек и поддержки.
- Актуальные интерфейс и документация – для быстрого освоения новой технологии, а также последующего удобства работы с системой.
- Экономичность – для возможности поддержания учебного приложения без трудностей и ограничений, связанных с работой с пробными периодами, а также для избегания лишних трат со стороны студента.

В результате сравнения аналогов были показаны преимущества использования Github Actions. Помимо вышесказанного, дополнительные удобства предоставляется тем, что репозиторий учебного проекта уже расположен на Github и не требует никаких дополнительных сил для начала работы с данной технологией.

### 3. РЕЗУЛЬТАТЫ РАБОТЫ В ОСЕННЕМ СЕМЕСТРЕ

#### 3.1. Подтверждение результата

Ссылка на github-репозиторий с исходным кодом:

<https://github.com/NikaOrl/vkr-orlova>

Ссылка на настроенный в ходе НИР CI: <https://github.com/NikaOrl/vkr-orlova/actions>

Ссылка на DockerHub репозиторий:

<https://hub.docker.com/repository/docker/nikaorl/vkr-orlova>

#### 3.2. Технологии для реализации решения

Предполагается использовать следующий стек технологий:

Для приложения, которому будет создаваться непрерывная интеграция (рисунок 3.2.1.):

- MySQL – реляционная СУБД (с использованием Knex – ORM абстракцией низкого уровня)
- Express – каркас веб-приложений, работающий поверх Node.js;
- Angular – фреймворк JavaScript, интерфейсной части веб-приложения, работающей в браузере;
- Node.js – JavaScript платформа для серверной разработки.

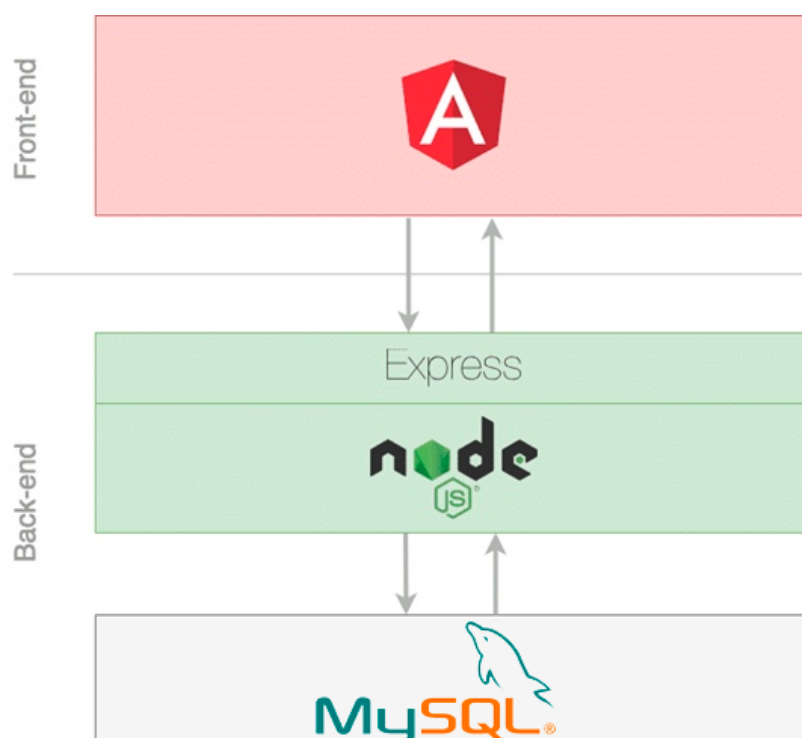


Рисунок 3.2.1– Иллюстрация взаимодействия технологий в выбранном стеке

Для непрерывной интеграции и непрерывного развертывания:

- Github – крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки
- Github Actions - функциональность, позволяющая автоматизировать рабочий процесс.
- DockerHub - открытый репозиторий образов контейнеров Docker.
- AWS - Elastic Beanstalk – сервис оркестрации, предлагаемый Amazon Web Services для развертывания приложений, которые управляют различными сервисами AWS, включая S3, CloudWatch и др. Elastic Beanstalk идеально подходит для интернет-приложений, созданных с использованием Docker.

### 3.3. Описание разработанных результатов

В ходе НИР были достигнуты следующие результаты:

- Приложение было покрыто unit-тестами;



- Были использованы GitHub Actions для непрерывной интеграции приложения;
- Были использованы GitHub Actions для непрерывной доставки с отправкой образа Docker в реестр Docker (Docker Hub);
- Были использованы GitHub Actions для непрерывного развертывания с Elastic Beanstalk.

Результат показан на рисунке 3.3.1.

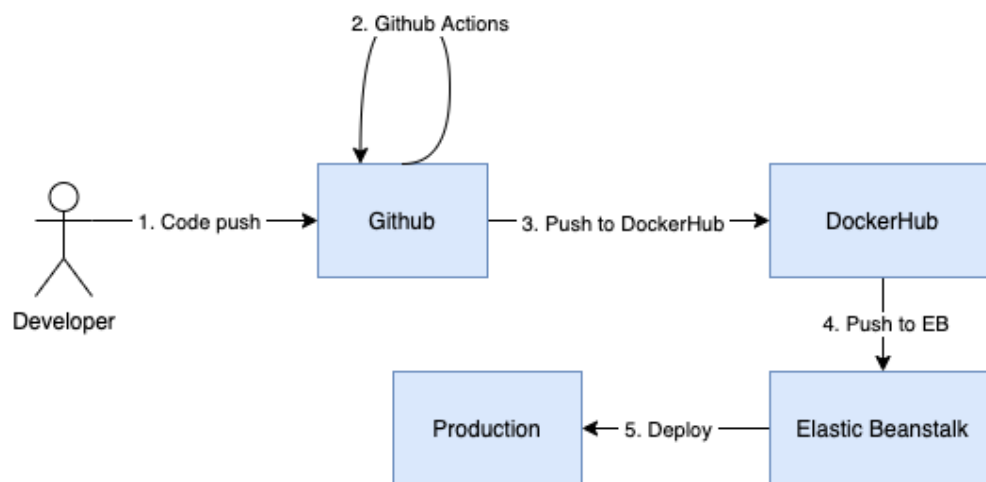


Рисунок 3.3.1. Ожидаемый результат

GitHub Actions поможет автоматизировать процесс CI / CD. При каждом изменении кода из репозитория приложения GitHub Actions получает уведомления, и они загружает изменения на сервер Github, устанавливает зависимости, запускает юнит-тесты и линтит код [1].

Если все тесты пройдут, GitHub Actions публикует image приложения в DockerHub. Если это не удастся, разработчик будет уведомлен [2]. Вся схема пошагово изображена на рисунке 3.3.2.

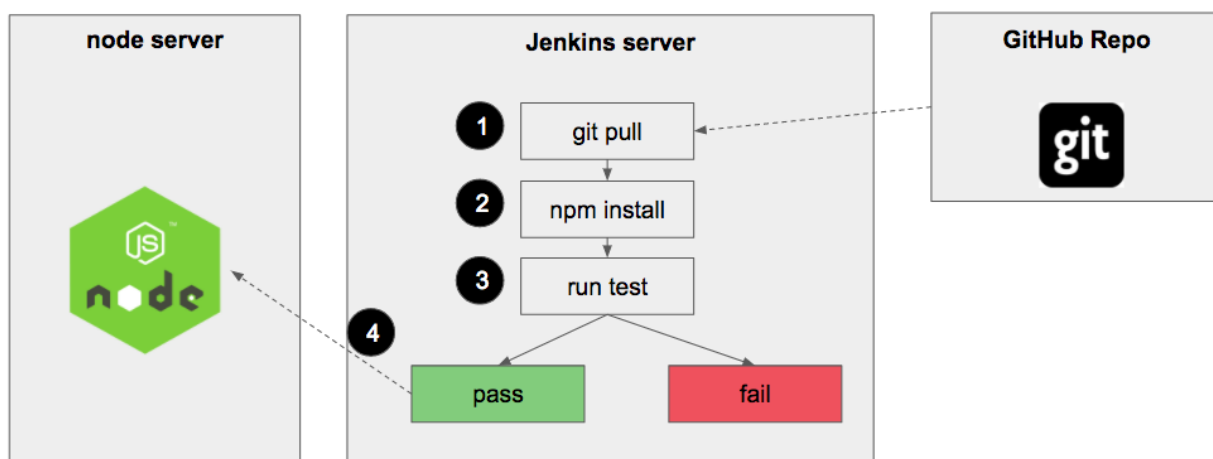


Рисунок 3.3.2. Схема непрерывной интеграции с Github Actions

Использование DockerHub позволяет выбирать любую службу развертывания, поддерживающую контейнеры Docker. В данной работе был использован Elastic Beanstalk в качестве платформы развертывания, потому что он предлагает простой способ настройки интеграции DockerHub и еще более простой способ продолжить развертывание новых версий.

Результат этой части выглядит как показано на рисунке 3.3.3.

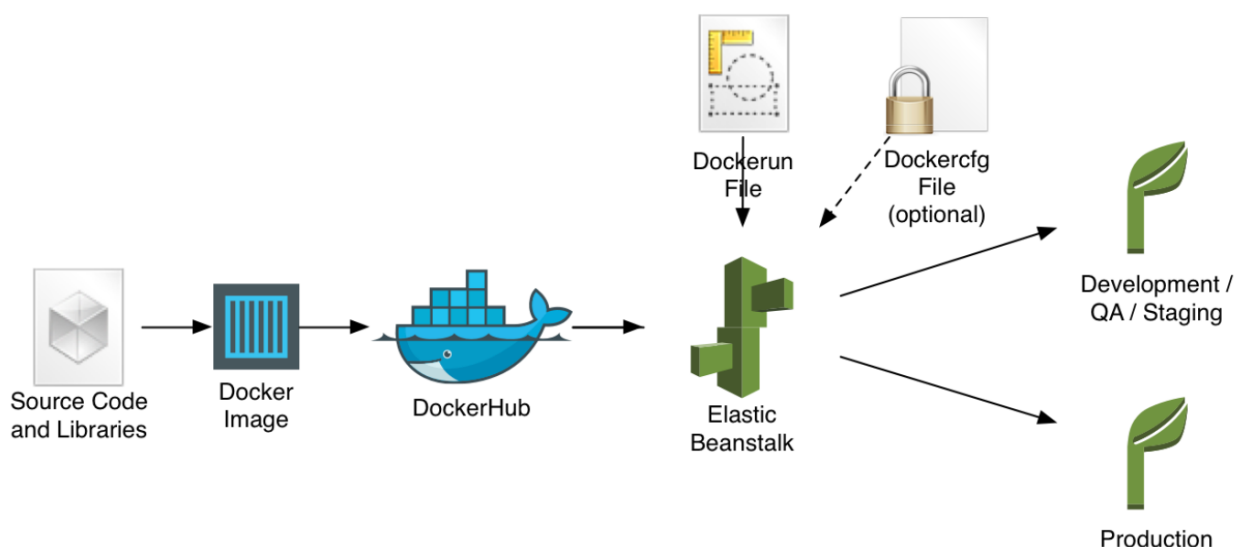


Рисунок 3.3.3. Схема непрерывного развертывания с Github Actions

Порядок выполнения следующий:

1. Получение Docker image из исходного кода.
2. Отправка Docker image в DockerHub

3. Отправка Docker image в Elastic Beanstalk
4. Развертывание версии приложения в производственной среде

Все это организовано с помощью Github Actions, благодаря чему получено полноценное CI/CD, в котором разработчику для развертывания приложения нужно лишь запустить код в репозиторий.

### 3.4. Примеры работы

В результате покрытия всех сущностей созданного кода тестами получилось 52 unit-теста. Все они проходят успешно. Результат их запуска в Karma с помощью команды Angular CLI `ng test` представлен на рисунке 3.4.1.

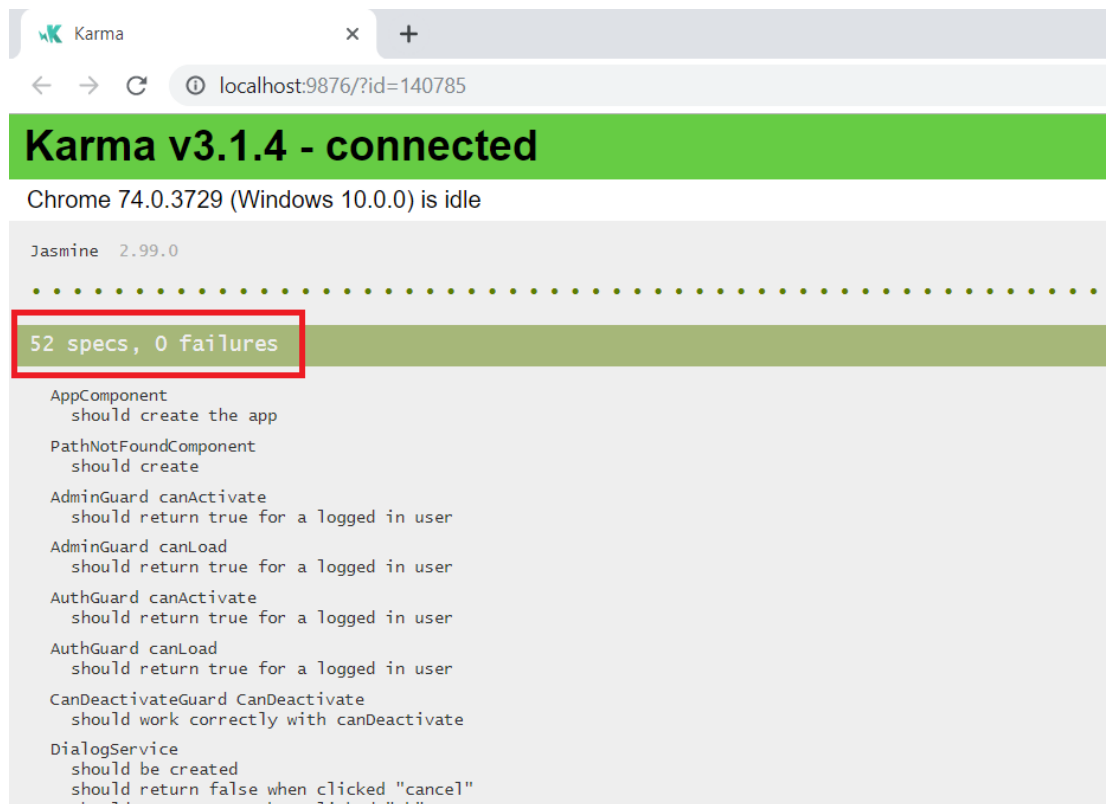


Рисунок 3.4.1 – Результат запуска unit-тестов

Далее на рисунке 3.4.2 следуют отчет о покрытии, генерируемый Angular при запуске в CLI команды `ng test --code-coverage`:

**All files**

90.14% Statements
384/426

80% Branches
28/25

84.31% Functions
129/153

89.69% Lines
348/388

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File ▾		Statements ▾		Branches ▾	
src	<div></div>	100%	3/3	100%	0/0
src/app	<div></div>	100%	3/3	100%	0/0
src/app/core	<div></div>	100%	3/3	100%	0/0
src/app/core/components/path-not-found	<div></div>	100%	4/4	100%	0/0
src/app/core/guards	<div></div>	100%	18/18	100%	0/0
src/app/core/services	<div></div>	100%	6/6	100%	2/2
src/app/groups/components/group-page	<div></div>	100%	4/4	100%	0/0
src/app/groups/components/group-table	<div></div>	90.63%	29/32	50%	1/2
src/app/groups/components/groups-edit	<div></div>	93.67%	74/79	100%	3/3
src/app/groups/services	<div></div>	88.46%	23/26	50%	1/2
src/app/login/components/login-page	<div></div>	92.86%	13/14	50%	1/2
src/app/login/services/authentication	<div></div>	90%	9/10	100%	2/2
src/app/marks/components/marks-edit	<div></div>	88.81%	119/134	100%	8/8
src/app/marks/components/marks-page	<div></div>	100%	3/3	100%	0/0
src/app/marks/components/marks-table	<div></div>	85.42%	41/48	50%	1/2
src/app/marks/services	<div></div>	82.05%	32/39	50%	1/2

Рисунок 3.4.2. – Отчет о покрытии unit-тестами

Для автоматизированного прогона тестов с CI был настроен пайплайн с помощью GithubActions [3]. При каждом push в репозиторий выполняются следующие действия в указанном порядке:

- 1. Склонировать ветку из репозитория
- 2. Запустить установку npm install
- 3. Запустить проверку кода с помощью npm lint
- 4. Запустить тесты с помощью npm test
- 5. Сбилдить docker image
- 6. Залогиниться в docker hub
- 7. Запустить image в docker hub
- 8. Развернуть image в Elastic Beanstalk

Вот так выглядит push.yml, который github использует для этого [4]:

```

on: [push]
name: npm build, lint, test and publish
jobs:
  build-and-publish:
    name: build and publish
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Setup node
        uses: actions/setup-node@v1
        with:
          node-version: "12"
      - run: npm install
      - run: npm run test-once
      - run: npm run lint
      - name: Login to docker hub
        uses: actions-hub/docker/login@master
        env:
          DOCKER_USERNAME: ${ secrets.DOCKER_USERNAME }
          DOCKER_PASSWORD: ${ secrets.DOCKER_PASSWORD }

      - name: Build :latest
        run: docker build -t nikaorl/vkr-orlova:latest .

      - name: Push to docker hub :latest
        uses: actions-hub/docker@master
        with:
          args: push nikaorl/vkr-orlova:latest

```

В шаге, когда мы хотим войти в Docker Hub, здесь используются Github-secrets, которые передаются в переменные окружения нашей сборки. Можно установить эти переменные окружения разными способами [5]. В данном случае это было настроено через GitHub, что показано на рисунке 3.4.3.

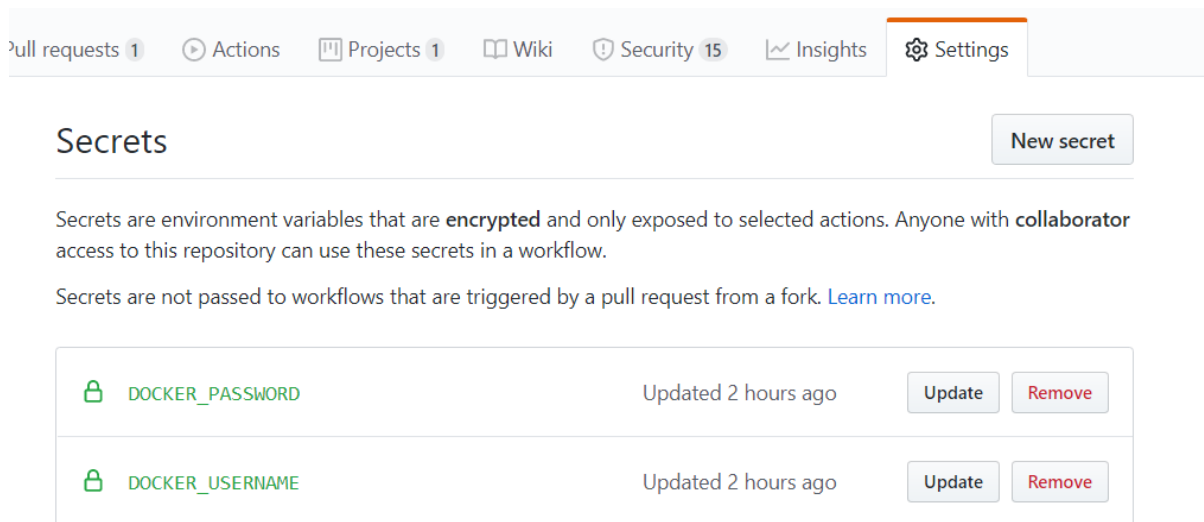


Рисунок 3.4.3 – Secrets в Github

В результате благодаря этим настройкам GithubActions вступят в игру и начнут выполнять все этапы [6]. Результат будет выглядеть, как показано на рисунке 3.4.4.

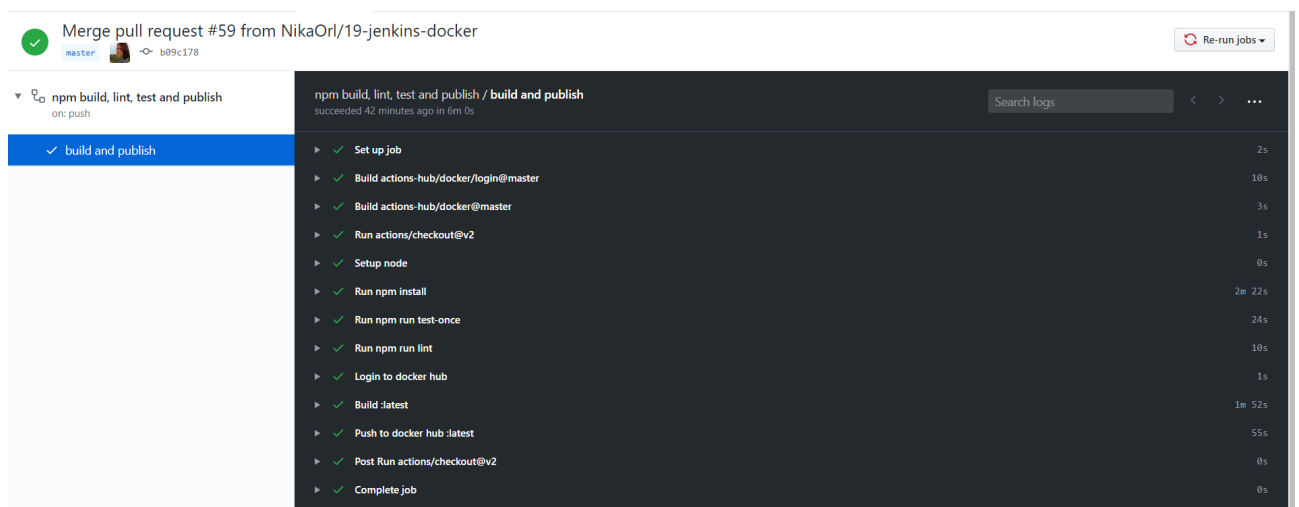


Рисунок 3.4.4 – Пайплайн в Github

А в реквесте будет висеть уведомление, о том, что все проверки пройдены и image был опубликован в DockerHub, что показано на рисунке 3.4.5.

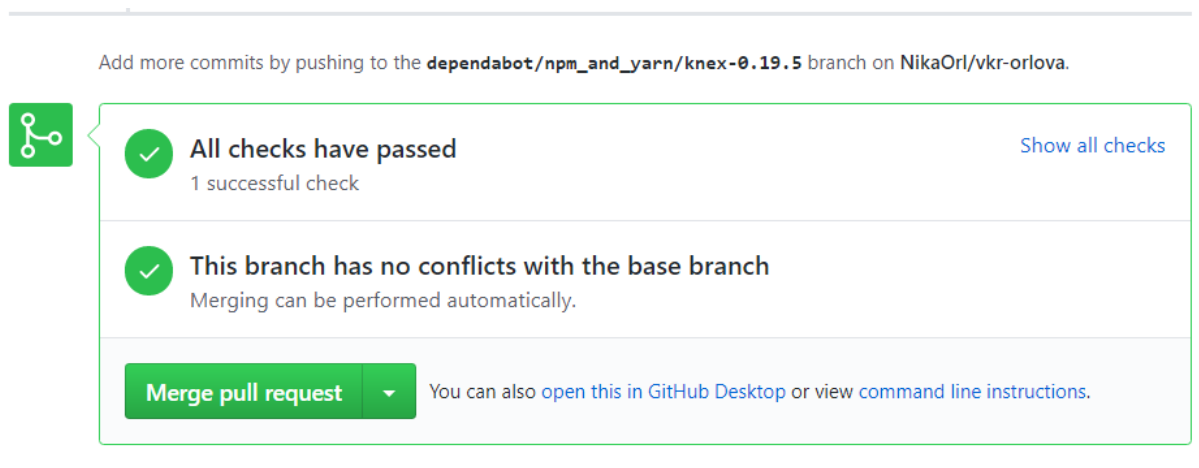


Рисунок 3.4.5 – Реквест после проверки пайплайном в Github

Сам докер имеет следующие настройки, которые расположены в Dockerfile, текст которого расположен ниже [7]:

```
# Base Node Image
FROM node:14-alpine AS base
WORKDIR /vkr-orlova
RUN npm i -g node-prune

# Couchbase sdk requirements
RUN apk update && apk add bash

# Install dependencies
FROM base AS dependencies
COPY package*.json ./
RUN npm ci && npm cache clean --force

# Copy Files and Build
FROM dependencies AS build
WORKDIR /vkr-orlova
COPY . .
RUN /vkr-orlova/node_modules/.bin/ng build

# Copy it and remove dev dependencies
FROM build AS prodDependencies
WORKDIR /vkr-orlova
COPY --from=dependencies /vkr-orlova/package*.json ./
COPY --from=dependencies /vkr-orlova/node_modules ./node_modules/
RUN npm prune --production && node-prune

FROM node:14-alpine
WORKDIR /vkr-orlova
COPY --from=build /vkr-orlova/dist/vkr-orlova ./dist
COPY --from=prodDependencies /vkr-orlova/node_modules ./node_modules
EXPOSE 3000
CMD ["node", "./dist/main.js"]
```

Для создания image, сборка Docker была разделена на два этапа [8]:

- компиляция исходного кода в готовый к выпуску вывод,
- запуск скомпилированного приложения в образе Docker.

Новое image было помещено в DockerHub, что показано на рисунке 3.4.6 и оно полностью готово к выгрузке на сервер.

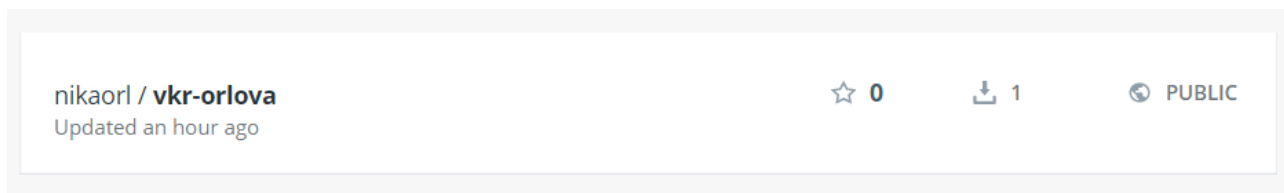


Рисунок 3.4.6 – Image в Docker Hub

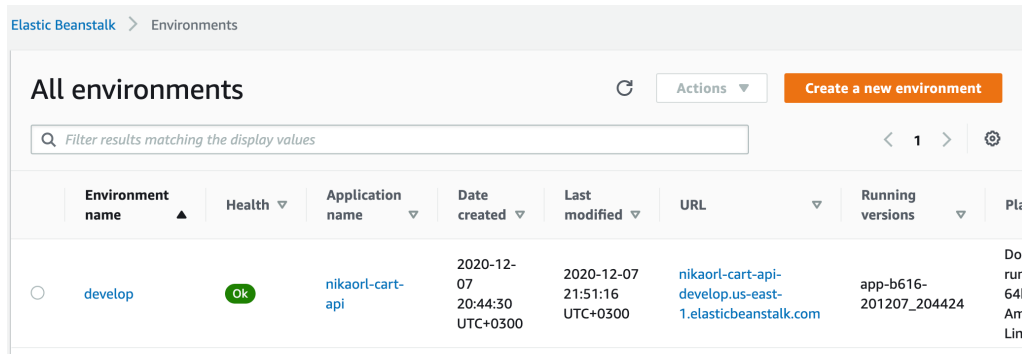
Далее был инициализирован Elastic Beanstalk с помощью EB CLI и создан необходимый environment. В консоль AWS были добавлены все необходимые настройки и переменные окружения. Полученный в результате конфиг показан на рисунке 3.4.7.

```
elasticbeanstalk > ! config.yml
1  branch-defaults:
2    62-Deploy-to-EB:
3      environment: develop3
4      group_suffix: null
5  global:
6    application_name: vkr-orlova
7    branch: null
8    default_ec2_keyname: null
9    default_platform: Docker running on 64bit Amazon Linux 2
10   default_region: us-east-1
11   include_git_submodules: true
12   instance_profile: null
13   platform_name: null
14   platform_version: null
15   profile: null
16   repository: null
17   sc: git
18   workspace_type: Application
19
```

Рисунок 3.4.7. Конфигурация EB.



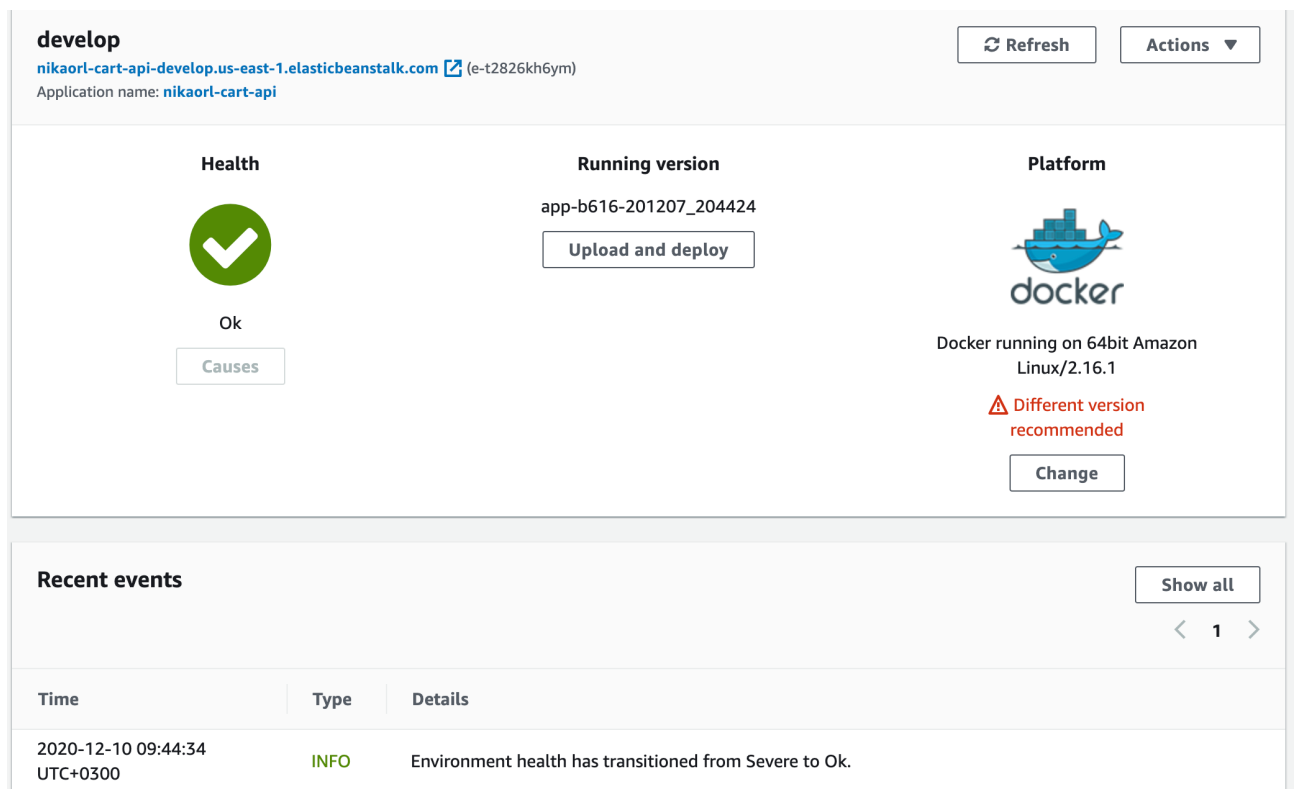
В Elastic Beanstalk было добавлено новое окружение и консоль AWS не видит с ним никаких проблем, что показано на рисунке 3.4.8.




Environment name	Health	Application name	Date created	Last modified	URL	Running versions
develop	Ok	nikaorl-cart-api	2020-12-07 20:44:30 UTC+0300	2020-12-07 21:51:16 UTC+0300	nikaorl-cart-api-develop.us-east-1.elasticbeanstalk.com	app-b616-201207_204424

Рисунок 3.4.8. Новый environment в Elastic Beanstalk.


Если открыть environment подробнее, можно увидеть, что оно действительно использует написанный DockerFile, что показано на рисунке 3.4.9.



**develop**  
nikaorl-cart-api-develop.us-east-1.elasticbeanstalk.com (e-t2826kh6ym)  
Application name: nikaorl-cart-api

**Health**  
  
Ok  
Causes

**Running version**  
app-b616-201207\_204424  
Upload and deploy

**Platform**  
  
Docker running on 64bit Amazon Linux/2.16.1  
Different version recommended  
Change

**Recent events** Show all

Time	Type	Details
2020-12-10 09:44:34 UTC+0300	INFO	Environment health has transitioned from Severe to OK.

Рисунок 3.4.9. Подробный отчет о статусе environment.

Там же можно найти url развернутого приложения. На данный момент без всех необходимых настроек для фронтенд-части приложения и без загрузки базы данных, можно протестировать лишь то, что приложение действительно развернуто с помощью специально добавленного тестового эндпойнта, работающего следующим образом:

```
router.get("/test", function(req, res, next) {  
  res.send("test");  
});
```

То есть при обращении к url {host}/api/test он возвращает test. Если теперь открыть URL найденный в консоли AWS, перейти на него и перейти там на данный URL, то увидим именно это сообщение. Это показано на рисунке 3.4.10.

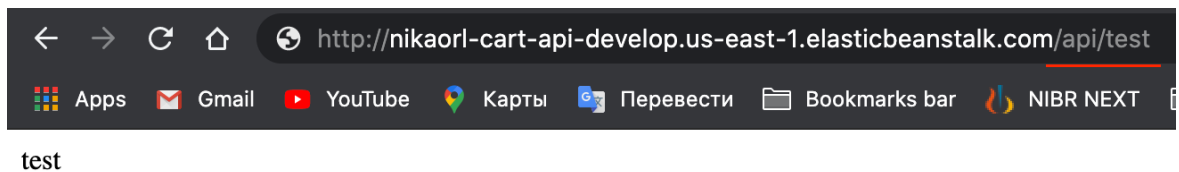


Рисунок 3.4.10. Тестирование развернутого на ЕВ приложения.

Как видно, в результате приложение было развернуто на новом URL, предоставленном AWS Elastic Beanstalk.

#### **4. ПЛАН РАБОТЫ НА ВЕСЕННИЙ СЕМЕСТР**

В весеннем семестре планируется:

- 4.1. Продолжить работу с AWS чтобы настроить консоль и приложение для:
  - разворачивания на AWS базы данных для приложения;
  - настройки фронтенд-части;
  - реализации авторизации с помощью AWS.
- 4.2. Продолжить написание новой функциональности приложения, согласовывая требования с руководителем.

## ВЫВОДЫ

Тестирование и развертывание – два неотъемлемых элемента веб-разработки. При некоторой смешанной автоматизации они становятся решениями, которые обычно называют «непрерывной интеграцией» (CI) и «непрерывным развертыванием» (CD). «Непрерывный» аспект этих решений означает, что ваши проекты будут автоматически протестированы и развернуты, что позволит вам больше сосредоточиться на написании кода и о том, чтобы пасти его на серверы.

Внедряя Continuous Integration следует на самых ранних этапах разработки, можно сразу после создания структуры проекта. Это позволит существенно сократить время на развертывание сборок для тестового и в дальнейшем проще перенести для рабочего окружения.

Для проекта на Node.js написание тестов как юнит, так и интеграционных - это “must have” практика, так как нет компиляции проекта и сложно выявлять опечатки.

В этой научно-исследовательской работе были рассмотрены рекомендации, касающиеся систем непрерывной интеграции, оценки эффективности тестов, анализа качества кода, а также проведена реализация этого на реальном приложении.

## СПИСОК ЛИТЕРАТУРЫ

1. How to set up CI/CD Pipeline for a node.js app with Jenkins // Medium. URL: <https://medium.com/@mosheezderman/how-to-set-up-ci-cd-pipeline-for-a-node-js-app-with-jenkins-c51581cc783c> (дата обращения: 14.05.2020).
2. How to CI and CD a Node.JS Application Using GitHub Actions // Medium. URL: <https://blog.bitsrc.io/https-medium-com-adhasmana-how-to-do-ci-and-cd-of-node-js-application-using-github-actions-860007bebae6> (дата обращения: 14.05.2020).
3. Now lets add some steps to our pipeline! Find problems in your code with ESLint // DEV. URL: <https://dev.to/paulasantamaria/improving-your-ci-pipeline-for-node-js-2aab> (дата обращения: 14.05.2020).
4. Тестирование Node.js-проектов. Часть 2. Оценка эффективности тестов, непрерывная интеграция и анализ качества кода // Хабр. URL: <https://habr.com/ru/company/ruvds/blog/435464/> (дата обращения: 14.05.2020).
5. Continuous Integration для новичков // Хабр. URL: <https://habr.com/ru/post/352282/> (дата обращения: 14.05.2020).
6. Настройка непрерывной интеграции и непрерывного развертывания с Дженкинсом // envatotuts+. URL: <https://code.tutsplus.com/ru/tutorials/setting-up-continuous-integration-continuous-deployment-with-jenkins--cms-21511> (дата обращения: 14.05.2020).
7. Testing and Documenting Node.js APIs with Mocha and Acquit // StrongLoop. URL: <https://strongloop.com/strongblog/nodejs-testing-documenting-apis-mocha-acquit/> (дата обращения: 14.05.2020).
8. Continuous Integration and deployment (CI/CD Pipeline) with Jenkins and Node.js // CodeForGeek. URL: <https://codeforgeek.com/continuous-integration-deployment-jenkins-node-js/> (дата обращения: 14.05.2020).