Содержание

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

ЭВМ — Электронно-вычислительная машина

ВС — Вычислительная система

ПО — Программное обеспечение

DevOps — Development Operations, «Разработка и Эксплуатация»

SaaS — Software as a Service, «Программное обеспечение как услуга»

IaC — Infrastructure as Code, «Инфраструктура как код»

CI/CD — Continuous Integration/Continuous Deployment, «Продолжительная интеграция/Продолжительная развёртка»

VCS — Version Control System, «Система контроля версий»

CLI — Command Language Interface, «Интерфейс командной строки»

ВВЕДЕНИЕ

В современном мире информационных технологий крайне распространена DevOps методология разработки ПО[1]. Тем не менее развёртка рабочего окружения занимает значительное время, поскольку включает множество задач от описания файлов конфигураций до разработки дополнительных программных средств развёртывания.

В практических целях автору был предложен веб-сервис для развёртывания. Результаты выполнения задания должны быть представлены в выпускной квалификационной работе.

На основании выданного технического задания, определяется цель исследования: развернуть и автоматизировать управление жизненным циклом веб-сервиса на базе Node.js.

Для реализации цели исследования ставятся следующие задачи исследования:

— провести обзор необходимых средств для автоматизации управления жизненным циклом веб-сервиса,

— рассмотреть полученный для развёртывания веб-сервис и проанализировать требования,

— провести проектирование механизмов автоматизации управления жизненного цикла веб-сервиса,

— составить план тестирования механизмов развёртывания веб-сервиса,

— провести практические работы по развёртке и автоматизации управления жизненного цикла веб-сервиса,

— провести тестирование механизмов развёртывания и обосновать полученные результаты.

Поставленные задачи определяют предмет исследования:

Основные параметры имеющихся DevOps решений с целью определения наиболее подходящих из них для развёртывания веб-сервисов по наиболее популярным и современным методологиям IaC, CI/CD и SaaS.

Объект исследования: программные и аппаратные средства, задействованные и которые могут быть использованы для автоматизации управления жизненным циклом веб-сервиса.

При проведении исследований были просмотрены 20 источников, в том числе 11 электронных ресурсов, что отражено в библиографическом списке. При написании выпускной квалификационной работы (ВКР) из библиографического списка использовано 5 источников.

ВКР состоит из введения, основной части, включающей четыре главы, заключения и приложений. В ВКР содержится 44 страница основного текста, 16 рисунков, 4 таблицы, 5 листингов и 0 приложений[2].

1 ОБЗОР КОМПОНЕНТОВ АВТОМАТИЗАЦИИ УПРАВЛЕНИЯ ЖИЗНЕННЫМ ЦИКЛОМ ВЕБ-СЕРВИСА

Разрабатываемая в рамках данной работы система автоматизации управления жизненным циклом веб-сервиса, согласно техническому заданию, должна отвечать следующим требованиям:

— использовать современные DevOps методологии,

— взаимодействие с конечным пользователем по методологии SaaS,

— возможность добавления в систему не описанных заранее сервисов по методологии IaC,

— использование методологии CI/CD для взаимодействия с инфраструктурой и организации контроля качества,

— наличие конфигурации прав доступа (приватизации исходного кода),

— наличие хранилища пакетов и образов,

— стоимость установки и обслуживания системы — не более 2 500 рублей за установку 2 500 рублей в месяц за обслуживание на момент написания данной работы.

Для корректного формулирования требований к разрабатываемому комплексу автоматизации управления жизненным циклом веб-сервиса, необходимо проанализировать и сравнить характеристики схожих по назначению решений, применяемых в данный момент на архитектуре «Клиент-Сервер», либо по своим характеристикам подходящих для такого применения.

Рассматривая рынок развёртывания программного обеспечения, стоит начать с такого ключевого элемента, как VCS.

1.1 Система контроля версий

Наиболее популярной системой контроля версий на момент написания работы является git[3]. Git — система управления версиями программного обеспечения с распределенной архитектурой. В отличие от

некогда популярных систем вроде CVS и Subversion (SVN), где полная история версий проекта доступна лишь в одном месте, в Git каждая рабочая копия кода сама по себе является репозиторием. Это позволяет всем разработчикам хранить историю изменений в полном объеме. Разработка в Git ориентирована на обеспечение высокой производительности, безопасности и гибкости распределенной системы.

1.2 Git хостинг

Сам по себе git предоставляет только инструменты для локальной разработки и имеет весьма ограниченный функционал. Для работы в большинстве случаев выбирается SaaS Git хостинг исходного кода. Такие сервисы предоставляют удобный масштабируемый хостинг для Git-репозиториев с веб-интерфейсом для просмотра и редактирования кода, а также гибкими настройками доступа. Современные и простые способы организации процессов CI/CD и решения самых разных задач с их помощью.

В настоящее время на рынке имеется несколько лидирующих представителей, отличающихся в основном стоимостью и набором дополнительных инструментов[4].

Наиболее популярным из таких хостингов является GitHub от международной компании Microsoft. Для проектов с открытым исходным кодом сервис бесплатен. Функции приватизации и контроля доступа для команд предоставляются за отдельную плату. Для развёртывания проектов по методологии CI/CD, GitHub предоставляет инструмент Actions. Хранилище образов и пакетов предоставляется с ограничением и только проектам с открытым исходным кодом. В целом, данный хостинг лучше всего подходит для таких проектов и редко используется командами разработчиков в коммерческой сфере деятельности.

Следующим из таких хостингов является Bitbucket[5] от австралийской компании Atlassian. Данный хостинг более акцентирован на коммерческую разработку и предоставляет функции приватизации бесплатно для небольших команд. Возможности CI/CD аналогичны GitHub Actions, только предоставляются продуктом Bamboo. Хранилище обра-

зов и пакетов отсутствует. Bitbucket следует рассматривать, как альтернативу GitHub только для коммерческой разработки[6].

Одним из наиболее подходящих хостингов является GitLab от украинских разработчиков (ныне зарубежной компании GitLab Inc). Ключевой особенностью является то, что этот сервис изначально разрабатывался, как полноценная система для управления жизненным циклом программного обеспечения на всех этапах разработки. Благодаря этому в нём реализован расширенный набор функций для обеспечения полноценного рабочего окружения по методологии CI/CD. Бесплатно сервис предоставляет, как приватизацию и управление доступом к исходному коду, так и хранилища образов и пакетов. Возможности непрерывной интеграции и доставки включают в себя отчеты о тестировании в реальном времени, параллельное выполнение, локальный запуск скриптов, поддержку Docker[7].

Так же был рассмотрен продукт от разработчиков из Санкт-Петербурга (международной компании JetBrains) — Space[8]. Данное решение является интегрированной средой для командной работы, которая включает управление исходным кодом, постановку и работу над задачами, управление командами разработчиков и инструменты коммуникации. Все функции Space так же предоставляет бесплатно, но с ограничениями в разумных пределах, подходящими небольшим командам разработчиков. Несмотря на большое количество возможностей, данный продукт существует на рынке сравнительно небольшое количество времени и большинство ключевых функций ещё не реализовано.

Для удобства конкретные данные о хостингах были приведены в таблице 1.1. Все данные приведены с учётом бесплатного тарифа использования хостингов.

Таблица 1.1 — Сравнение Git хостингов

Функция	GitHub	BitBucket	GitLab	Space
Настройки доступа	Огр.	+	+	+
Хранилище контейнеров и пакетов	500 Mb	-	10 Gb	10 Gb

Инструменты CI/CD	Огр.	Огр.	Огр, +	Огр.
Количество пользователей	Неогр.	5	Неогр.	Неогр.

1.3 Инструмент оркестрации

Помимо git хостинга данному проекту так же потребуется механизм масштабирования системы. Технология контейнеризации (Docker) позволяет запускать приложения в отдельных независимых средах — контейнерах. Они упрощают развертывание приложений, изолируют их друг от друга и ускоряют разработку. Но когда контейнеров становится слишком много, ими трудно управлять. Тут на помощь приходят IaC системы оркестрации [9].

У Docker есть стандартный инструмент оркестрации — Docker Swarm. Он поставляется вместе с Docker, довольно прост в настройке и позволяет создать кластер в кратчайшие сроки. Из минусов следует отметить узкую функциональность: возможности ограничены Docker API. Это значит, что Swarm способен сделать лишь то, что позволяют возможности Docker[10].

Альтернативным решением является Kubernetes — это универсальное средство для создания распределенных систем. Это комплексная система с большим количеством возможностей, что также является и минусом, поскольку поддержка такой системы значительно усложняется. Kubernetes мощный инструмент, имеющий много возможностей, которые позволяют строить действительно комплексные распределенные системы. К минусам относится управление, как оно использует отдельный набор команд и инструментов, несовместимых с Docker CLI[11]. Гибкость данного решения излишне и лишь затруднит введение проекта в рабочее состояние в контексте данной работы.

Подводя итоги исследования доступных решений систем автоматизации управления жизненным циклом веб-сервисов, можно сделать следующие выводы:

— анализ источников показывает, что на рынке отсутствуют подходящие под требования проектируемой системы готовые решения, позволяющие осуществлять дальнейшие доработки системы под конкретные задачи;

— анализ использованных источников показывает, что цели и задачи работы возможно реализовать на основе Git хостинга GitLab, разработав программные механизмы системы. Правильный выбор сервисных компонентов системы позволит выполнить требования технического задания и достигнуть цели работы.

2 ПРОЕКТИРОВАНИЕ СИСТЕМЫ АВТОМАТИЗАЦИИ УПРАВЛЕНИЯ ЖИЗНЕННЫМ ЦИКЛОМ ВЕБ-СЕРВИСА

2.1 Анализ требований к системе

Веб-сервис представляет классическое приложение на базе архитектуры «Клиент-Сервер». Пользователь получает доступ к сервису посредством веб-браузера, запрашивающего HTML страницу у SSR сервиса web-client, содержание которой зависит от сервиса api. В этих двух сервисах используется общая библиотека исходного кода common. В качестве хранилища данных используется СУБД PostgreSQL 14.1. Данная развёртка представлена на рисунке 2.1.

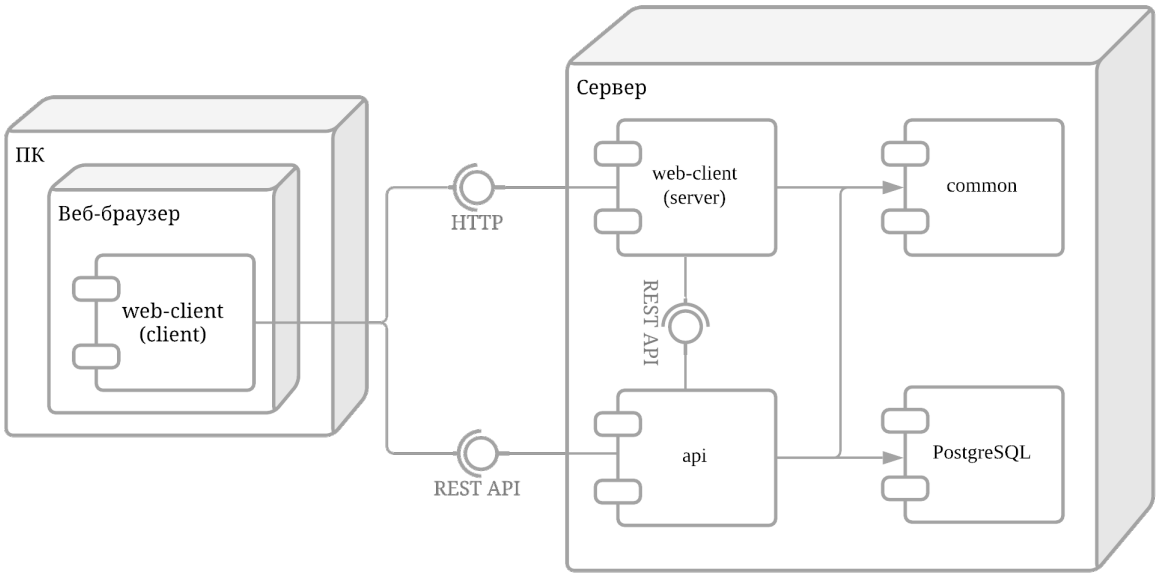

Рисунок 2.1 — Диаграмма развёртывания веб-сервиса

Так как полученный для развёртывания веб-сервис базируется на Node.js и исходный код разбит на библиотеки, то для работы системы потребуется регистр Node пакетов. Так же необходимым будет регистр Docker образов.

Согласно требованиям были сформулированы основные действующими лица (актёры) в работе системы:

— администратор — пользователь занимающийся настройкой прав доступа другим пользователям и конфигурацией развёртывания веб-сервиса,

— разработчик — пользователь системы имеющий доступ к репозиториям с исходным кодом, хранилищам пакетов и контейнеров, а так же управлению релизами веб-сервиса,

— сотрудник отдела качества (тестировщик) — пользователь системы имеющий доступ к просмотру аналитических данных и проведению автоматизированных тестовых сценариев внутри заранее подготовленных окружений.

Пользователь системы подразумевает любого актёра. На основании описания актёров и их основных возможностей была составлена диаграмма случаев использования. Изображение данной диаграммы представлено на Рисунке 2.2.

Согласно требованиям системой должно поддерживаться три основных рабочих окружения веб-сервиса под различные цели:

— develop — инсценировка рабочего окружения веб-сервиса для разработчиков,

— testing — окружение для проведения ручного тестирования и сбора аналитических данных,

— release — рабочее окружение веб-сервиса для реальных пользователей.

С точки зрения CI/CD взаимодействие пользователя с системой сосредоточено вокруг коммита (Commit) в репозиторий и автоматическим запуском одной или нескольких задач (Jobs) внутри определённой линии (Pipeline). Каждая задача является набором последовательно исполняемых инструкций ожидаемо завершённых без ошибок. В случае ошибки выполнение всей линии завершается и повторяется только по запросу пользователя. При этом линии задач строятся динамически в зависимости от конкретного репозитория и ветки. Результатами работы линий являются артефакты, которые содержат основную информацию

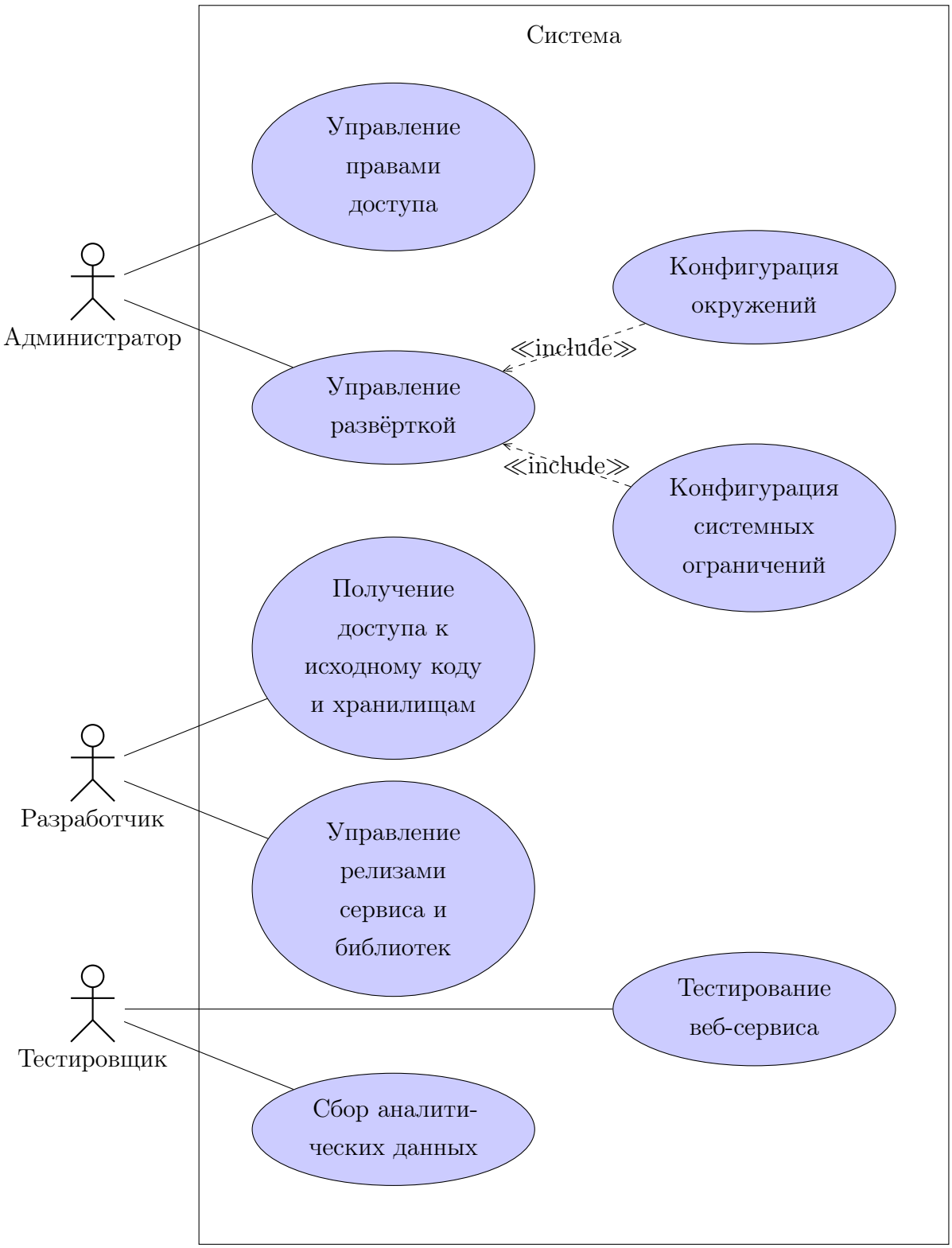

Рисунок 2.2 — Диаграмма случаев использования системы

о результатах работы задачи. Данное поведение системы в общем виде отражено на Рисунке 2.3.

Так как линия задач зависит от репозитория, то необходимо систематизировать репозитории в системе:

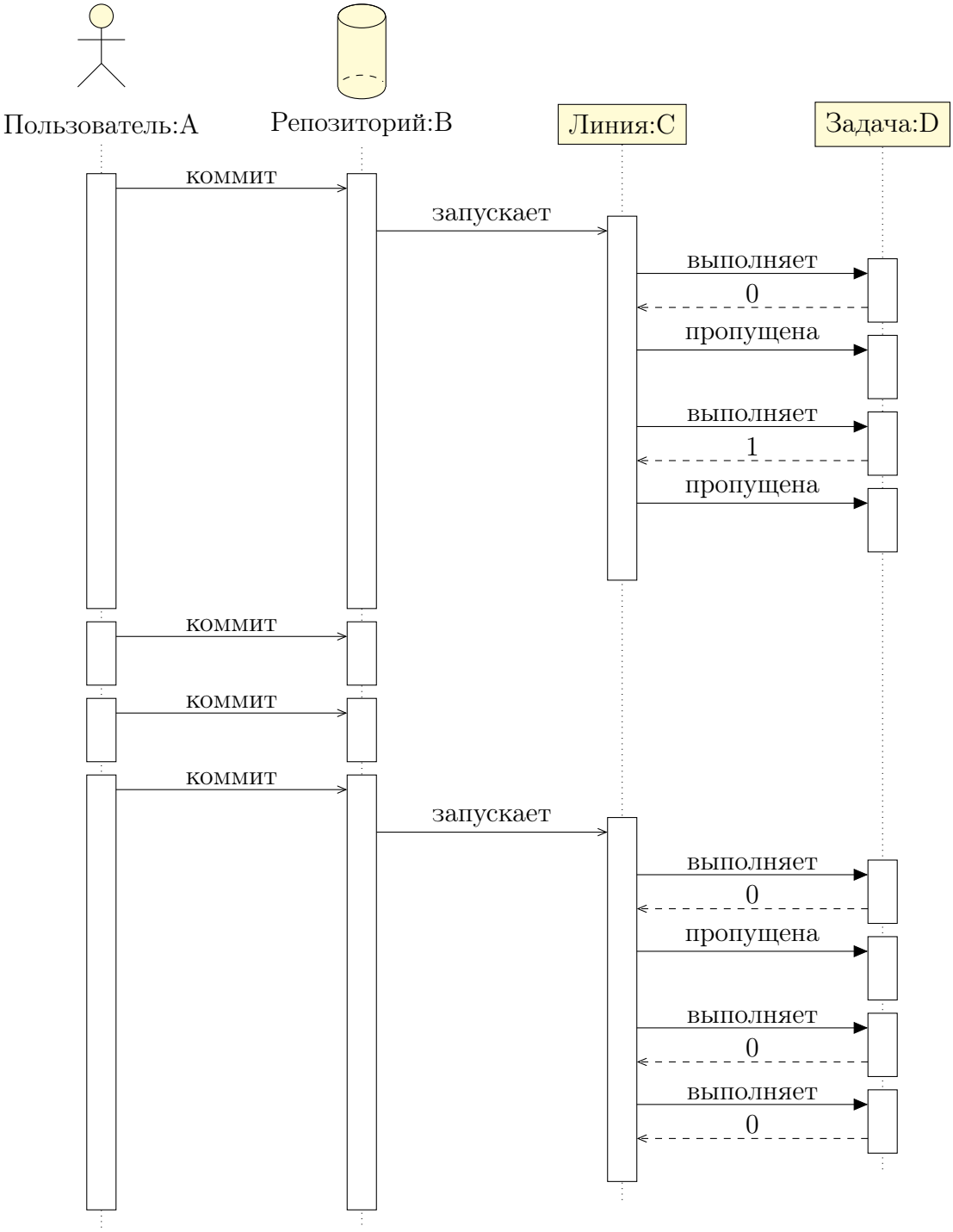

Рисунок 2.3 — Поведение системы в общем виде

— репозиторий с исходным кодом компонента веб-сервиса (Source repository на рисунке 2.4) — не уникален, обязательно содержит Dockerfile в корне, предоставляется полный доступ разработчикам, доступ к аналитическим данным тестировщику,

— репозиторий с конфигурациями развёртывания (deployment на рисунке 2.4) — уникален, содержит общие скрипты и конфигурации, предоставляется доступ только администратору,

— репозиторий с исходным кодом библиотек компонентов веб-сервиса (node-packages на рисунке 2.4) — уникален, предоставляется полный доступ разработчикам.

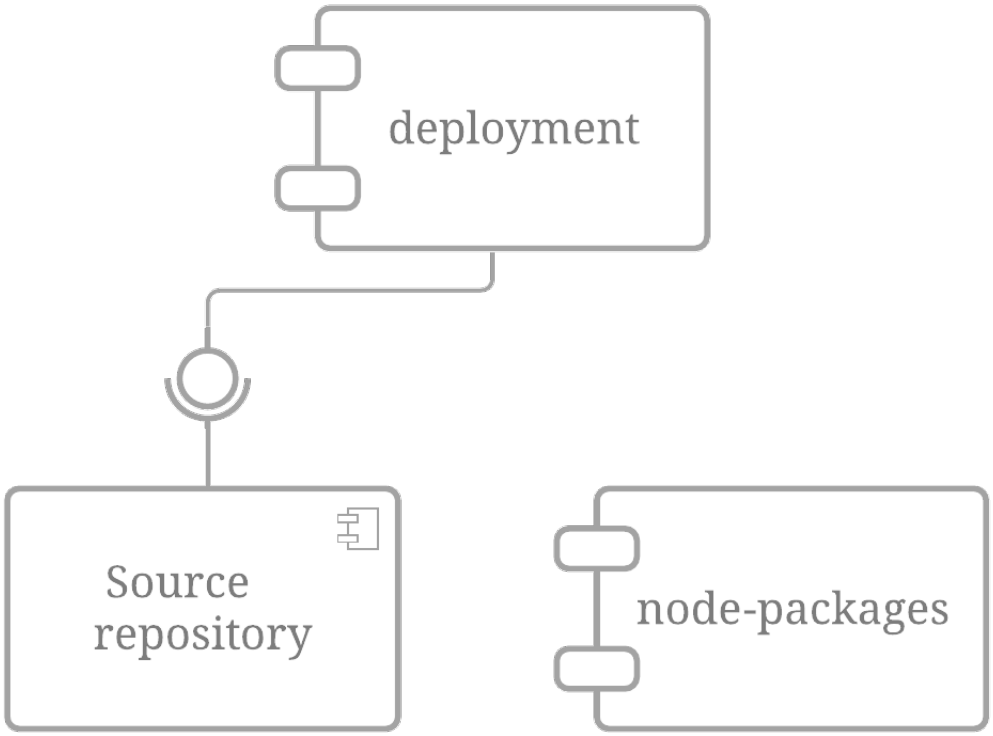

Рисунок 2.4 — Диаграмма компонентов репозиториев

Таким образом, основными объектами в системе являются:

— пользователь с набором прав доступа,
— репозиторий одного из типов,
— линия с задачами.

2.2 Проектирование линий задач

Для проектирования линий задач системы далее будет приведён разбор случаев использования с точки зрения CI/CD.

В рамках данной работы ставится задача развернуть и настроить систему по автоматизации управления жизненным циклом веб-сервиса,

включающую автоматизированное тестирование компонентов веб-сервиса.

Основной целью такого тестирования является снижение количества ошибок в работе веб-сервиса и как следствие повышение соответствия к заявленным требованиям. Данная цель будет достигаться при помощи двух основных линий, разделённых по виду тестирования и представленных в таблице 2.1.

Таблица 2.1 — Линии тестирования

Название вида	Условие запуска	Местонахождение	Типы ошибок
Модульное	Каждый коммит	Репозитории с исходным кодом	Семантические, компиляции, логические на уровне модуля
Интеграционное	Каждый день в 10 утра	Репозиторий с конфигурациями развёртывания	Логические на уровне веб-сервиса

Вследствие этого командой контроля качества были подготовлены автоматизированные тестовые сценарии. Данные инструменты представляют консольные приложения на базе Node.js, завершающихся с не нулевым кодом в случае нахождения ошибки.

— проведение автоматизированного тестирования — линия будет срабатывать на каждую синхронизацию с основной веткой удаленного репозитория и проводить разные виды тестирования,

— управление релизами сервиса и библиотек — линия будет составляться в зависимости от ветки репозитория, собирать исходный код, загружать готовый к работе код в хранилище и оповещать кластер о выходе обновления при необходимости,

— управление развёрткой — линия будет заключаться в применении обновлённых конфигураций окружения из репозитория к кластеру,

— получение доступа к исходному коду и хранилищам — случай использования будет реализовываться не средствами CI/CD,

— сбор аналитических данных — данные будут предоставляться артефактами в результате работы линий задач,

— управление правами доступа — случай использования будет реализовываться не средствами CI/CD.

Самым частым этапом является проведение автоматизированного тестирования. Так как система заранее не может знать о возможных сценариях использования веб-сервиса, то вся ответственность об их содержании переносится на тестировщика. В целом, процесс тестирования будет происходить в несколько основных этапов: семантическое тестирование исходного кода, юнит и интеграционное тестирование библиотек и определённых сервисов компонентов системы. На основании данных этапов была составлена линия задач процесса тестирования веб-сервиса:

— lint — семантическое тестирование исходного кода путём запуска встроенного скрипта модуля разработчиками,

— test — юнит и интеграционное тестирование библиотек веб-сервиса путём запуска скрипта модуля разработчиками и предоставление артефактов выполнения,

Создание релиза будет происходить похоже на создание релиза в git flow, только к комитам привязаны действия линий задач: исполнение комита при помощи git, проведение тестирования, сборка образа и оповещение кластера. На основании данных этапов была составлена линия задач создания релиза веб-сервиса:

— test — осуществление контроля качества путём запуска задач линии тестирования,

— build — сборка исходного кода нужной версии и загрузкой в хранилище в зависимости от требуемого окружения (опциональная задача, требует подтверждения пользователем),

— publish — оповещение кластера или зависимых сервисов о выходе обновления.

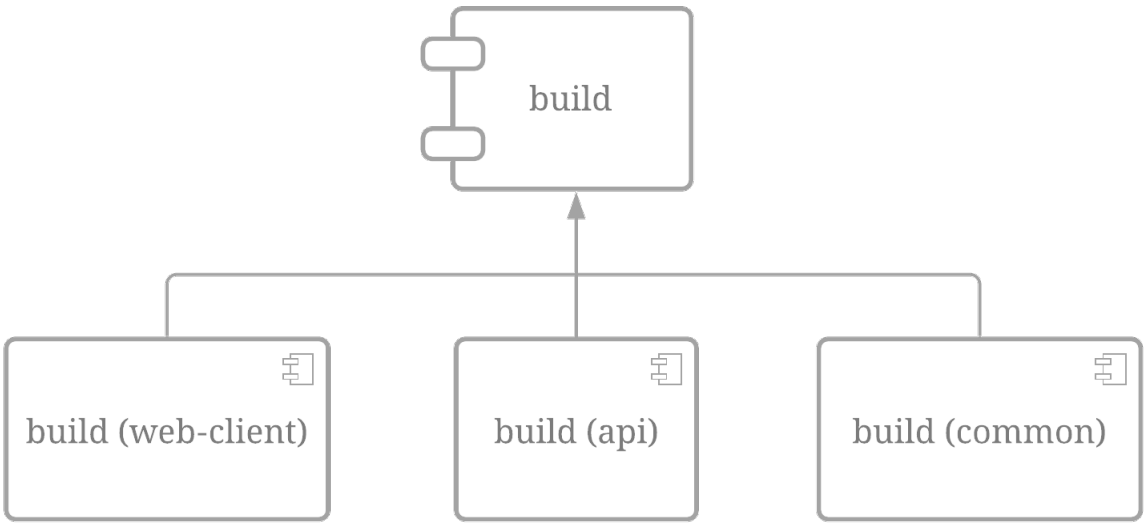

Рисунок 2.5 — Диаграмма объектов стадии build

Данная линия будет иметь следующие аргументы:

— название сервиса компонента, который необходимо обновить;
— название окружения, в котором необходимо произвести релиз.

Так как шаги сборки исходного кода и оповещения о релизе зависят лишь от входных инструкций Dockerfile, то данные шаги будут скрыты от разработчиков веб-сервиса в репозитории с конфигурациями развёртывания. Разработчику необходимо будет только импортировать необходимые задачи из репозитория.

Конфигурация развёртывания веб-сервиса состоит из линии, включающей только одну задачу: применение конфигураций развёртывания к кластеру. В целях структуризации конфигураций для этих целей будет использоваться задача publish из линии по созданию релиза, которая при отсутствии аргументов будет обновлять конфигурации рахвёртки кластера.

2.3 Выбор сервисов архитекутры

Проводя обзор доступных на рынке git хостингов, можно сделать вывод, что наиболее распространенным git хостингом на сегодняшний день является хостинг компании GitLab Inc. К тому же, по соотношению цена-функционал хостинг этой компании существенно обходит кон-

курентов. Также, к существенному преимуществу можно отнести наличие обширного сообщества пользователей и разработчиков программных решений на основе git хостинга GitLab, что позволяет иметь доступ к множеству готовых решений и получать помощь в разработке при необходимости.

Для реализации поставленной в данной работе задачи гибкость настройки всей инфраструктуры окружения не требуется, а также ставится в приоритет скорость ввода в рабочее состояние. Поэтому в качестве оркестратора вместо гибкости Kubernetes был выбран Docker Swarm[12]. Но так как в данной работе делается акцент на гибкость всей системы, то далее будет рассмотрена описание конфигурации для использования с Kubernetes, не считая уставноку и настройку самого кластера. Для работы будет подготовлена одна вершина Docker Swarm в статусе менеджер, поскольку более не требуется на данном этапе.

Одной и ключевой настройкой является открытие портов на уровне операционной системы сервера[13]:

— TCP порт 2377 для коммуникации между менеджерами кластера,

— TCP и UDP порты 7946 для взаимодействия между нодами кластера,

— UDP порта 4789 для управления сетевым трафиком.

Как было сказано ранее, для развёртывания в работе используется Docker, поэтому необходим простой инструмент удалённого доступа к сокету Docker сервиса на рабочем сервере. Для этих целей будет использоваться GitLab Runner с установленным исполнителем задач Docker. Кратко описать работу GitLab Runner можно следующим образом: GitLab Runner запускается в отдельном контейнере с добавленным volume на сокет Docker, таким образом получается избежать достаточно сложного и нецелесообразного запуска Docker внутри Docker, так как в этом случае GitLab Runner получает доступ напрямую к сокету Docker сервера.

Данное решение имеет потенциальную проблему с безопасностью, поскольку если злоумышленник получит доступ к описанию задач

GitLab CI/CD, то он сможет запускать на рабочем сервере любое ПО. Для избежания данной проблемы будут установлены настройки доступа внутри GitLab. Так же для избежания потери полезного времени работы GitLab Runner, необходимо будет произвести настройку кэша GitLab Runner. Ключевой настройкой является политика загрузки образов для задач, поскольку по умолчанию GitLab Runner в любом случае будет загружать образ из регистра, даже если образ представлен локально. Согласно требованиям GitLab Runner должен будет запускать минимум три задачи за единицу времени, данное значение будет отражено в конфигурации на этапе реализации.

2.4 Составление плана тестирования

Перед началом составления плана сразу стоит отметить, что тестирование будет проводиться самим же разработчиком системы, что нарушает один из основных принципов тестирования. Но в данном случае это оправдано, поскольку работа выполняет одним человеком.

Так же автор работы имеет доступ к исходному коду веб-сервиса и имеет открытый доступ к алгоритмам работы системы, что определяет метод тестирования — белого ящика.

Основная цель проведения данного тестирования — это снижение количества ошибок в работе линий и задач, а так же конфигурационных файлов. Отсюда следует набор основных ошибок для нахождения при поведении тестирования: синтаксические в описании конфигураций и логические в работе линий и задач. Так как GitLab предоставляет графический веб-интерфейс, то проведение тестирования на разных браузерах и операционных системах нецелесообразно, поскольку сбор такого типа ошибок не удовлетворяет поставленной цели.

Ключевыми факторами, определяющими результат тестового сценария являются:

а) репозиторий;

б) хеш коммита в репозитории.

Тестовый сценарий не может считаться пройденным или нет в случае, если GitLab Runner не отвечает на входящие запросы, поскольку это не содержит информации о результате выполнения линий задач, но длительный простой задач в случае корректной работы линий свидетельствует о перегруженности сервера.

Основную часть синтаксических и логических ошибок можно найти и исправить на этапе описания файлов конфигураций, поскольку GitLab предоставляет удобный веб эмулятор GitLab Runner. Использование данного эмулятора не поможет при поиске более сложных логических ошибок, поскольку задачи не будут исполняться, тем не менее его применение значительно ускоряет рабочий процесс разработки и описания файлов конфигураций.

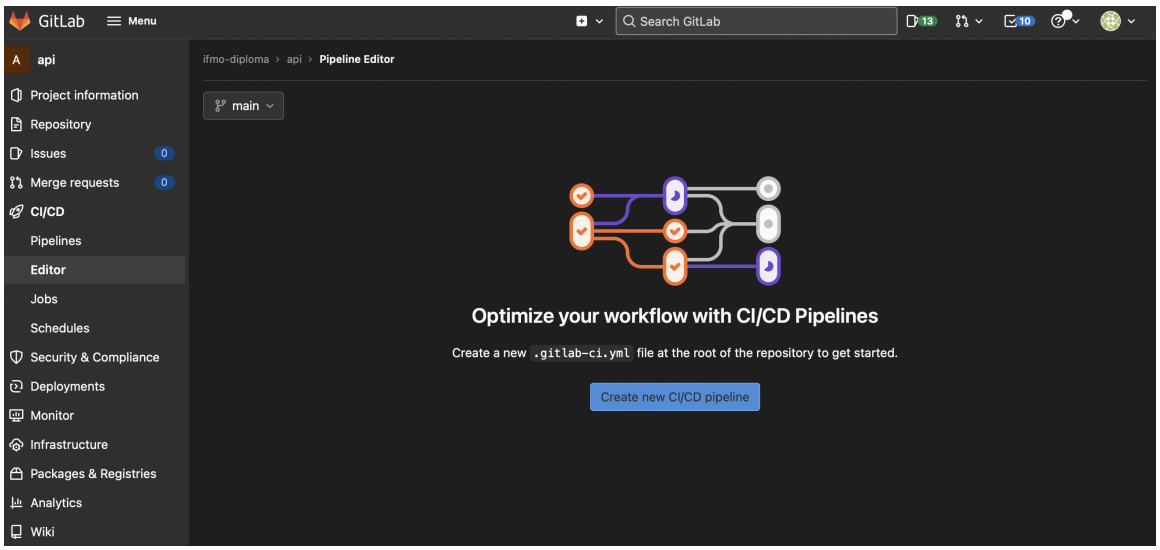

Рисунок 2.6 — GitLab эмулятор линий и задач

Для нахождения и последующего исправления комплексных логических, требующих запуска задач, был составлен план тестирования в виде таблицы 2.2. Тестовые сценарии в плане основаны на случаях использования системы

Таблица 2.2 — План тестирования

№	Название тестового сценария	Тестовый сценарий	Тестовые данные	Ожидаемый результат

1	Линия задач автоматизации тестирования	Открыть репозитории сервисов, проверить срабатывание и составление линий автоматизированного тестирования	Список сервисов компонентов веб-сервиса	Линии собираются правильно, задачи выполняются, артефакты предоставляются
2	Линия задач релиза создания релиза	Открыть репозитории сервисов, проверить срабатывание и составление линий создания релиза	Список сервисов компонентов веб-сервиса	Линии собираются правильно, задачи выполняются, веб-сервиса обновляется
3	Линия задач управления развёрткой	Открыть репозиторий развёртывания, изменить конфигурации развёртывания, зафиксировать изменения коммитом	Конфигурации развёртывания веб-сервиса	Веб-сервис правильно реагирует на изменение конфигураций развёртывания

3 РЕАЛИЗАЦИЯ СИСТЕМЫ АВТОМАТИЗАЦИИ УПРАВЛЕНИЯ ЖИЗНЕННЫМ ЦИКЛОМ ВЕБ-СЕРВИСА

3.1 Подготовка GitLab

Перед реализацией необходимо произвести базовую настройку окружения. Для этого был зарегистрирован GitLab аккаунт, установлен SSH ключ и создана группа проекта. Результаты создания группы представлены на рисунке 3.1.

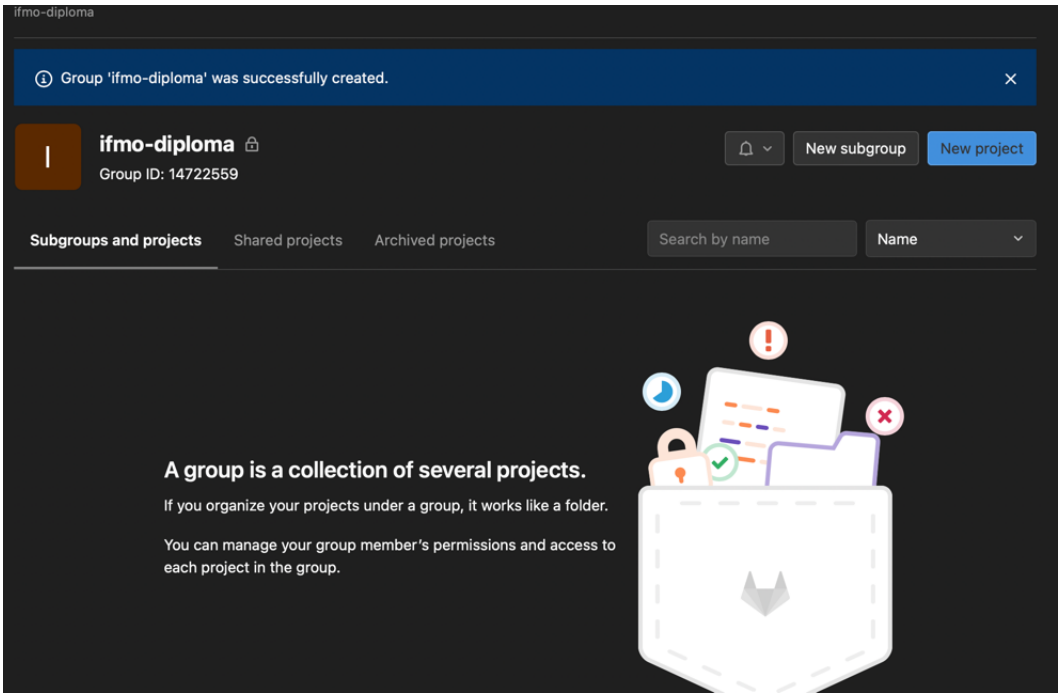

Рисунок 3.1 — Скриншот создания группы в GitLab

Следующим этапом было произведено создание необходимых репозиториев с последующей загрузкой исходного кода предоставленного для работы веб-сервиса:

— web-client — репозиторий для хранения исходного кода веб-клиента проекта,

— api — репозиторий для хранения исходного кода API проекта.

— node-packages — репозиторий для хранения исходного кода общих npm зависимостей проекта.

Так же в корне каждого репозитория был загружен Dockerfile для развёртывания данного сервиса. Так как в качестве модели ветвления была выбрана модель git flow, то так же были подготовлены соответствующие ветки в репозиториях под разные окружения: develop, testing и release. Установка доступа к данным репозиториям предоставляется только разработчикам данных программных решений в целях безопасности.

Подготовка хранилищ npm пакетов и Docker образов не требуется, так как GitLab берёт на себя данную ответственность и не требует дополнительных действий от пользователя.

На следующем этапе был подготовлен репозиторий deployment для хранения общих скриптов и конфигураций окружения и развёртывания. Установка доступа к этому репозиторию предоставляется только команде обеспечения развёртывания в целях безопасности. Результаты создания репозиториев представлены на рисунке 3.2.

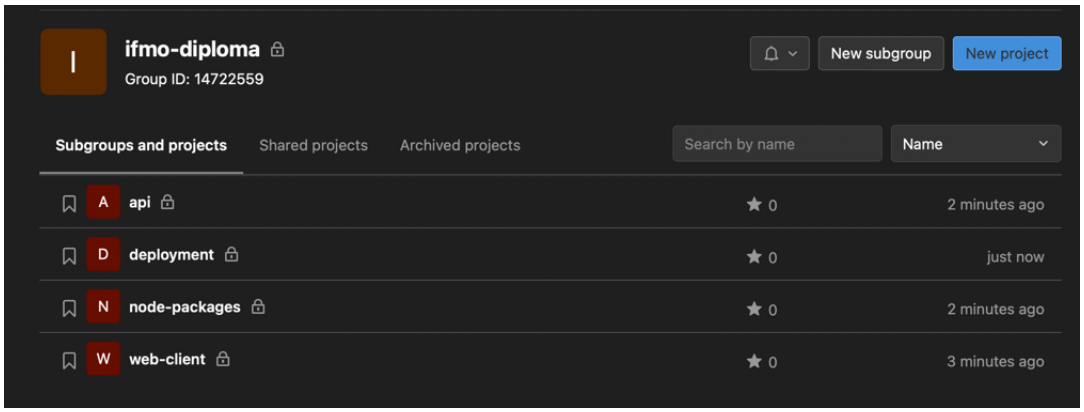

Рисунок 3.2 — Скриншот создания репозиториев в GitLab

3.2 Установка кластера Docker Swarm

Следующим этапом был установлен и настроен Docker Swarm кластер.

Для этого на рабочей системе было произведено открытие необходимых портов в операционной системе[14]:

Листинг 3.1 — Открытие портов в Linux

```
1 $ sudo ufw allow 2377
```

```
2 $ sudo ufw allow 7946
3 $ sudo ufw allow 4789
```

Дальше была произведена инициализация кластера на сервере:

Листинг 3.2 — Инициализация кластера

```
1 $ docker swarm init --advertise-addr 192.168.1.101
2   Swarm initialized: current node (dxn1zf6l61qsb1josjja83ngz) is now a
        manager.
3   To add a worker to this swarm, run the following command:
4     docker swarm join --token <token> 192.168.1.101:2377
5   To add a manager to this swarm, run 'docker swarm join-token manager'
        and follow the instructions.
```

В данном этапе нет необходимости при использовании Kubernetes.

3.3 Установка и настройка GitLab Runner

После была произведена установка и настройка GitLab Runner на рабочем сервере. В качестве дистрибутива на этапе проектирования был выбран Docker, как самый быстрый и удобный. После установки необходимо зарегистрировать GitLab Runner, для этого на странице настроек CI/CD группы в GitLab был получен регистрационный токен. Конфигурация runner производится путём редактирования config.toml[15] файла в соответствии с этапом проектирования, основными настройками являются:

— Установка executor — docker executor,
— Установка volumes — /var/run/docker.sock[16],
— Установка pull-policy — if-not-present,
— Установка concurrent — 3.

На данном этапе runner полностью готов к работе и ожидает входящих задач.

3.4 Описание CI/CD конфигураций

На следующем этапе необходимо подготовить общие для работы сервисов конфигурации запуска, которые будут храниться в репозитории deployment:

— build.yaml — набор универсальных задач, предназначенный для сборки Docker образа и последующей загрузки в регистр контейнеров,

— publish.yaml — набор универсальных задач, предназначенный для оповещения кластера об обновлении сервиса для загрузки новой версии,

— publish-api.yaml — расширенная версия publish.yaml, содержащая дополнительные задачи для проведения миграция базы данных в случае необходимости,

Одним из наиболее интересных файлов конфигураций является .gitlab-ci.yaml, поскольку в нём содержится основная логика управления развёрткой веб-сервиса. На данных строчках содержится автоматическое подключение к кластеру Docker образов без с учётом риска передачи паролей через переменные окружения. Так же в данных строчках содержится описание стадий задач, необходимых для развёртки сервиса api. Пример приведён на листинге 3.3.

Листинг 3.3 — Задачи сервиса api (**deployment/.gitlab-ci.yml**)

```
image: docker/compose:alpine-1.29.2

before_script:
  - cat ${DOCKER_REGISTRY_WRITE_BOT_PASSWORD} | docker login -u
      ${DOCKER_REGISTRY_WRITE_BOT_LOGIN} --password-stdin
      registry.gitlab.com/itmo-diploma/web-client

migrate_db:
  stage: "Migrate DB"
  tags:
    - orchestrator
  only:
    variables:
      - $MIGRATE_IMAGE_TAG
    refs:
      - master
  script:
    - docker pull registry.gitlab.com/itmo-diploma/api:"$MIGRATE_IMAGE_TAG"
    - docker run --rm --net=itmo-diploma_"$BRANCH"_backend
        registry.gitlab.com/itmo-diploma/api:"$MIGRATE_IMAGE_TAG" npm run
        db:migrate:"$BRANCH"

# Upload REST API interface
deploy_endpoints:
  stage: "Deploy Endpoints"
```

```
22    tags:
23      - orchestrator
24    only:
25      variables:
26        - $DEPLOY_ENDPOINTS
27      refs:
28        - master
29    script:
30      - docker-compose -p itmo-diploma_"$BRANCH" -f
          docker-compose."$BRANCH".yaml up --no-deps "endpoints"
```

Данная задача используется для безопасного обновления конкретного сервиса по его названию и названию окружения, которая добавляется в линию только при указании значения названия сервиса и ветки. Пример приведён на листинге 3.4.

Листинг 3.4 — Обновление сервиса (**deployment/.gitlab-ci.yml**)

```
1  update_service:
2    stage: "Update a service"
3    tags:
4      - orchestrator
5    only:
6      variables:
7        - $SERVICE && $BRANCH
8      refs:
9        - master
10   script:
11     - docker-compose -f docker-compose."$BRANCH".yaml pull "$SERVICE"
12     - docker-compose -f docker-compose."$BRANCH".yaml up -d --no-deps
         "$SERVICE"
```

В конце файла содержится задача, условием запуска которой является отсутствие названия сервиса и веток. Данная задача используется для экстренного перезапуска веб-сервиса. Пример приведён на листинге 3.5.

Листинг 3.5 — Перезапуск веб-сервиса (**deployment/.gitlab-ci.yml**)

```
1  update:
2    stage: "Update"
3    tags:
4      - orchestrator
5    only:
6      - master
7    except:
```

```
8       variables:
9         - $SERVICE
10        - $BRANCH
11        - $MIGRATE_IMAGE_TAG
12        - $DEPLOY_ENDPOINTS
13    script:
14      - docker-compose -p itmo-diploma_dev -f docker-compose.dev.yaml up -d
            --build
15      - docker-compose -p itmo-diploma_master -f docker-compose.master.yaml
            up -d --build
16
17  stages:
18    - "Deploy Endpoints"
19    - "Migrate DB"
20    - "Update a service"
21    - "Update"
```

Далее была произведена конфигурация на уровне сервисов компонентов, в репозитории api и web-client были добавлены конфигурации CI/CD путём создания .gitlab-ci.yml файла, содержащего основные задачи.

Для репозитория node-packages были описаны задачи с использованием CLI lerna, которая предназначена для организации работы монорепозитория. Использование этой библиотеки обусловлено большим сообществом разработчиков и открытостью исходного кода. Данная программа обновляет версии библиотек при помощи построения и обхода графа зависимостей, построенного на основании package.json файлов. Пример данного графа в условиях данного веб-сервиса приведён на рисунке 3.3.

3.5 Развёртка сервисов внутри кластера

Завершающим этапом реализации является описание конфигурационных файлов Docker Swarm. Для этого в настройках CI/CD репозитория deployment были добавлены переменные окружения (секреты) на все рабочие окружения (develop, testing и release), содержащие аргументы сервисов компонентов системы (доступы к базе данных, секрет ключа авторизации и так далее)[17]. Аналогично были добавлены Docker Swarm конфигурационные файлы под каждый окружения:

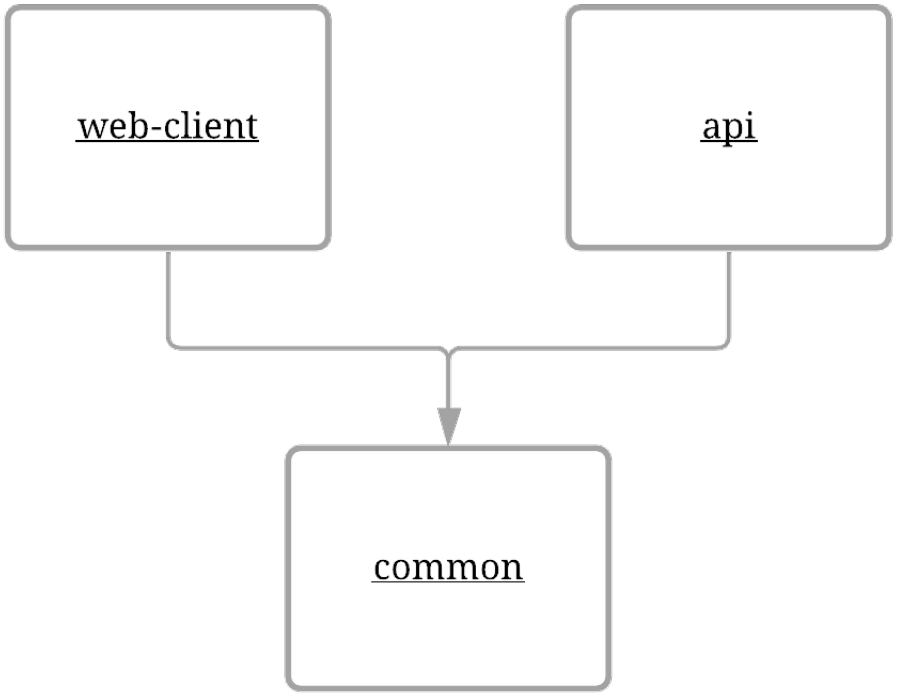

Рисунок 3.3 — Граф зависимостей компонентов веб-сервиса

— develop — каждый сервис запускается на сервере в одном экземпляре, ресурсы сервера сильно ограничены во избежание лишней нагрузки,

— testing — аналогичен develop, только используется для целей ручного тестирования веб-сервиса,

— release — web-client запускается в одном экземпляре, база данных и api запускаются в трёх, большая часть ресурса отведена под эти сервисы.

На листинге 3.6 приведён пример части конфигурационного файла Docker Swarm для запуска сервиса api.

Листинг 3.6 — Сервисы api (**deployment/docker-compose.master.yaml**)

```
1 version: '3.7'
2
3 services:
4   production-db:
5     build:
6       context: ./postgres
7       args:
8         ENV: production
9     environment:
```

```
10        POSTGRES_PASSWORD: ${MASTER_POSTGRES_PASSWORD}
11        POSTGRES_USER: postgres
12        POSTGRES_DB: postgres
13        PGDATA: /var/lib/postgresql/data/pgdata
14      volumes:
15        - /data:/var/lib/postgresql/data:z
16      networks:
17        - backend
18
19    production-api:
20      pull_policy: always
21      image: registry.gitlab.com/psytechapp/api:latest-dev
22      restart: always
23      environment:
24        NODE_ENV: "production"
25        SENDGRID_API_KEY: ${SENDGRID_API_KEY}
26      networks:
27        - esp
28        - backend
29        - nginx-network
30
31    production-endpoints:
32      build:
33        context: ./gcloud
34      entrypoint: sh -c \
35        "/usr/local/wait-for -t 30 production-api:3000 -- echo 'API is
              ready' &&
36        curl http://production-api:3000/api/v1/json | jq '.' >
              ./swagger.json &&
37        cat ./swagger.json &&
38        gcloud endpoints services deploy ./swagger.json"
39      networks:
40        - backend
41
42    production-esp:
43      image: gcr.io/endpoints-release/endpoints-runtime:2
44      command:
45        - "--service=api.psytech.app"
46        - "--rollout_strategy=managed"
47        - "--listener_port=3000"
48        - "--backend=http://production-api:3000"
49      networks:
50        - nginx-network
51        - esp
52
53    production-web-client:
```

```
    pull_policy: always
    image: registry.gitlab.com/psytechapp/web-client:latest-dev
    restart: always
    environment:
      NODE_ENV: "production"
      WEB_SOCKET_API_URL: "https://socket.psytech.app"
      BASE_INTERNAL_API_URL: "http://production-api:3000"
      CONTACT_PHONE_NUMBER_TEXT_RU: ${CONTACT_PHONE_NUMBER_TEXT_RU}
      CONTACT_PHONE_NUMBER: ${CONTACT_PHONE_NUMBER}
    networks:
      - nginx-network

networks:
  backend:
    driver: bridge
    ipam:
      config:
        - subnet: 192.167.1.1/28
  esp:
    driver: bridge
    ipam:
      config:
        - subnet: 192.165.1.1/28
  nginx-network:
    external: true
```

На листинге ?? приведён пример части конфигурационного файла Docker Swarm для запуска сервиса web-client.

Листинг 3.7 — Сервисы web-client (**deployment/docker-compose.master.yaml**)

```
  production-web-client:
    pull_policy: always
    image: registry.gitlab.com/psytechapp/web-client:latest-dev
    restart: always
    environment:
      NODE_ENV: "production"
      WEB_SOCKET_API_URL: "https://socket.psytech.app"
      BASE_INTERNAL_API_URL: "http://production-api:3000"
      CONTACT_PHONE_NUMBER_TEXT_RU: ${CONTACT_PHONE_NUMBER_TEXT_RU}
      CONTACT_PHONE_NUMBER: ${CONTACT_PHONE_NUMBER}
    networks:
      - nginx-network

networks:
  backend:
    driver: bridge
```

```
    ipam:
      config:
        - subnet: 192.167.1.1/28
  esp:
    driver: bridge
    ipam:
      config:
        - subnet: 192.165.1.1/28
  nginx-network:
    external: true
```

В итоге веб-сервис был развёрнут, пример открытия веб страницы приведён на рисунке 3.4.

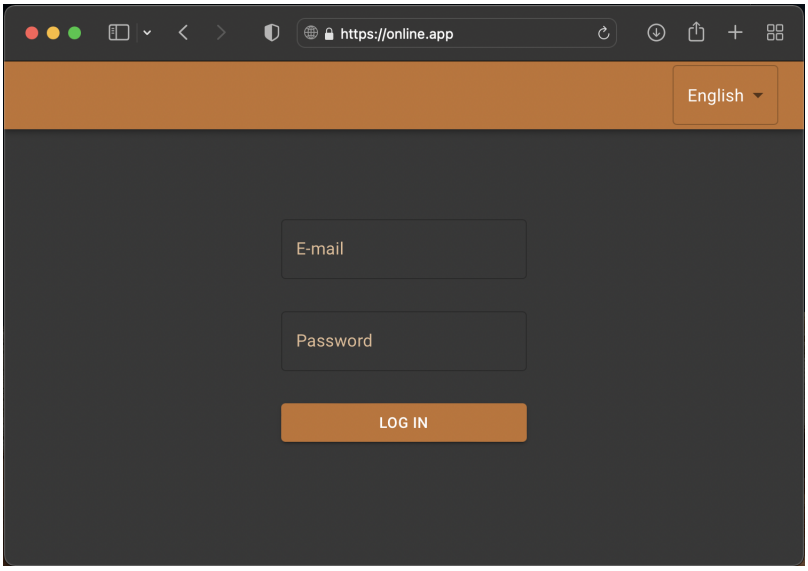

Рисунок 3.4 — Скриншот открытия страницы веб-сервиса

4 ТЕСТИРОВАНИЕ СИСТЕМЫ АВТОМАТИЗАЦИИ УПРАВЛЕНИЯ ЖИЗНЕННЫМ ЦИКЛОМ ВЕБ-СЕРВИСА

4.1 Проведение ручного тестирования

Тестирование проводилось в браузере Safari Version 15.2 (17612.3.6.1.6).

Перед проведением ручного тестирования каждая линия задач была проверена при помощи описанного ранее эмулятора GitLab, пример на рисунке 4.1.

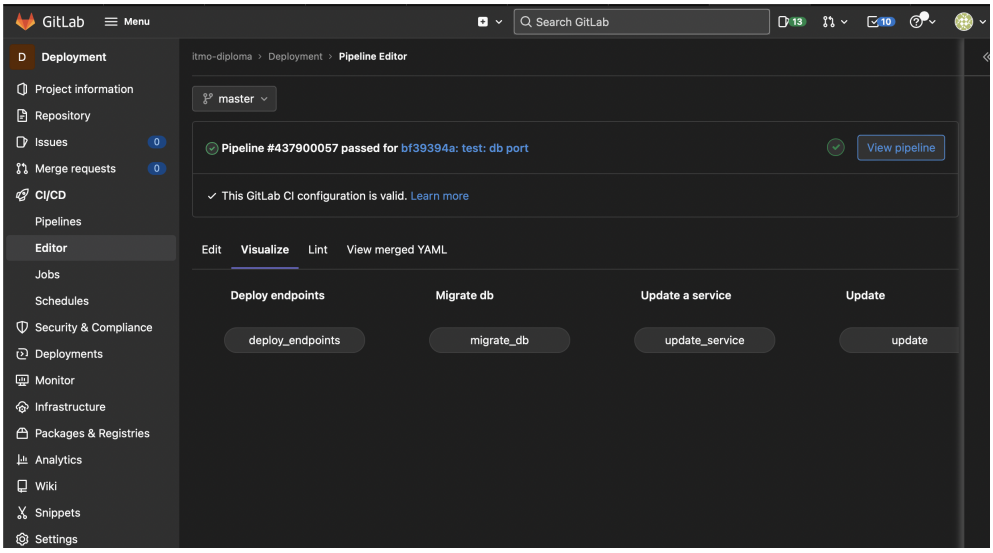

Рисунок 4.1 — Эмулятор GitLab с линией deployment

Описание прохождения сценариев тестирования в порядке описания в плане:

а) Для проведения данного сценария были рассмотрены следующие случаи: коммит не в основную ветку(не develop, testing или release) — были составлена линия только из задач тестирования, коммит в основную ветку(develop, testing или release) — были составлена линия из задач тестирования и задач создания релиза. Пример составленной линии представлен на Рисунке 4.2, а пример пройденных автоматических тестов с генерированным файлами артифектов на Рисунке 4.3.

б) Для проведения данного сценария были рассмотрены следующие случаи: создание релиза библиотеки — для этого был произведён коммит

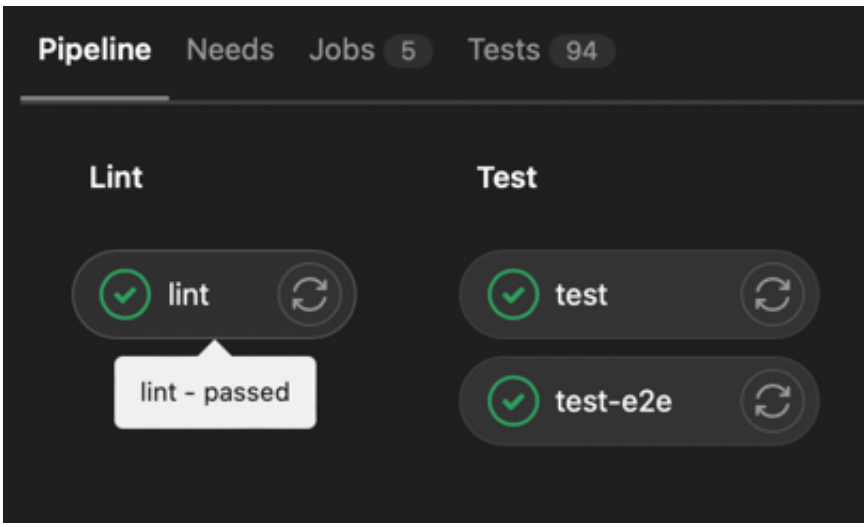

Рисунок 4.2 — Линия тестирования в GitLab

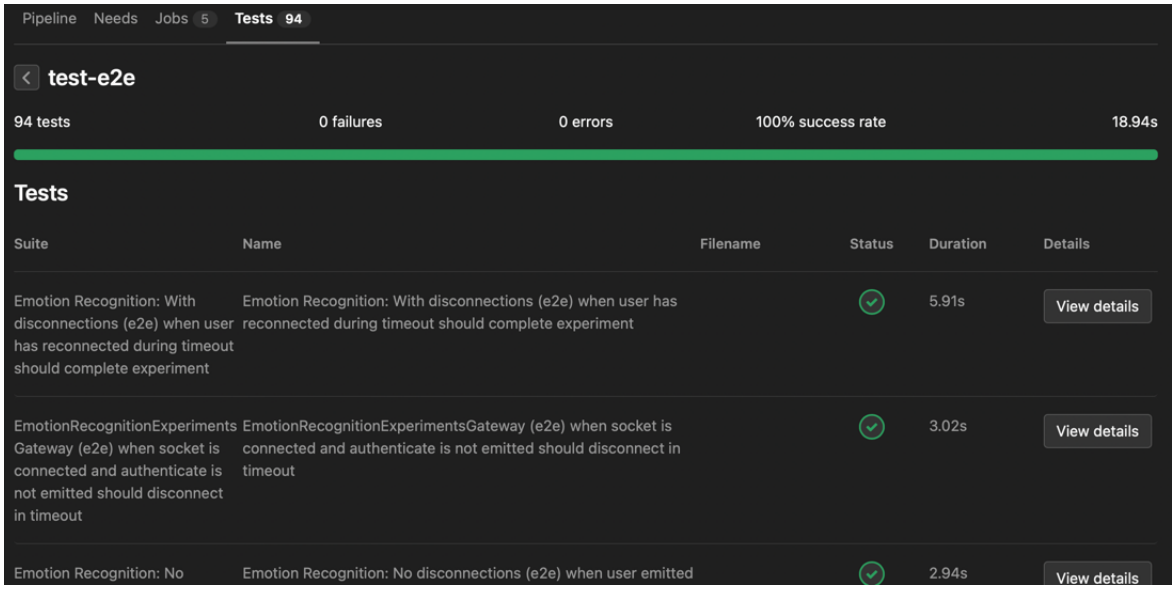

Рисунок 4.3 — Артефакты тестирования в GitLab

в ветку develop, в ответ была составлена линия из задачи отправки библиотеки в регистр, а так же задач сборки Docker образа и обновления кластера; создание develop релиза — для этого был произведён коммит в ветку develop, в ответ была составлена линия из задач тестирования, а так же задач сборки Docker образа и обновления кластера; создание testing релиза — для этого был произведён мёрж коммит в ветку testing из ветки develop, в ответ была составлена линия из задач тестирования, а так же задач сборки Docker образа и обновления кластера; создание release релиза — для этого был произведён мёрж коммит в ветку release

из ветки testing, в ответ была составлена линия из задач тестирования, а так же задач сборки Docker образа и обновления кластера;
Результаты тестирования представлены на скриншотах:

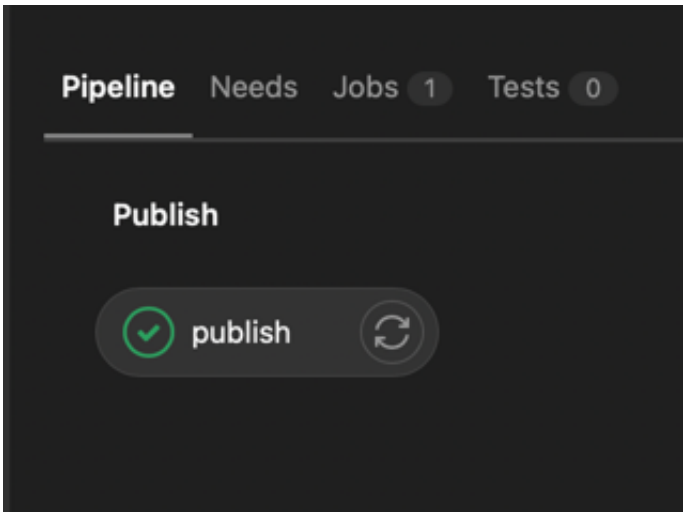

Рисунок 4.4 — Загрузка библиотеки в хранилище

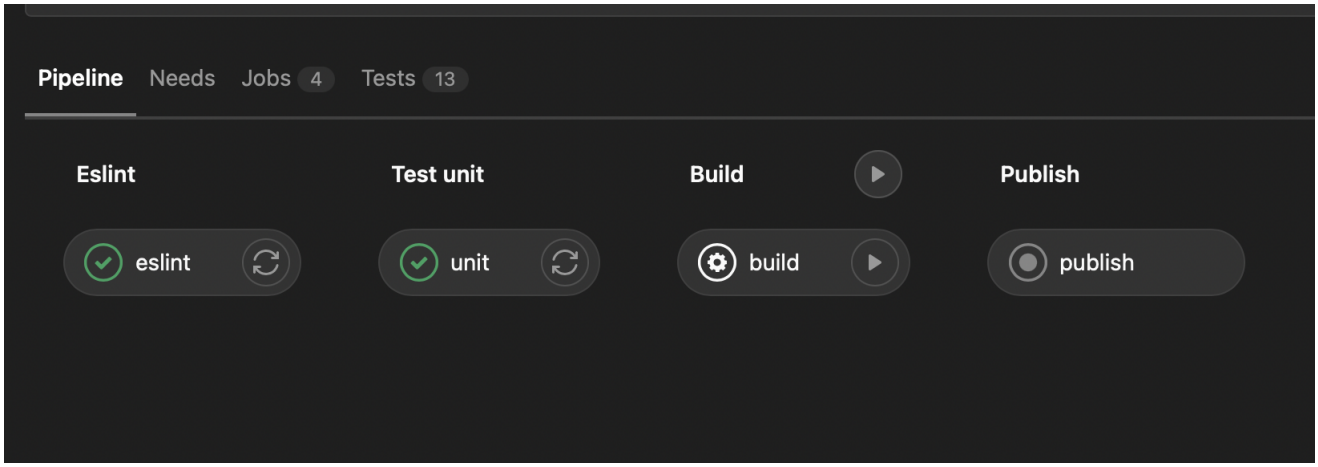

Рисунок 4.5 — Срабатывает автоматизация тестирования, сборка по нажатию пользователя

в) Для проведения данного сценария были рассмотрены следующие случаи: изменение количества сервисов api в окружении release — были составлена линия только из задач тестирования, изменение количества сервисов web-client в окружении release — были составлена линия только из задач тестирования, изменение системных ограничений сервиса api в окружении release — были составлена линия только из задач тестирования.

г)

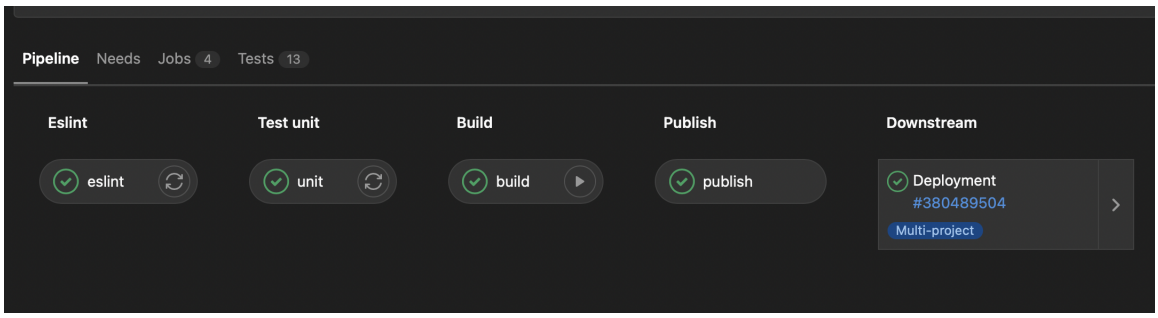

Рисунок 4.6 — Успешное создание релиза

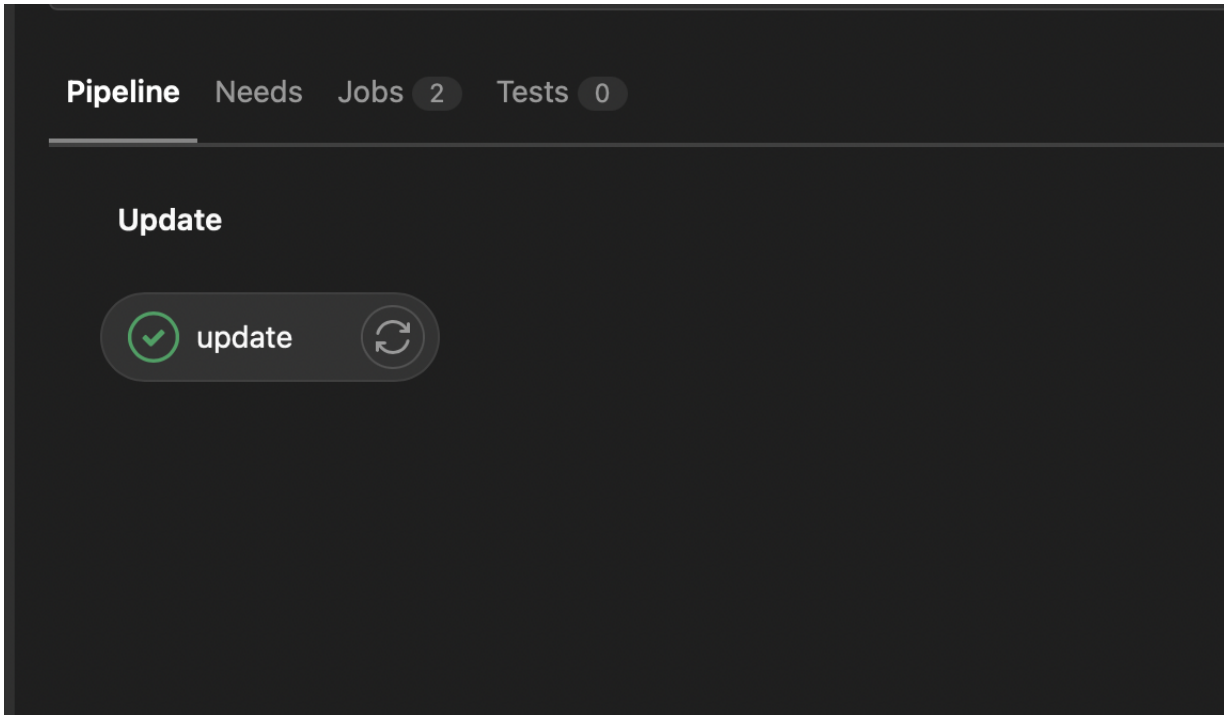

Рисунок 4.7 — Применение конфигураций развёртывания

4.2 Анализ результатов тестирования

Результаты тестирования представлены в виде таблицы:

Таблица 4.1 — Результаты тестирования

№	Название теста	Фактический результат	Ожидаемый результат	Комментарий

1	Линия задач автоматизации тестирования	Линии собираются правильно, задачи выполняются, артефакты предоставляются	Линии собираются правильно, задачи выполняются, артефакты предоставляются	-
2	Линия задач релиза создания релиза	Линии собираются правильно, задачи выполняются, веб-сервиса обновляется	Линии собираются правильно, задачи выполняются, веб-сервиса обновляется	Процесс создание релизов не самый простой
3	Линия задач управления развёрткой	Веб-сервис правильно реагирует на изменение конфигураций развёртывания	Веб-сервис правильно реагирует на изменение конфигураций развёртывания	Требуется несколько минут на применение настроек

Все тесты пройдены успешно согласно плану, несмотря на небольшие проблемы с производительностью.

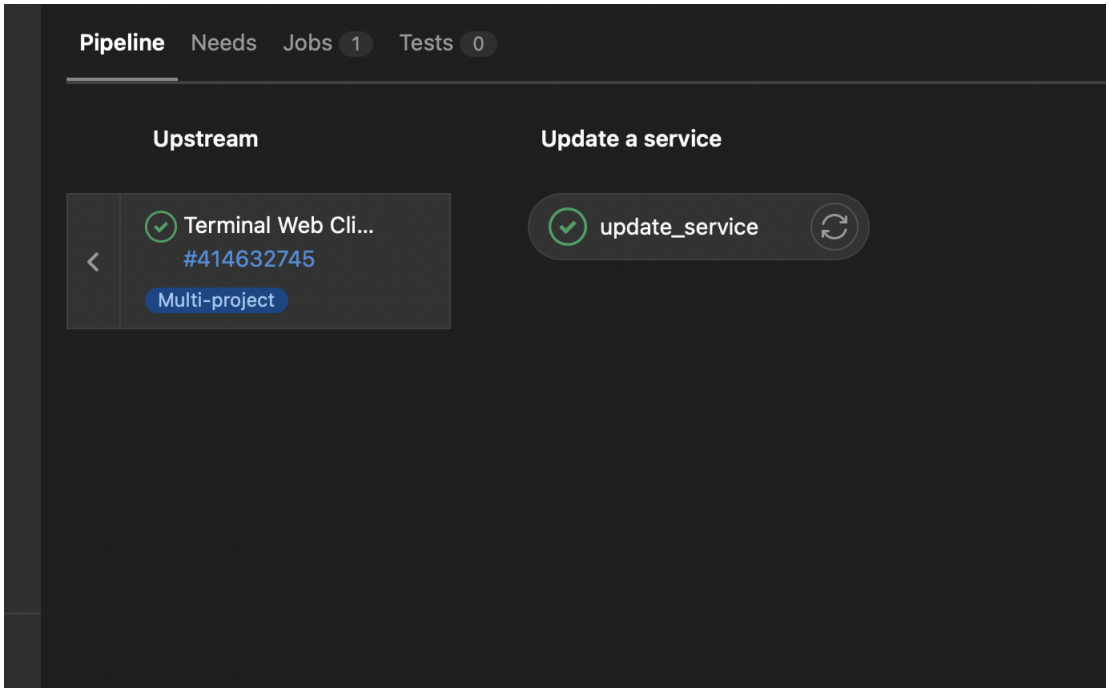

Рисунок 4.8 — Применение конфигураций развёртывания определенного сервиса компонента

ЗАКЛЮЧЕНИЕ

Результаты выполнения задания на выпускную квалификационную работу, - позволяют сделать следующие выводы:

— Цель исследования: развернуть и автоматизировать управление жизненным циклом веб-сервиса на базе Node.js, - полностью реализована; вариант предложенного решения развёртывания позволяет быстро ввести веб-сервис на базе Node.js в рабочее состояние и автоматизировать управление его жизненным циклом по современным методологиям DevOps.

— В рамках решения задачи: Провести обзор необходимых средств для автоматизации управления жизненным циклом веб-сервиса, - были рассмотрены преимущества системы контроля версий Git, проанализированы основные технические параметры и отличия между различными Git хостингами, а так же инструментами оркестрации контейнеров Docker Swarm и Kubernetes .

— В рамках решения задачи: Рассмотреть полученный для развёртывания веб-сервис и проанализировать требования, - была составлена и описана диаграмма развёртывания веб-сервиса, была установлена необходимость хранилища пакетов и образов, были описаны актёры и случаи использования системы в виде соответствующей диаграммы.
Отдельно были описаны рабочие окружения веб-сервиса и на основании необходимых для развёртывания компонентах были составлены виды репозиториев в системе.

— В рамках решения задачи: Провести проектирование механизмов автоматизации управления жизненного цикла веб-сервиса, - были выбраны и автоматизированы два основных вида тестирования веб-сервиса с предоставлением аналитических данных сотрудникам отдела качества, была составлена и задокументирована диаграмма компонентов конфигураций внутри репозиториев системы.
Отдельное внимание было уделено применению модели ветвления Git Flow вместе с инструментами автоматизации CI/CD в целях автоматизи-

рованного создания релизов сервисов и библиотек, а так же нахождению компонентов для обновления при помощи обхода графа зависимостей.
Так же была спроектирована диаграмма компонентов репозиториев для организации хранения исходного кода отдельно от конфигурационных файлов развёртки веб-сервиса.

— В рамках решения задачи: Составить план тестирования механизмов развёртывания веб-сервиса, - были сформулированы цели и задачи тестирования, была выбрана методология тестирования, а так же были описаны тестовые сценарии.

— В рамках решения задачи: Провести практические работы по развёртке и автоматизации управления жизненного цикла веб-сервиса, - в GitLab была создана группа проекта, были созданы необходимые репозитории согласно диаграммам и загружен исходный код веб-сервиса.
На рабочем сервере были открыты TCP и UDP порты, а так же были введены требуемые команды для инициализации кластера Docker Swarm.
Отдельно был зарегистрирован GitLab Runner и сконфигурирован под оптимальную работу с кешированием в системе.

— В рамках решения задачи: Провести тестирование механизмов развёртывания и обосновать полученные результаты, - было произведено ручное тестирование по методологии «Белого ящика» функций полученной системы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Д, Ким.* Проект "Феникс". Как DevOps устраняет хаос и ускоряет развитие компании / Ким Д. — Бомбора, 2020. — P. 384.

2. *Ю.А., Сентерев.* Выпускная квалификационная работа в вопросах и ответах / Сентерев Ю.А. — Университет ИТМО, 2017. — P. 64.

3. Git — Book [Электронный ресурс]. — (дата обращения 25.11.2021). `https://git-scm.com/book/en/v2/`.

4. Рейтинг сервисов-репозиториев для хранения кода 2018 [Электронный ресурс]. — (дата обращения 25.11.2021). `https://tagline.ru/source-code-repository-rating/`.

5. Использование BitBucket [Электронный ресурс]. — (дата обращения 25.11.2021). `https://bitbucket.org/product/ru/guides/`.

6. GitHub [Электронный ресурс]. — (дата обращения 25.11.2021). `https://github.com`.

7. GitLab [Электронный ресурс]. — (дата обращения 25.11.2021). `https://gitlab.com/`.

8. JetBrains Space [Электронный ресурс]. — (дата обращения 25.11.2021). `https://www.jetbrains.com/space/`.

9. *Бейер, Бетси.* Site Reliability Engineering. Надежность и безотказность как в Google / Бетси Бейер. — Питер, 2019. — P. 592.

10. *Ким, Джин.* Ускоряйся! Наука DevOps / Джин Ким. — Интеллектуальная Литература, 2020. — P. 224.

11. Чем полезен Docker Swarm и в каких случаях лучше использовать Kubernetes [Электронный ресурс]. — (дата обращения 25.11.2021). `https://mcs.mail.ru/blog/docker-swarm-ili-kubernetes-chto-luchshe`.

12. *Дэвис Дженнифер, Дэниелс Кэтрин.* Философия DevOps. Искусство управления IT / Дэниелс Кэтрин Дэвис Дженнифер. — Питер, 2017. — P. 416.

13. Swarm mode overview [Электронный ресурс]. — (дата обращения 25.11.2021). `https://docs.docker.com/engine/swarm/`.

14. *Граннеман, Скотт.* Linux. Карманный справочник / Скотт Граннеман. — Вильямс, 2019. — P. 464.

15. Документация GitLab [Электронный ресурс]. — (дата обращения 25.11.2021). `https://docs.gitlab.com`.

16. Документация Docker [Электронный ресурс]. — (дата обращения 25.11.2021). `https://docs.docker.com`.

17. *Джон, Арундел.* Kubernetes для DevOps: развертывание, запуск и масштабирование в облаке / Арундел Джон. — Питер, 2020. — P. 494.