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»

ВВЕДЕНИЕ

Веб технологии широко распространились в нашем мире и на сегодняшний день почти каждая ВС взаимодействует со всемирной паутиной. В свою очередь, поддержка и разработка наиболее популярной архитектуры «Клиент-Сервер» таких систем требует существенных временных затрат, поскольку уже с самого начала проектирования требуется решить ряд следующих задач:

— вертикальное и горизонтальное масштабирование системы,
— доставка обновлений сервиса на рабочие ЭВМ,
— управление окружением ВС,
— осуществления контроля качества поступающих изменений,
— управление версиями ВС,
— бесшовное развёртывание отдельных компонентов системы,
— оперативная загрузка срочных исправлений.

Для решения данных задач в современном мире применяется DevOps методология разработки ПО[3]. Основная идея заключается в предоставлении удобных инструментов связи разработчиков, системных администраторов и не только.

DevOps включает в себя множество различных технологий, среди которых можно выделить SaaS — это облачное решение при использовании [8] которого, пользователь получает доступ к сервису, как правило, через браузер или по API; IaC — это процесс управления и провизионирования датацентров и серверов с помощью машиночитаемых файлов определений, созданный как альтернатива физическому конфигурированию оборудования и оперируемым человеком инструментам; а так же CI/CD — это методология обеспечения оперативности вывода новой функциональности продукта и повышение качества разрабатываемого решения[9].

Теперь, вместо того, чтобы запускать сотню различных файлов конфигурации, достаточно запускать скрипт, который управляет инфраструктурой и масштабированием системы.

Несмотря на это, для ввода в рабочий процесс всё равно необходима дополнительная разработка механизмов развёртки, начиная от описания файлов конфигураций до разработки дополнительных порграммных средств.

В зависимости от решаемых задач и условий эксплуатации разработчиками выбираются конкретные DevOps инструменты.

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 методолгии SaaS,

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

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

— наличие конфигурации прав доступа (приватизации исходного кода),

— наличие хранилища пакетов, образов и контейнеров,

— стоимость установки и обслуживания системы — не более 12 500 рублей за установку и не более 5 000 рублей в месяц за обслуживание на момент написания данной работы.

Поставленные требования определяют предмет исследования:

Простое и доступное в установке и поддержке ПО для автоматизации управления жизненным циклом веб-сервиса[1].

Объект исследования: ПО, 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 три главы, заключения и приложений. В ВКР содержится 10 страниц основного текста, 0 рисунков, 1 таблица и 0 приложений.

1 ОБЗОР СУЩЕСТВУЮЩИХ СИСТЕМ АВТОМАТИЗАЦИИ УПРАВЛЕНИЯ ЖИЗНЕННЫМ ЦИКЛОМ ВЕБ-СЕРВИСА

1.1 Система контроля версий

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 развёртки программного обеспечения, стоит начать с такого ключевого элемента, как VCS.

Наиболее популярной системой контроля версий на момент написания работы является git[16]. 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 лидерующих представителей, отличающихся в основном стоимостью и набором дополнительных инструментов[18].

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 развёртки проектов по методологии CI/CD, GitHub предоставляет инструмент Actions. Хранилище образов и пакетов предоставляется с ограничением и только проектам с открытм 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[17] 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 образов и пакетов отсутствует. Bitbucket следует рассматривать, как альтернативу GitHub только для коммерческой разработки[14].

Одним из наиболее подходящих хостингов является GitLab от украинских разработчик (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 реализовано большинство основных 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[19].

Так же был рассмотрен продукт от разработчиков из Санкт-Петербурга (международной компании JetBrains) — Space[15]. 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.

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 [5].

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[7].

Альтернативным решением является Kubernetes — это универсальное средство для создания распределенных систем. Это комплексная система с большим количеством возможностей, разработка и поддержка которой самостоятельно значительно трудозатратна. 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[20]. Гибкость данного решения излешне 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 механизны части 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

Так как полученный для равёртки веб-сервисе базируется на Node.js и исходный код разбит на библиотеки, то то для корректной работы системы потребуется наличие регистров Node пакетов. Помимо регистра пакетов необходимым будет регистр Docker образов, поскольку веб-сервис уже представлен в виде Docker образов и имеет все необходимые образы ПО необходимые окружению.

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

— администратор — пользователь занимающийся настройкой прав доступа другим пользователям и конфигурацией развёртки 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 аналитеческих 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 2.1.

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

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

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

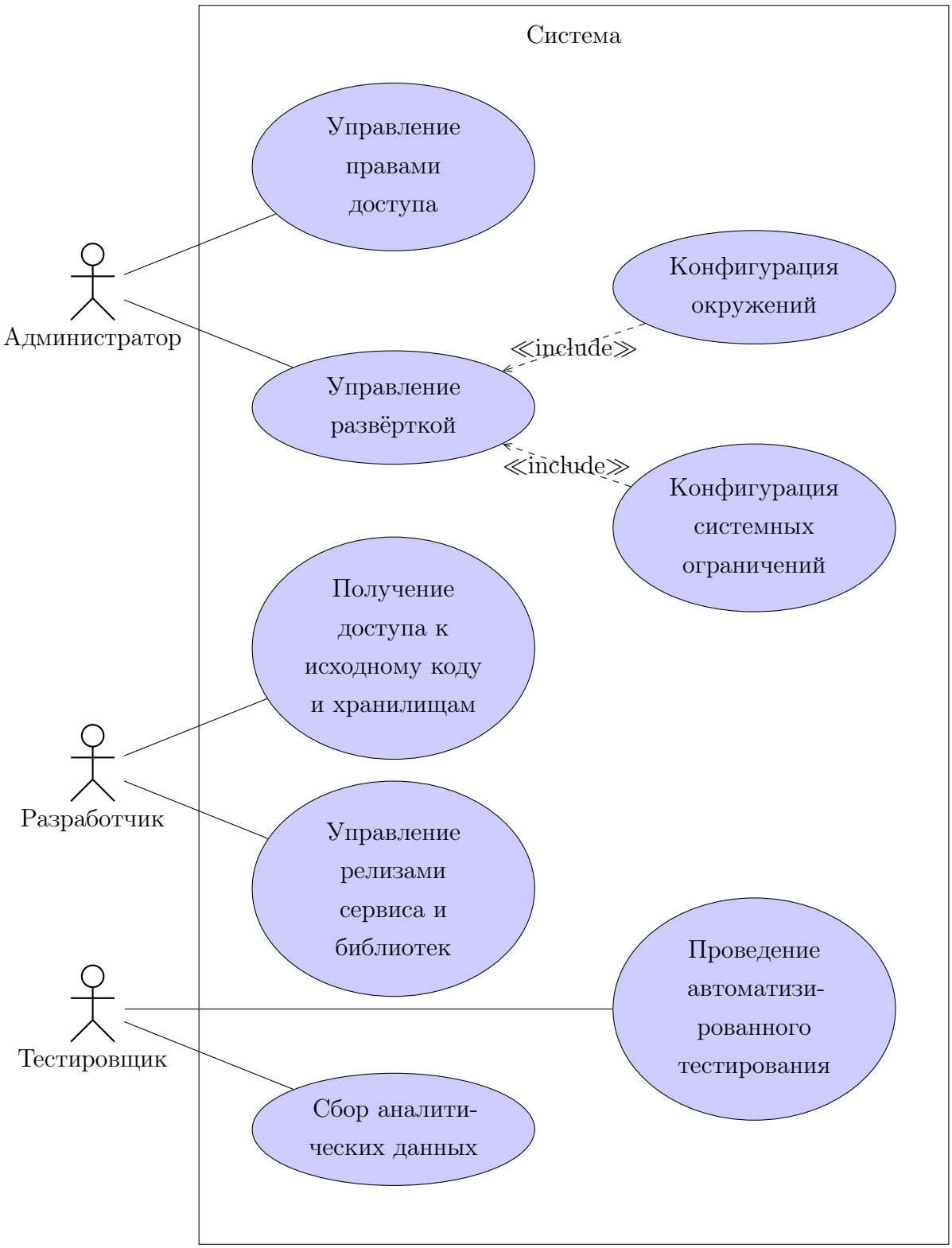

Рисунок 2.1 — Диаграмма случаев использования системы

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

С точки зрения CI/CD взаимодействие пользователя с системой сосредоточено вокруг комита в репозиторий и автоматическим запус-

ком 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 действию пользователя. При этом линии задач строятся динамически в зависимости от конкретного репозитория и ветки, поэтому некоторые задачи могут быть пропущены при выполнении, а некоторые комиты оставаться без линий задач вовсе. Результатами работы линий являются артефакты, которые содержар основную информацию о результатах работы системы. В общем виде поведение системы отражено на Рисунке 2.2.

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

— репозиторий с исходным кодом компонента веб-сервиса — может быть несколько, обязательно содержит Dockerfile в корне, предоставляется полный доступ разработчикам, доступ к аналитическим данным тестировщику.

— репозиторий с конфигурациями развёртки — только один, содержит общие скрипты и конфигурации, предоставляется доступ только администратору,

— репозиторий с исходным кодом библиотек компонентов веб-сервиса — только один, предоставляется полный доступ разработчикам. доступ к аналитическим данным тестировщику.

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 был отдельно разобран каждый случай использования и описан в виде линии задач:

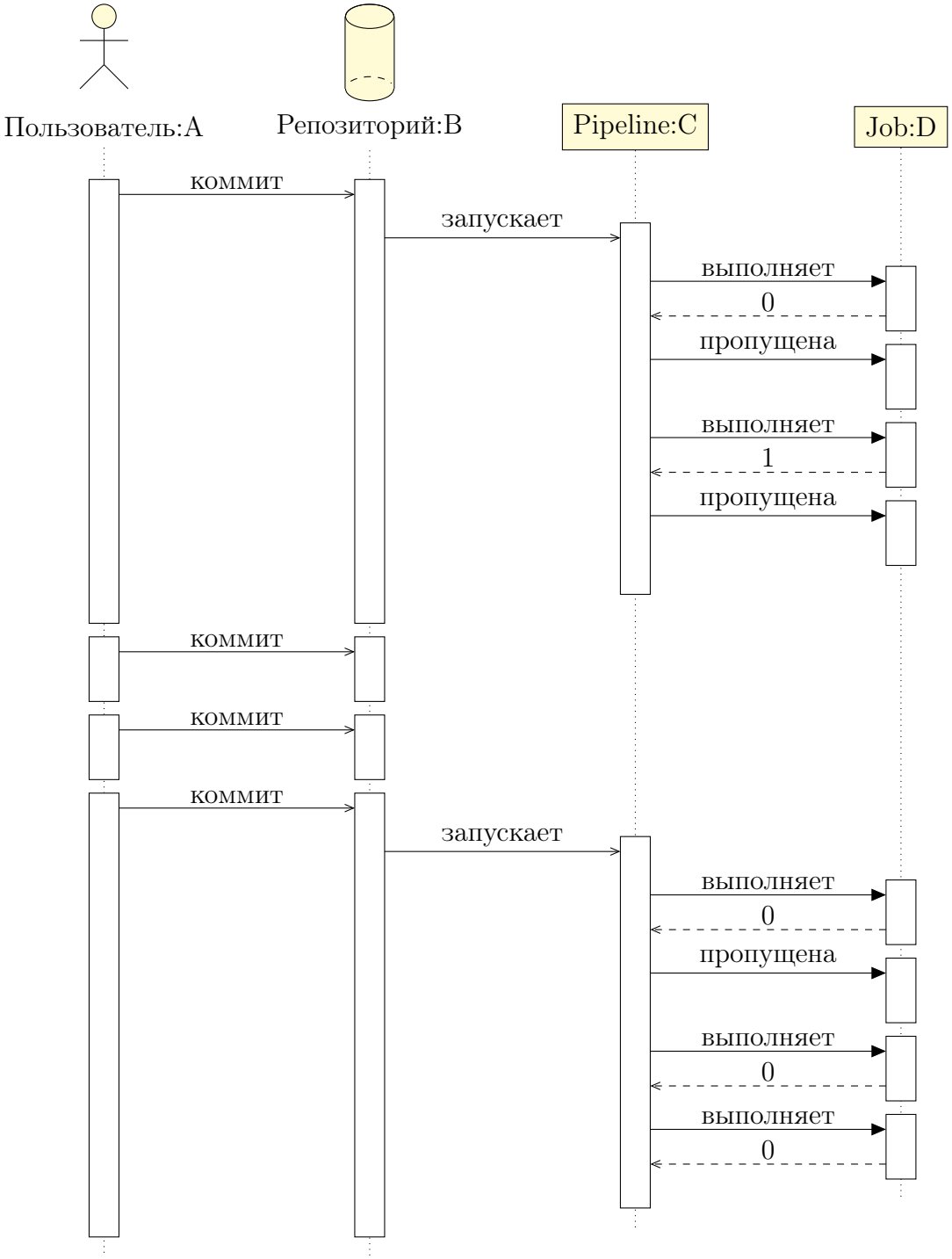

Рисунок 2.2 — Общий случай работы системы

— 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, за-

гружать 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 резульате 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 о их содержании переносится на тестировщика. В целом, процесс тестирования будет происходить в несколько основных этапов: семантическое тестирования 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 тестирования 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 коммита 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.

Данная линия будет иметь два аргумента: название сервиса компонента, которые необходимо обновить, и 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 развёртки. Разработчику необходимо будет только импортировать необходимые задачи из репозитория.

Конфигурация развёртки веб-сервиса состоит из линии, включающей только одну задачу: применение конфигураций развёртки 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 конкурентов. 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 ор-

кестратора вместо гибкости Kubernetes был выбран Docker Swarm[4]. 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 мэнеджер, поскольку более не требуется на данном этапе.

Одной и ключевой настройкой является открытие портов на уровне операционной системы сервера[12]:

— 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 развёртки 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 ранера можно следующим образом: раннер запускается в отдельном контейнере с добавленным volume на сокет Docker, таким образом получается избежать достаточно сложного и нелесообразного запуска Docker внутри Docker, так как в этом случае раннер 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 ранера, необходимо будет произвести настройку кэша ранера. Ключевой настройкой является политика загрузки образов для задач, поскольку по умолчанию ранер в любом случае будет загружать образ из регистра, даже если образ представлен локально. Согласно требованиям ранер должен будет запускать минимум три задачи за единицу времени, данное значение будет отражено в конфигурации на этапе реализации.

2.4 Составление плана тестирования

В качестве метода тестирования было выбрано тестирования белого ящика, так как исходный код системы известен.

На основании диаграммы случаев использования были описаны следующие основные тестовые сценарии и представлены в виде таблицы 2.1.

Таблица 2.1 — 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 развёртки, изменить конфигурации развёртки, зафиксировать изменения коммитом	Конфигурации развёртки веб-сервиса	Веб-сервис правильно реагирует на изменение конфигураций развёртки

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, установле SSH ключ и создана группа проекта. Результаты создания группы представлены на рисунке 3.1.

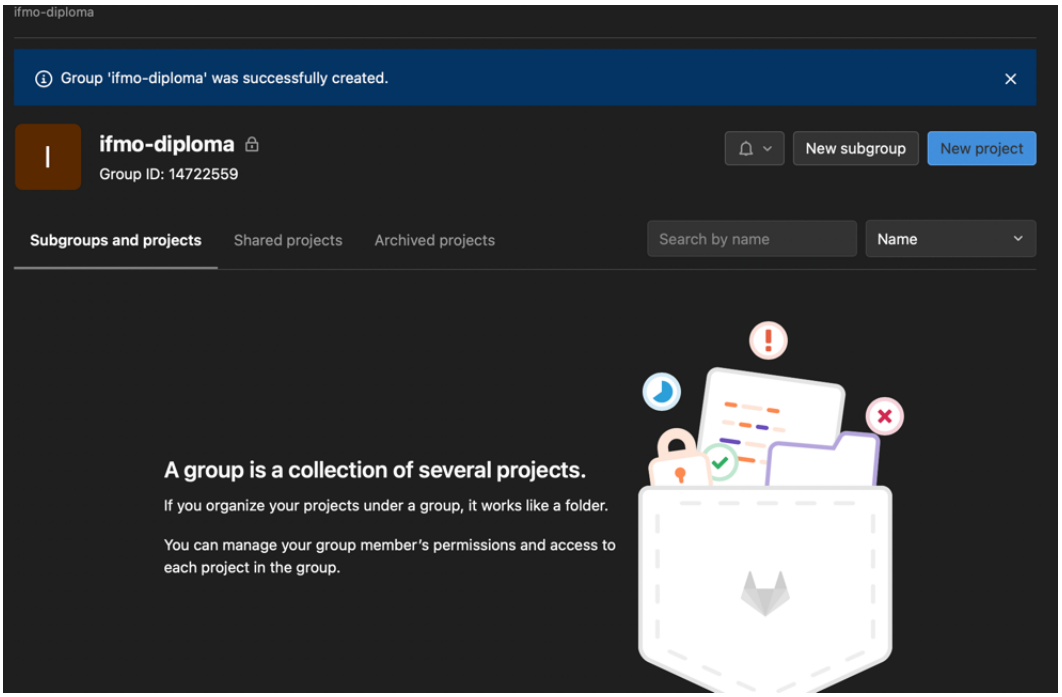

Рисунок 3.1 — Скриншот создания группы в GitLab

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 развёртки данного сервиса. Так как в качестве модели ветвления была выбрана модель git flow, то так же были подговлены соответствующие ветки в репозиториях под разные окружения: develop, testing и release. Установка доступа к данным репозиториям предоставляется только соответсвующим разработчикам данных программных решений в целях безопасности.

Подготвка 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 развёртки. Установка доступа к этому репозиторию предоставляется только команде обеспечения развёртки в целях безопасности. Результаты создания репозиториев представлены на рисунке 3.2.

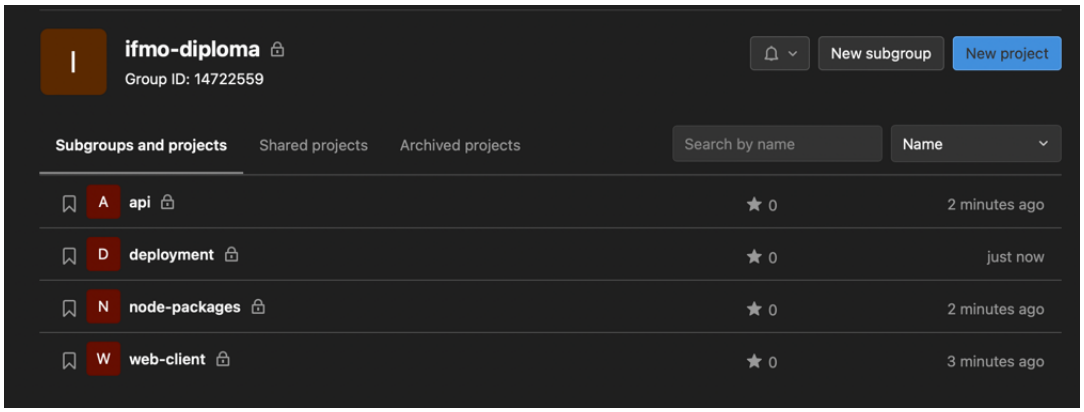

Рисунок 3.2 — Скриншот создания репозиториев в GitLab

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[2]:

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 comman:
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[11] файла в соответствии с этапом проектирования, основными настройками являются являются:

— Установка executor — docker executor,
— Установка volumes — /var/run/docker.sock[10],
— 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 о об обонвлении сервиса для загрузки новой версии,

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

— build-package.yaml — набор задач для для отслеживания к какому конкретному сервису принадлежит npm пакет путём построения графа зависимостей на основании package.json файлов,

— publish-package.yaml — набор задач для загрузки собранного npm пакета .

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

Аналогичные задачи были описаны для репозитория node packages.

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 окружния (develop, testing и release), содержащие аргументы сервисов компонентов системы (доступы к базе данных, секрет ключа авторизации и так далее)[6]. Аналогично были добавлены Docker Swarm конфигурационные файлы под каждый окружения:

— develop — каждый сервис запускается на сервере в одном экзмепляре, 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 экзмепляре, база данных и api запускаются в трёх, большая часть ресурса отведена под эти сервисы.

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).

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 4.1, а пример пройденных автоматических тестов с генерированным файлами артифектов на Рисунке 4.2.

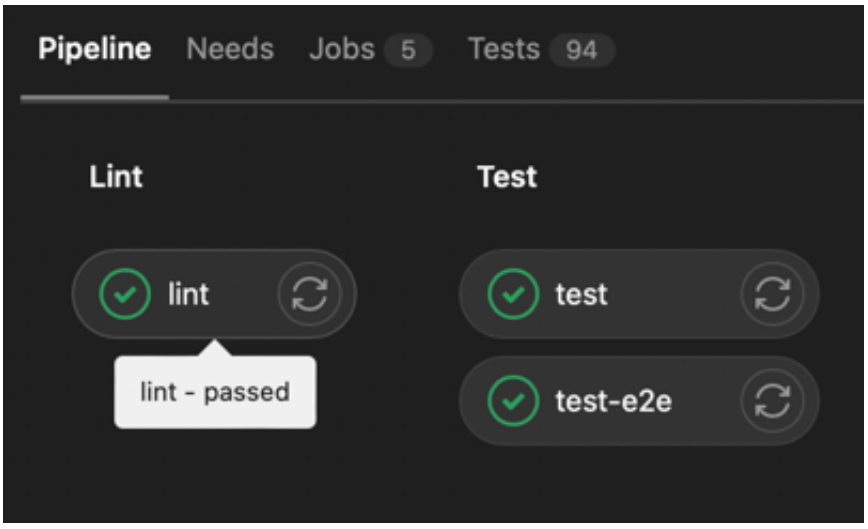

Рисунок 4.1 — Линия тестирования в GitLab

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

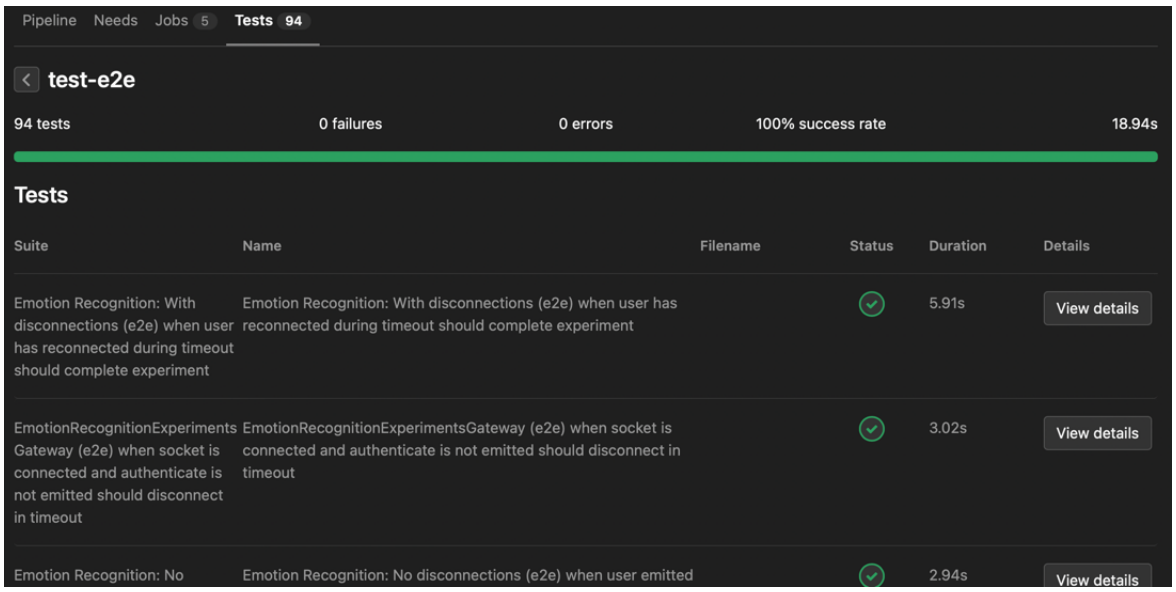

Рисунок 4.2 — Артефакты тестирования в GitLab

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:

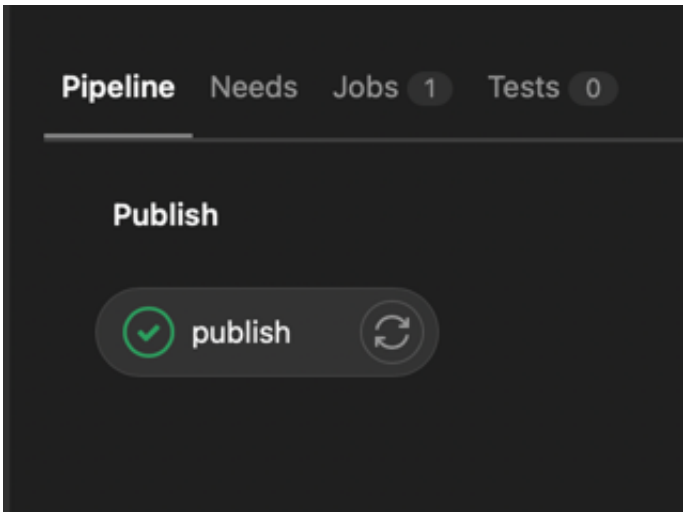

Рисунок 4.3 — Загрузка библиотеки в хранилище

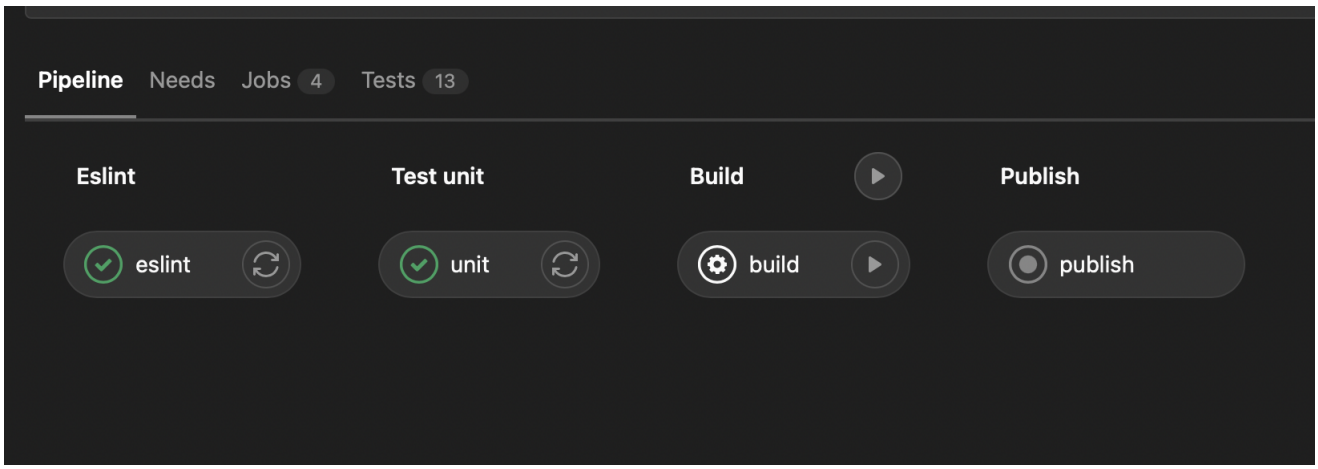

Рисунок 4.4 — Срабатывает автоматизация тестирования, сборка по нажатию пользователя

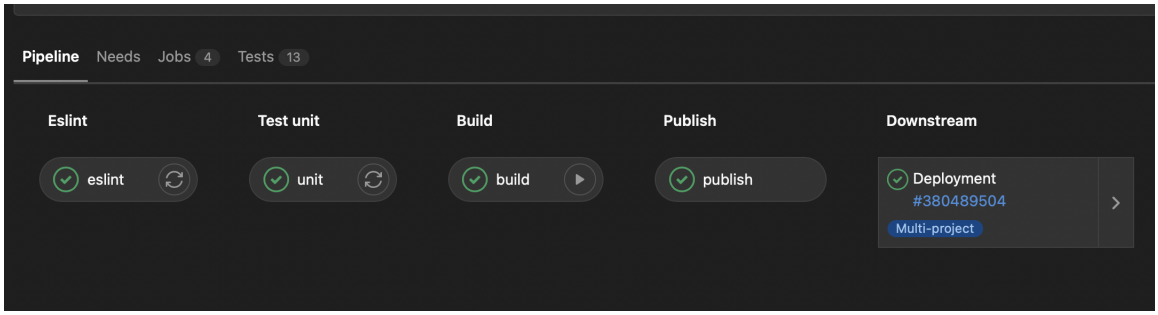

Рисунок 4.5 — Успешное создание релиза

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)

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 развёртки	Веб-сервис правильно реагирует на изменение конфигураций развёртки	Требуется несколько минут на применение настроек

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

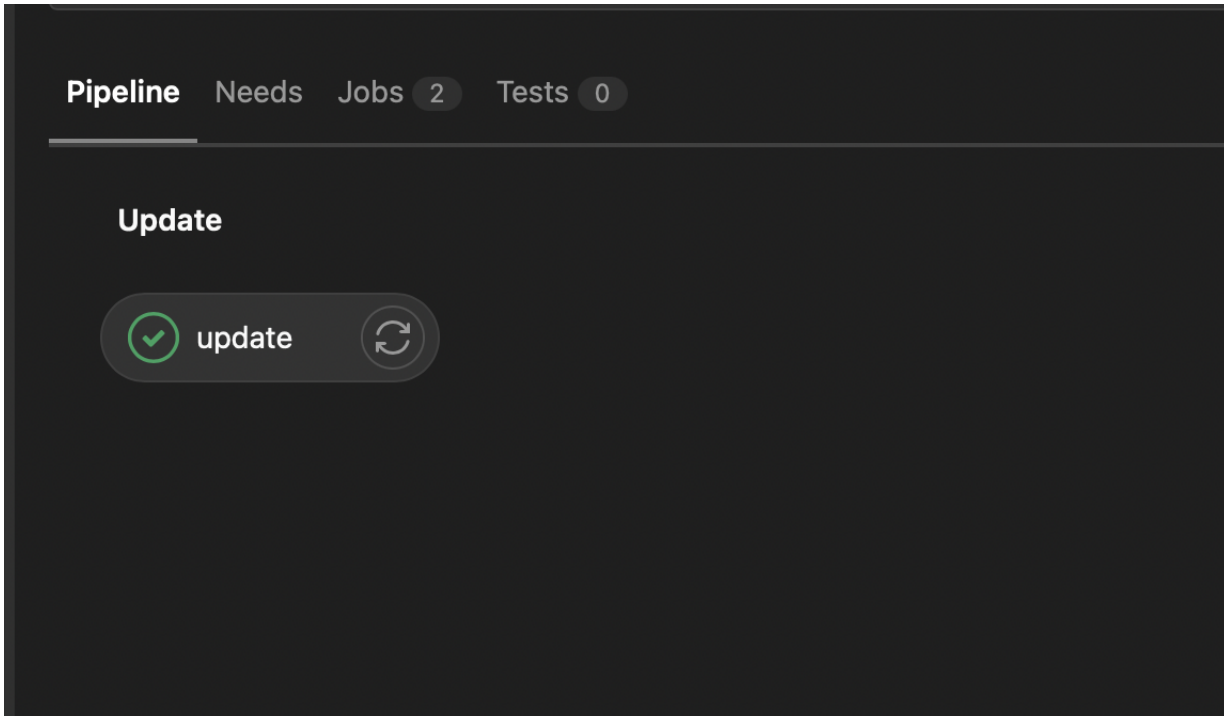

Рисунок 4.6 — Применение конфигураций развёртки

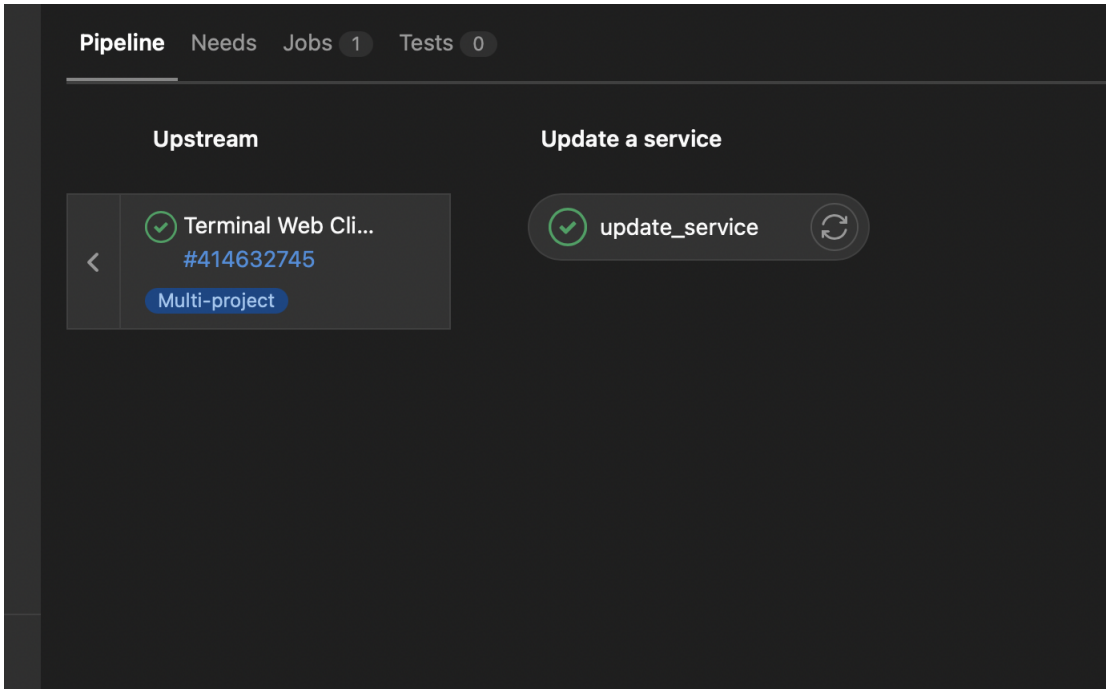

Рисунок 4.7 — Применение конфигураций развёртки определенного сервиса компонента

ЗАКЛЮЧЕНИЕ

В ходе проделанной работы была разработана система автоматизации управления жизненным циклом веб-сервиса. Разработанная система отвечает всем требованиям, установленным в техническом задании.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

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

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

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

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

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

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

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

8. *Эберхард, Вольф.* Continuous delivery. Практика непрерывных апдейтов / Вольф Эберхард. — Питер, 2018. — P. 320.

9. *Джез, Хамбл.* Непрерывное развертывание ПО. Автоматизация процессов сборки, тестирования и внедрения новых версий программ / Хамбл Джез. — Вильямс, 2016. — P. 432.

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

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

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

13. Документация npm [Электронный ресурс]. — (дата обращения 25.11.2021). `https://docs.npmjs.com`.

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

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

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

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

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

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

20. 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.