Содержание

Глоссарий

Вычислительная система — Совокупность аппаратно-программных средств, образующих единую среду, предназначенную для решения задач обработки информации (вычислений).

Обозначения и сокращения

ЭВМ — Электронно-вычислительная машина

ВС — Вычислительная система

ПО — Программное обеспечение

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.

Подавляющее большинство этих задач решается при помощи методологии SaaS (Software as a Service, «Программное обеспечение как услуга»). Основная идея заключается в предоставлении программного обеспечения по подписке. SaaS — это облачное решение при использовании которого, пользователь получает доступ к сервису, как правило, через браузер или по API. Оплачивает доступ и максимально быстро получает на руки готовый инструмент.

В данной работе сделан акцент на примение готовых SaaS решений, поскольку в условиях современной коммерческой разработки игнорирование таких технологий по большей части нецелесообразно и влечёт серьёзные временные, и как следствие, денежные потери.

Помиомо SaaS так же используется методология IaC (Infrastructure as Code, «Инфраструктура как код») — это процесс управления и провизионирования датацентров и серверов с помощью машиночитаемых файлов определений, созданный как альтернатива физическому конфигурированию оборудования и оперируемым человеком инструментам. Теперь, вместо того, чтобы запускать сотню различных

файлов конфигурации, IaC позволяет просто запускать скрипт, который управляет инфраструктурой и масштабированием системы.

Большинство таких средств исполльзует в качестве основы Docker, который предоставляет инструментарий для контейнерезации программного обеспечения.

Так как в полученном для равёртки сервисе уже используется Node.js и Docker, то для корректной работы разрабатываемого комплекса требуется наличие регистров Node пакетов и Docker образов.

Так же в наши дни крайне распространена методология разработки CI/CD (Continuous Integration/Continuous Deployment, «Продолжительная интеграция/Продолжительная развёртка»). Она представляет собой автоматизацию тестирования и доставки новых проектов разрабатываемого проекта разработчикам, аналитикам, инженерам качества, конечным пользователям и другим заинтересованным сторонам. Метод обеспечивает оперативность вывода новой функциональности продукта и повышение качества разрабатываемого решения. Методология предоставляет интерфейс конвееров (Pipelines), состоящий из последовательно выполняемых задач (Jobs).

В зависимости от решаемых задач и условий эксплуатации разработчиками выбираются конкретные инструменты из области рассмотренных выше методологий.

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

— 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 NPM пакетов и регистра Docker образов,

— 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 задачи определяют предмет исследования:

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

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

При проведении исследований были просмотрены 19 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, 0 таблиц и 0 приложений.

1 ОБЗОР СУЩЕСТВУЮЩИХ СИСТЕМ ПО 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 системы контроля версий (или VCS — Version Control System). Так как в мире современных технологий большая часть практических задач решается именно такими системами.

Наиболее популярной системой контроля версий на момент написания работы является git. 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 спосо-

бы 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.

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. С его помощью автоматизация процессов развёртки и тестирования производится в декларативном стиле путём описания YAML файлов. 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 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.

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 ре-

альном времени, параллельное выполнение, локальный запуск скриптов, поддержку Docker. Несмотря на это, для ввода его в рабочий процесс всё равно необходима дополнительная разработка механизмов развёртки, начиная от описания YAML файлов конфигураций до разработки дополнительных порграммных средств.

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

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.

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.

Альтернативным решением является Kubernetes — это универсальное средство для создания распределенных систем. Это комплексная система с большим количеством возможностей, разработка и под-

держка 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. 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.

[1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19]

2 Конструкторский раздел

В данном разделе проектируется новая всячина.

2.1 Архитектура всячины

Проверка параграфа. Вроде работает.

Вторая проверка параграфа. Опять работает.
Вот.

— Это список с «палочками».
— Хотя он и не по ГОСТ, кажется.

а) Поэтому для списка, начинающегося с заглавной буквы, лучше список с цифрами.

Формула 2.1 совершено бессмысленна.

$$a = cb \tag{2.1}$$

Окружение `cases` опять работает (см. 2.2), спасибо И. Короткову за исправления..

$$a = \begin{cases} 3x + 5y + z, \text{если хорошо} \\ 7x - 2y + 4z, \text{если плохо} \\ -6x + 3y + 2z, \text{если совсем плохо} \end{cases} \tag{2.2}$$

2.2 Подсистема всякой ерунды

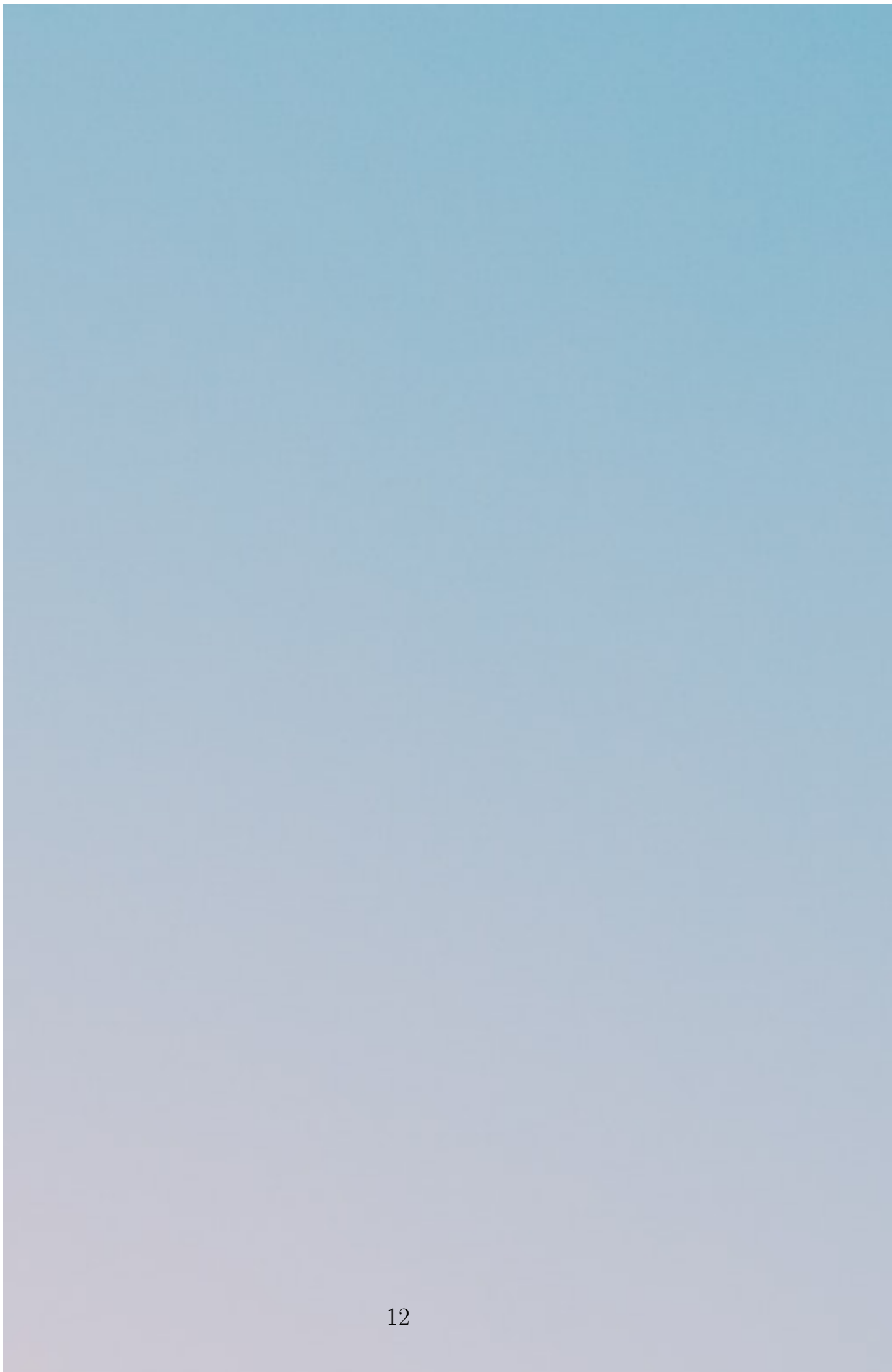
Культурная вставка dot-файлов через утилиту dot2tex (рис. 2.1).

2.2.1 Блок-схема всякой ерунды

Кстати о заголовках

У нас есть и **subsubsection**. Только лучше её не нумеровать.

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 оркестратора вместо гибкости Kubernetes был выбран Docker Swarm. Но так как в данной работе делаеся 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.

Как было сказано ранее, для развёртки в работе используется Docker, поэтому необходим простой инструмент удлаенного 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 локаль-

но. Согласно требованиям ранер должен будет запускать минимум три задачи за единицу времени, данное значение будет отражено в конфигурации на этапе реализации.

Автоматизация контроля качества будет реализована путём запуска описанных разработчиками тестовых сценариев внутри Pipelines. Отчёты по прохождению сценариев будут представлены в веб интерфейсе GitLab путём интеграции с CI/CD.

Для контроля версий будет использована модель ветвления git flow со следующими окружениями:

— develop — инсценировка рабочего окружения веб сервиса для разработчиков,

— testing — окружение для проведения ручного тестирования,

— release — рабочее окружение сервиса для реальных пользователей.

Основная проблема заключается в обновлении пакетов зависимостей проекта, поскольку для обновления необходимо пройти засимимым сервисам и построить граф. Для решения этой проблемы будет разработано программное решение на базе CI/CD с применением bash скриптов при помощи package.json файлов веб сервиса.

Для проведения тесирования был составлен план содержащий набор ключевых функция для тестирвоания. Подробнее план представлен в виде таблице с описанием сценариев, входных данных и ожидаемого результата в тестке работы ВКР.

Рисунок 2.2 — UML диаграмма

3 Технологический раздел

В данном разделе описано изготовление и требование всячины. Кстати, в Latex нужно эскейпить подчёркивание (писать «`some\_function`» для **some_function**).

Для вставки кода есть пакет **listings**. К сожалению, пакет **listings** всё ещё работает криво при появлении в листинге русских букв и кодировке исходников utf-8. В данном примере он (увы) на лету конвертируется в koi-8 в ходе сборки pdf.

Есть альтернатива **listingsutf8**, однако она работает лишь с **lstinputlisting**, но не с окружением **lstlisting**

Вот так можно вставлять псевдокод (питоноподобный язык определен в **listings.inc.tex**):

Листинг 3.1 — Алгоритм оценки дипломных работ

```
1 def EvaluateDiplomas():
2     for each student in Masters:
3         student.Mark ← 5
4     for each student in Engineers:
5         if Good(student):
6             student.Mark ← 5
7         else:
8             student.Mark ← 4
```

Еще в шаблоне определен псевдоязык для BNF:

Листинг 3.2 — Грамматика

```
1  ifstmt → "if" "(" expression ")" stmt  |
2           "if" "(" expression ")" stmt1 "else" stmt2
3  number → digit digit *
```

В листинге 3.3 работают русские буквы. Сильная магия. Однако, работает только во включаемых файлах, прямо в TEX нельзя.

Листинг 3.3 — Пример (**test.c**)

Можно также использовать окружение **verbatim**, если **listings** чем-то не устраивает. Только следует помнить, что табы в нём «съедаются». Существует так же команда **verbatiminput** для вставки файла.

```
a_b = a + b; // русский комментарий
if (a_b > 0)
    a_b = 0;
```

Перед реализацией необходимо произвести базовую настройку окружения. Для этого был зарегистрирован GitLab аккаунт, установле SSH ключ и создана группа проекта.

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.

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.

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

Базовая подготовка окружения завершена. Следующим этапом был установлен и настроен Docker Swarm кластер. Одной и ключевой настройкой является открытие портов на уровне операционной системы сервера:

— 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.

Дальше была произведена инициализация кластера на сервера. В данном этапе нет необходимости при использовании Kubernetes.

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 файла в соответствии с этапом проектирования, основными настройками являются являются:

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

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

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, содержащего следующие задачи:

— lint — осуществление контроля качества изменений путём проверки семантических правил написания кода,

— test — осуществление контроля качества путёт запуска описанных разработчиками тестов кода,

— build — импортированная из репозитория common задача для сборки Docker образа и загрузки в регистр,

— publish — импортированная из репозитория common задача для оповещения кластера о загрузке новой версии (опциональная задача, требует подтверждения пользователем).

Аналогичные задачи были описаны для репозитория node packages.

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). 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.

Так же был подготовлен пример описания аналогичных файлов при использовании с Kubernetes.

Основная часть работы сделана полном объёме, дальше было произведено ручное тестирование функций программного комплекса согласно плану тестирования, составленному на этапе проектирования. В ходе тестирования все компоненты системы работали исправно и тесты ошибок не выявили.

Итоговая стоимость разработки с учётом только стоимости использования сервисов и обсулживания составляет 0 рублей за установку и 0 рублей в месяц за обслуживание.

Заключение

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

19. 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.