Введение

В современном мире стремление к автоматизации – естественный процесс. Сама по себе, автоматизация – это передача техническим средствам функций управления и контроля, ранее выполнявшихся человеком вручную.

В IT индустрии актуальность применения автоматизированных средств для разработки программных продуктов обусловлена тем, что оперативная подготовка программного продукта к релизу — ключевое конкурентное преимущество во многих отраслях бизнеса. Поэтому в основе быстро растущих и гибко реагирующих на любые изменения современных компаний лежит большое разнообразие инженерных культур и практик. Одной из таких практик является CI/CD, которой занимаются DevOps-инженеры.

В [разработке ПО](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%9F%D0%9E), CI/CD или CICD — это комбинация [непрерывной интеграции](https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%BF%D1%80%D0%B5%D1%80%D1%8B%D0%B2%D0%BD%D0%B0%D1%8F_%D0%B8%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B0%D1%86%D0%B8%D1%8F) (continuous integration) и [непрерывной доставки и развертывания](https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%BF%D1%80%D0%B5%D1%80%D1%8B%D0%B2%D0%BD%D0%B0%D1%8F_%D0%B4%D0%BE%D1%81%D1%82%D0%B0%D0%B2%D0%BA%D0%B0) (continuous delivery или continuous deployment) программного обеспечения в процессе разработки.

Сам по себе перевод исходного кода в рабочее состояние на конкретном сервере (другими словами, деплой) — рутинная задача. Для этого необходимо проверить и собрать код, провести тестирование, перенести все файлы на сервер, выполнить миграции, поправить конфигурации и др. На выполнение этих задач вручную уходит значительное количество времени. В тоже время высока вероятность упустить мелкую деталь, которая в итоге сломает конвейер. Таким образом, на подготовку кода к релизу уйдут месяцы, что недопустимо в условиях постоянной конкуренции.

Прежде чем новый программный продукт появится на рынке, программисты вынуждены преодолевать большое количество проблем, которые могут внезапно возникнуть на любом этапе разработки.

Наиболее часто возникающие проблемы в процессе разработки:

— **нехватка времени на разработку нового функционала;**

— низкое качество кода за счет необходимости выполнить работу в срок;

— большое количество ошибок в коде, приводящие к **тратам времени на** поиск их причин и исправление;

— проблемы с интеграцией, исправление которых обходится дорого;

— постоянная необходимость в наличии текущей стабильной версии вместе с продуктами сборок — для тестирования, демонстрации;

— **разобщённость между разработкой и операционной деятельностью.**

Решая перечисленные проблемы, CI/CD нормализует работу и коммуникацию бизнеса с IT — исчезают противоречия, а процесс построения CI/CD конвейера делает процесс разработки и выпуска ПО более компактным и эффективным. Пока компьютеры решают однообразные задачи, разработчики могут сосредоточиться на решении проблем бизнеса и проведении экспериментов.

Исходя из выше написанного были выделены цель и задачи.

Целью работы является автоматизация CI/CD процесса на выделенные сервера.

Задачи исследования:

— провести анализ предметной области;

— провести обзор существующих программных решений и выявить их преимущества и недостатки;

— определить требования к разрабатываемому программному продукту;

— разработка приложения;

— тестирование разработанного программного продукта.

Объект исследования — непрерывная интеграция, доставка и развертывание программного обеспечения.

Предмет исследования — исследование методов автоматизации процесса continuous integration, continuous delivery, continuous deployment.

Методы исследований. Для решения поставленных задач были использованы методы математического моделирования, системного анализа, программной инженерии.

Практическая значимость работы заключается в том, что разработанное программное обеспечение будет помогать разработчикам ускорять разработку программных средств и предоставлять клиенту необходимый функционал.

Глава 1

* 1. Характеристики проблемной области

CI/CD — одна из популярных методологий разработки. Впервые идея ее внедрения [была озвучена](https://martinfowler.com/articles/continuousIntegration.html) в 2006 году, а уже в 2008 году специалисты [выразили мнение](https://circleci.com/blog/a-brief-history-of-devops-part-iii-automated-testing-and-continuous-integration/), что ее широкое распространение связано с развитием облачных сервисов. При этом стремление применить ее для решения других задач обусловлено не популярностью, а преимуществами системы — возможностью быстро согласовывать и внедрять обновления.

Эта методология становится необходимостью в условиях жесткой конкуренции. Для ее внедрения этой методологии нужно расставить бизнес-приоритеты, иметь четкий план действий, согласованные технологии и команду опытных DevOps-специалистов. CI/CD подойдет для задач, касающихся веб-разработки, омниканальных решений, e-commerce и прочего комплекса frontend- и middleware-компонентов. Однако внедрение CICD оправдано не всегда — например, в сферах с редким обновлением софта методология себя не оправдает.

Таблица 1 — Достоинства и недостатки применения методологии CICD.

|  |  |
| --- | --- |
| Достоинства | Недостатки |
| Методология уменьшает время запуска обновлений до нескольких дней (в отдельных случаях, недель). Благодаря этому, разработчики получают возможность быстрее опробовать нововведения и внедрять решения быстрее конкурентов. | Требования к опыту. На практике для получения результата нужен первичный опыт работы с методологией, а также правильная организация перестраивания всех процессов. |
| Оперативное тестирование и много итераций помогают разработчику быстро выявлять варианты, не имеющие перспектив, еще на начальных этапах. | Непрерывное обновление и непрерывная поставка должны быть четко скоординированы, что возможно только после тщательной настройки взаимодействия между специалистами всех уровней. |
| Проведение автоматического тестирования помогает повысить качество результата выявить ошибки и другие проблемы на самых ранних этапах разработки. |  |

* 1. Описание существующих способов/процессов решения задачи (рекомендуется оформлять процессы решения задачи в виде диаграмм IDEF0 стандарта SADT или диаграмм деятельности стандарта UML 2.0), их критические места, достоинства и недостатки;

Для применения методологии необходимо выполнение ряда базовых требований к разрабатываемому проекту. Исходный код и всё, что необходимо для сборки и тестирования проекта, должно храниться в репозитории [системы управления версиями](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%8F%D0%BC%D0%B8), а операции копирования из репозитория, сборки и тестирования всего проекта должны быть автоматизированы и легко вызываться из внешних программ.

Для организации процесса непрерывной интеграции на выделенном сервере запускается служба, в задачи которой входят:

— получение исходного кода из [репозитория](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BF%D0%BE%D0%B7%D0%B8%D1%82%D0%BE%D1%80%D0%B8%D0%B9);

— сборка проекта;

— выполнение тестов;

— развёртывание готового проекта;

— отправка отчетов.

* 1. Результаты анализа существующих способов/процессов решения задачи

1.4 Выбор перспективных подходов к решению задачи;

1.5 Необходимые теоретические сведения, описания математических и иных моделей, используемых в перспективных подходах

К основным теоретическим понятиям можно отнести такие термины как система контроля версий, исходный код, репозиторий, сборка, интеграция, доставка и развертывание программного обеспечения.

Система контроля версий — [программное обеспечение](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) для облегчения работы с изменяющейся информацией. Система управления версиями позволяет фиксировать изменения документа (создавая коммиты), хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

Такие системы наиболее широко используются при [разработке программного обеспечения](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) для хранения [исходных кодов](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) разрабатываемой программы.

Исходный код — текст [компьютерной программы](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0) на каком-либо [языке программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) или [языке разметки](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D1%80%D0%B0%D0%B7%D0%BC%D0%B5%D1%82%D0%BA%D0%B8), который может быть прочтён человеком. В обобщённом смысле — любые входные данные для [транслятора](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%BB%D1%8F%D1%82%D0%BE%D1%80). Исходный код [транслируется](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%BB%D1%8F%D1%82%D0%BE%D1%80) в исполняемый код целиком до запуска программы при помощи [компилятора](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80) или может исполняться сразу при помощи [интерпретатора](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D1%80%D0%B5%D1%82%D0%B0%D1%82%D0%BE%D1%80).

Сборка — процесс получения подготовленного для использования информационного продукта из [исходного кода](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4). Чаще всего включает [компиляцию](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) и [компоновку](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%BE%D0%BD%D0%BE%D0%B2%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), выполняется инструментами автоматизации. В результате сборки получают [исполняемый файл](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D0%BF%D0%BE%D0%BB%D0%BD%D0%B8%D0%BC%D1%8B%D0%B9_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8C) — [двоичный файл](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D1%8B%D0%B9_%D1%84%D0%B0%D0%B9%D0%BB), содержащий [исполняемый код (машинные инструкции)](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) [программы](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0) или [библиотеки](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)).

Локальная сборка может осуществляться по внешнему запросу, по расписанию, по факту обновления репозитория и по другим критериям.

Сборки по расписанию, как правило, проводятся в нерабочее время, ночью, и планируются таким образом, чтобы к началу очередного рабочего дня были готовы результаты тестирования. Для различия дополнительно вводится система нумерации сборок — обычно, каждая сборка нумеруется натуральным числом, которое увеличивается с каждой новой сборкой. Исходные тексты и другие исходные данные при взятии их из репозитория (хранилища) системы контроля версий помечаются номером сборки. Благодаря этому, точно такая же сборка может быть точно воспроизведена в будущем — достаточно взять исходные данные по нужной метке и запустить процесс снова. Это даёт возможность повторно выпускать даже очень старые версии программы с небольшими исправлениями.

Сборка проекта вовлечена в процесс непрерывной интеграции, которая заключается в постоянном слиянии рабочих копий в общую основную ветвь разработки (до нескольких раз в день), и непрерывной доставки, при которой программное обеспечение производится короткими итерациями, гарантируя, что ПО является стабильным и может быть передано в эксплуатацию в любое время.

1.6 Анализ существующих аналогов-прототипов, реализующих перспективные подходы

На рынке существует большое количество платных и бесплатных инструментов для настройки такого процесса, который мы называем “процессом непрерывной интеграции” или CI/CD. Однако для построения эффективного процесса недостаточно просто установить инструмент. За каждым этапом конвейера стоит сложная логика того, что должно быть запущено, на каких вычислительных ресурсах и как эти ресурсы используются.

Инструменты CICD можно классифицировать. Выделяют:  
1. Локальные сервисы. К ним можно отнести GitLab CI, TeamCity, Bamboo, GoCD Jenkins, Circle CI.

2. Облачные сервисы, например, BitBucket Pipelines, Heroku CI, Travis, Codeship, Buddy CI, AWS CodeBuild. Если у команды нет потребности в локальном хостинге, то можно использовать именно их. Для личных проектов все большее число репозиториев кода предоставляют решение «все-в-одном», которое дает доступ к инструментам CI.

3. Правительственные сервисы, предназначенные для тех, кто работает в правительстве и хочет улучшить качество своих приложений, которые не могут быть размещены в общедоступном облаке. Возможными вариантами являются Hats и Nectar.

Некоторые из компаний предоставляют разработчикам возможность использовать как локальную, так и облачную версию своего инструмента. Помимо инсталляторов существуют различные библиотеки и расширения для сред разработки и редакторов кода, например, для VS Code.

Проанализируем самые популярные из них. Сравнительный анализ представлен в таблице 2.

Таблица 2 — Достоинства и недостатки различных инструментов CICD

|  |  |  |
| --- | --- | --- |
| Инструмент | Достоинства | Недостатки |
| Gitlab | 1.Популярность и активная поддержка сообщества.  2.Поддержка GitLab Pages и Jekyll.  3.Автоматическое масштабирование CI-раннеров.  4.Средства для отслеживания проблем. 5.Ограничение доступа к репозиториям  6.Поддержка работы с различными системами контроля версий  7.Надёжная система безопасности. | 1.Для каждой задачи нужно описывать и загружать/выгружать артефакты.  2.Нельзя протестировать результаты объединения веток до их фактического объединения.  3.При описании стадий CI/CD-конвейера в них пока нельзя выделять отдельные этапы. |
| Jenkins | 1.Бесплатность, открытый исходный код, поддержка множества платформ. Jenkins может работать на платформах macOS, Windows и Linux. Он может функционировать и в среде Docker, что позволяет организовать единообразное и быстрое выполнение автоматизированных задач. Этот инструмент, кроме того, может выполняться в виде сервлета в контейнерах, поддерживающих Java, в таких, как Apache Tomcat и GlassFish.  2. Развитая экосистема плагинов.  3. Простая установка и настройка.  4.Качественная документация. 5. Наличие REST API.  API для удалённого доступа к системе представлен в трёх вариантах: XML, JSON с поддержкой JSONP, Python. 6. Поддержка параллельного выполнения задач. 7. Поддержка работы в распределённых средах.  8.Полный контроль над установкой инструмента.  9.Поддержка различных языков программирования. | 1.При использовании плагинов могут возникать сложности.  2.При использовании Jenkins в маленьких проектах затраты времени, необходимые на его самостоятельную настройку, могут оказаться неоправданно большими.  3.Отсутствие общих аналитических сведений по CI/CD-цепочкам. |

1.7 Анализ и выбор (с обоснованием) релевантных моделей, методов и средств, технологий для решения задачи

Для автоматизации CICD процесса при реализации проекта одним человеком подойдет инструмент, установленный локально на машину, на которой производится разработка.

В качестве системы контроля версий был выбран Git, для работы с которым необходимо изучить работу хуков. Git предоставляет возможность запуска пользовательских скриптов в случае возникновения определённых событий. Такие действия называются хуками и разделяются на две группы: серверные и клиентские. Если хуки на стороне клиента запускаются такими операциями как слияние или создание коммита, то на стороне сервера они инициируются сетевыми операциями, такими как получение отправленного коммита.

1.8 Точная постановка задачи с ограничениями к ней, требованиями к ее решению, ограничениями на возможные методы и способы решения задачи

Сервис по автоматизации CICD процесса может представлять собой npm библиотеку. Преимущество этого варианта в том, что готовую работу легко опубликовать, используя аккаунт на сайте npm и пару npm команд, и не менее легко подключить к нужному проекту.

Для работы с системой контроля версий отлично подойдет husky. [Husky](https://github.com/typicode/husky) — это инструмент, который позволяет нам легко перехватывать Git-хуки и запускать нужные скрипты на этих этапах.

Он работает путем включения объекта прямо в наш package.json файл, который настраивает Husky для запуска указанных нами сценариев. После этого Husky управляет тем, на каком этапе жизненного цикла Git будут запускаться наши скрипты.

Он использовался в таких проектах как [webpack](https://github.com/webpack/webpack), [angular](https://github.com/angular/angular), [docusaurus](https://github.com/facebook/docusaurus), [typescript-eslint](https://github.com/typescript-eslint/typescript-eslint), [react-bootstrap](https://github.com/react-bootstrap/react-bootstrap), [NodeBB](https://github.com/NodeBB/NodeBB), [ant-design](https://github.com/ant-design/ant-design).

Его достоинства:

— отсутствие большого числа зависимостей и легкий вес (6 КБ);

— работает на новой современной функции Git (core.hooksPath);

— следует рекомендациям npm и yarn в отношении автоматической установки;

— поддерживает macOS, Linux и Windows;

— поддерживает Git-интерфейсы;

— поддерживает пользовательские каталоги;

— поддерживает монорепозитории.

Список использованных источников

1.CI/CD Википедия [Электронный ресурс] URL: <https://ru.wikipedia.org/wiki/CI/CD>

2.Непрерывная интеграция Википедия [Электронный ресурс] URL: https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%BF%D1%80%D0%B5%D1%80%D1%8B%D0%B2%D0%BD%D0%B0%D1%8F\_%D0%B8%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B0%D1%86%D0%B8%D1%8F

3. Gitlab Documentation [Электронный ресурс] URL: <https://docs.gitlab.com/>

4. Jenkins Documentation [Электронный ресурс] URL: https://www.jenkins.io/doc/

5. Github Actions Documentation [Электронный ресурс] URL: https://docs.github.com/en/actions

6. Buddy Documentation [Электронный ресурс] URL: https://buddy.works/docs

7. Метод CI/CD: чистый код и строгая дисциплина [Электронный ресурс] URL: https://habr.com/ru/company/automacon/blog/671918/

8. Когда и зачем нужен CI/CD [Электронный ресурс] URL: <https://habr.com/ru/company/southbridge/blog/649027/>

9. [Электронный ресурс] URL: <https://habr.com/ru/company/ruvds/blog/522334/>

10. [Электронный ресурс] URL: <https://about.gitlab.com/devops-tools/github-vs-gitlab/>

11. [Электронный ресурс] URL: <https://softlist.biz/articles/gitlab-vs-github-kto-luchshe-d/>

12. [Электронный ресурс] URL: <https://medium.com/southbridge/ci-cd-%D0%BF%D1%80%D0%B8%D0%BD%D1%86%D0%B8%D0%BF%D1%8B-%D0%B2%D0%BD%D0%B5%D0%B4%D1%80%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B8%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D1%8B-f0626b9994c8>

13. [Электронный ресурс] URL: <https://sbercloud.ru/ru/warp/blog/cicd-about>

14. [Электронный ресурс] URL: <https://typicode.github.io/husky/#/?id=used-by>

15. [Электронный ресурс] URL: <https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks>