Цель: рассказать о влиянии методологии CI/CD (Continuous Integration, Continuous Delivery — **непрерывная интеграция и доставка**) на разработку программного обеспечения.

CONTINUOUS DELIVERY

**ВСТУПЛЕНИЕ**

В реальной жизни деплой приложения для разработчика ставится очень проблематичесной задачей, особенно для начинающих разработчиков, каторые не имеют никакого отношения к администрированию операционных ситсем.

Раньше разработчики не задумывались о необходимости создания инструментов для автоматизации, потому что потоки данных были не очень большими и производительность серверов была не столь высока. Разработчики просто копирывали все файлы со своего компьютера на удаленный сервер по ftp, но перед этим им приходилось как-то обмениваться кодом со всей командой, ждать пока тестировщики все проверят и задачик одобрит проект, а в случае неудачи все начинается с начала. Это было весьма неудобно и времезатратны, поэтому были созданы инструменты для автоматизации этих рутинных задач. Теперь разработчикам остается только писать код, заливать их в свою ветку и ждать результатов.

Это послужило началом для создания инструиментов автоматизации ПО

Автоматизация разработки нужна для автоматического развертывания, тестирования и деплоя приложения. То есть каждый этапп разработки кроме написания кода в идеале должен происходить автоматически.

В работе будут приводиться примеры из репозитория автора, ссылку на репозиторий, а так же остальные ресурсы вы сможете найти в конце документа

**ПЛЮСЫ И МИНУСЫ**

Автоматическое развертывание имеет множество плюсов:

1. разработчики могут сконецентрированися на написании кода ине отвлекаться на сборку и развертывание проекта
2. все этапы разработки кроме написания кода происходят автоматически
3. отпадает необходимость в большом количетве dev ops-ов.
4. время до деплоя значительно сокращается
5. появляется версионирование
6. быстрый откат, в случае неполадок
7. меньший шанс прохождения ошибки в продакшн
8. возможность настройки тестового и реального сервера одновременно, без надобности создания отдельных конфигов (не считая переменных среды)
9. быстрая интеграция сторонних библиотек в проект
10. динамическое проведение тестов при сборке
11. можно настроить лишь один раз и будет работать всегда

Минусы:

1. переход на автоматизацию занимание довольно много времени
2. обновления в ветку должны осуществляться очень часто для максимальной эффективности
3. зависимоть от стороннего софта

Одним из самых популярны подходов к автоматизации является ci cd. Данных подход является подможеством agle артиектуры и широко используется всеми большими компаниями мира, таким как гугл и еще.

**МОДЕЛИ ПО (далее “Модель”)**

Модели разработки:

1)Основой для заложения всех моделей являеется Водопадная модель разработки ПО У. У. Ройсома 1970 года (вставить картинку). Суть этой моели в том, что каждый этап разработки начинается только после завершения предыдущего . Она была популярна почти до 2000 годов,

2)но ей на смену пришла **Гибкая методикиа разработки** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *agile software development*, *agile-разработка*) — обобщающий термин для целого ряда подходов и практик. Суть методики: данная методика подразумеванет постоянное взаиможействие заказчика с группой разработчика. Самая главная цель - выпустить минимально жизнеспособный продукт. Этот подход очень эффективен и продуктивен по сравнению с другими и часто используется в компаниях-гигантах.

3)RAD-модель — разновидность инкрементной модели. Эта модель подразумевает максимально быструю разработку ПО. Обычно проект состоит из независимых модулей, каторые разрабатываются и внедряются параллельно. Однако каждый этап разработки подразумевает использование состояния прошлого этапа. Сборка приложения должна проводиться в автоматическом режиме.

4)итеративная или итерационная модель. Подходит для очень длительных и больших проектов. Важной частью этой модели является четкое понимание конечной цели. Т.е. не обязательно сразу давть готовый продукт, а можно представлять его маленькими функциональными частями, но при этом стремиться к достижению поставленой цели и усовершенствовать продукт с каждым новым релизом.

5)«Incremental Model» (инкрементная модель) подразумевает постепенное внедрение модулей. Эта модель очень похожа на модель водопада, но проект не является одним целым, а состояит из независимых модулей, дополняющих базовый функционал.

Мы перечислили основные и самые популярные модели ПО. Помимо перечисленных существует еще огромное количество моделей разработки, таких как спиральная, v-образная, однако они не нашли широкого применения и считаются специфичными.

**МЕТОДОЛОГИИ:**

Если модель описывает, какие стадии жизненного цикла оно проходит и что происходит на каждой из них, то методологии описывают методы кправления разработкой.

Самый вариативной моделью является agile-model. Она подразделяется на такие методологии как:

1)scrum - методология которая подразумевает поэтапное выполнение задач с помощью “спринтов”. Спринт - это какой-то промежуток времени, за каторый команда должна справиться с поставленой задачей. В свою очередь, команда состоит из разработчиков, дизайнеров, тестировщиков и других людей необходимых для полной независимоти. ТАких команд обычно несколько и каждая занимается своей задачей. Заказачик или scrum-мастер создает список задач (бэклог), в котором создает и расставляет приоритеты для задач. Когда спринт заканчивает комманда переходит к выполнению следущей задачи, независимо от результата. Это помогает выпустить полностью функцилональный продукт в указанный срок. Самым важным показателем является скорость работы команды. В ходе спринта команды стремятся избегать изменений в прогнозах спринта: изменения приведут к неверным выводам относительно оценки задач

2)kanban - это методология подразумевающая выполнение задач точно в срок. Данный подход является очень популярным и интересным. Вся методололгия завязана на доске задач (бэклоге). Бэклог - это место где заказчик или team-lead размещает задачи для выполнения и цель каждого разработчика уменьшить количество этих задач. Самым важным показателем является продолжительность цикла. Стоит отметить, что изменения к требованиям продукта могут происходить в любое время и это никак не повлияет на разработку.

3)ci/cd - это комбинация непрерывной интеграции и непрерывного развертывания программного обеспечения в процессе разработки. Подразумаевает полностью автоматизированный процесс разработки. Методология строится на доверии между заказчиком и командой разработки. Основное правило методологии: делать изменения кода как можно чаще. Сегодня мы остановимся на этой методологии, потому что к моменту написания она является самой популярной, удобной и эффективной для ведения бизнеса.

**CI/CD**

Для начала выясним что такое ci cd и для кого оно нужно:

Как уже было сказано данная методология относится к agile модели и значит наследует её образ мышления Это означает что проект может меняться в процессе разработки и быстро адаптироваться к новым условиям.

Цифры и статистика:

вставить картинку со статистикой

В декабре 2017 года компания Digital Ocean провела опрос Currents среди разработчиков (и не только) по использованию ci cd .Около 58 % не используют CI/CD. Среди них 46 % считают, что потребности в CI/CD для их рабочего процесса нет (т.е. около 27 % от общего числа участников опроса), а 38 % — собираются внедрить CI/CD.

По результатам опроса было выявлено что самым популярным инструментом ci cd является Jenkins 44%, второе место занимает GitLab 39% и третье Travis 26% .

ВЫБОР КОМАНИЙ:

При выборе инструмента ci cd компании часто образают выбор на скорость разработки и бюджет. Для примера сравним два самым популярных инструмента и выявим сильные и слабые стороны для них:

Будем сравнивать Jenkins и GitLab:

Jenkins:

Во-первых Jenkins это спонсорский проект и его нужна разворачивать самостоятельно, это дает более гибкую настроку во время установки. У него можно выделить несколько сильных сторон:

* развитая система управления учетными данными
* огромное количество плагинов
* интеграция со сторонними сервисами
* поддержка разных языков программирования
* Настройка ранеров очень гибкая и довольно простая.

Минусы:

* при использовании плагинов, часто приходится изучать совершенно новую документацию, нет централизованных команд для выполнения задач
* настройка занимает много времени

GitLab CI/CD:

Это очень популярный интсремент разработки и поставляется вместе с gitLab, следовательно его не нужно устанавливать отделаьно. Рассмотрпим плюсы использования:

* возможность параллельного выполнения ранеров
* быстрая интеграция со сторонними сервисами
* надежная система безопасности
* полный контроль на git-Hub репозиторием
* простота настройки (имеется графический интерфейс)

Минусы:

* Нельзя протестировать результаты объединения веток до их фактического объединения
* При описании стадий CI/CD-конвейера в них пока нельзя выделять отдельные этапы

Подводя итог, можно выделить несколько критериев для выбора инструмента ci/cd:

Когда стоит выбрать GitLab CI/CD:

* Если проект небольшой. Такой проект будет быстрее и удобнее настроить
* Если вам нужно выполнять несколько ранеров в одновременно
* Если вы в большей степени зависите от своего gitHub репозитория
* Если вам нужно готовое решение без самостоятельной установки

Когда стоит выбрать Jenkins:

* Вам нужна простота в развертывании кода
* Вы нуждаетесь в большом количестве плагинов
* У вас сложный проект состоящий из множества частей

Стоит отметить, что оба инструмента поддерживают javascript, контроль качества кода, создание ci cd конвейеров.

Этапы:

вставить картинку с этапами

Структура ci cd является циклом, каторый обычно изоображают в виде петили, означающей бесконечно повторяющиеся операции. Она является прототипом SDLS (System/**Software Development Life Cycle** - Жизненный цикл разработки ПО)

План: на этом этапе зарождается идея проекта и ее главная цель, что заказчик хочет видеть в проекте, а так же обсуждается бюджет и процесс разработки

Код: Разработчики приступают к написанию кода и постепенному внедрению его на платформу (напрмиер gitHub, JIra)

Сборка: Это процесс автоматизации сборки проектов на основе описания их структуры в файлах. (например: mave, webpack)

Тест: самая важная часть, в которой проверяется качество, стиль кода, а так же ошибки в нем. Существует много инструментов тестирования ( JTEST, Se)

Релиз: Обсуждение выполненныех задач с заказчиком и командой, подключение всех метрик и сторонних сервисов. Подготовка к выпуску проекта в продакшн

Развертывание: выгрузка проекта на сервер (docker, aws)

Управление: помогает соединить все части проекта в одно целое и дает возможность управлять всем одновременно (Kubernetes, Chef)

Мониторинг: выявление влияния нового релиза на бизнес. Сбор статистики по посещаемости (в случае если это web-сервис), улучшению пользовательского опыта и другим параметрам, а так же анализ этих данных. (Яндекс метрики, splunk)

После появления новых задач, команда программистов снова начинает писать код и процесс повторяется.

CI/cd

CI непрерывная интеграция- подразумет постоянное обновление кода, т.е разработка ведется динамически. Максимальная эффективность этого подхода будет достигаться, когда разработчики будут чаще делать изменения своего кода и сливать их с другими ветками, минимум 1 раз в день. Это очень удобно когда в команде много задач и проект очень большой. Если рассматривать непрервыную интеграцию как отдельную часть, то она будет выглядеть слудующим образом: (вставить картинку).   
Сначало разработчики пишут код, потом заливают его куда то, например в свою ветку github. Там идет автоматическоая сборка проекта из исходного кода, далее идут аавтоматические тесты и если все эти операции прошли успешно, тот проект передается дальше на развертывание, смерживание с основной веткой, программистам или куда-угодно. На этом этап непррывной интеграции заканчивается. Однако в случае ошибки при сборке или тестах код не будет никуда отправляться и разработчик будет заниматься отладкой, а после процесс будет повторяться

CD

На самом деле под CD могут подразумевать либо continuous delivery, либо continuous deployment.

continuous delivery (непрервыная доставка) - описывает процесс от начала разработки до успешного проведения тестов. Следует сразу после CI. Этот этап включает развертывание результатов на тестовый сервер (если такой имеется). К этому этапу весь проект должен быть готов к публикации. Отличительной особенностью этого этапа, является то, что развертывание проекта на продакшн должно происходить только после подтверждения человеком. Т.е. весь этап разработки должен быть автоматическим, кроме начала выгрузки на сервер. Для примера мы можем рассмотреть пример созданный с помощью gitHub actions.

После подтверждения начнется автоматический процесс деплоя на сервер.

continuous deployment (непрерывное развертывание) - тоже тсамое, что и continuous delivery, но только весь процесс разработки происходит автоматически, включая развертывание на сервер.

Каждая компания стремится к полной автоматизации всех процессов разработки, но это требует оченб больших матреиальных вложений в разработку и других факторов:

1. теститорование должно покрывать больше 90 процентов кода
2. поддержка версионирования
3. четкая организация работы в команде
4. минимум один высококвалифицированный разработчик на команду
5. очень быстрая и отзывчивая работа разработчиков.

После достижения этих целей вы получите:

1. заметное снижение затрат на тестирование. CI сервер теперь сможет быстро выполнять огромное количество тестов. (например около 100-200 в секунду)
2. возможность чаще выпускать релизы, тем самым уменьшая конкуренциюэ
3. исключение человеческого фактора и снижение рисков возможных багов
4. быстрая отзывчивость под интересы клиентов (в течении недели, а не года как это обычно бывает)
5. снижается нагрузка на DevOps ов

Pipeline:

CI/CD pipeline - это конвейр непрервыной интеграции и развретывания в катором описан порядок действий реализации проекта. Он автоматизируют процессы в жизненном цикле разработки программного обеспечения (SDLC), осуществляя плавную интеграцию и поставку новых функций. (встпавить картинку). PIpeline дает наглядное представление как работает проект и хорошо подходит для разделения этапов на еще более мелкие подзадачи. Мы уже видели пример ci/cd pipeline, когда описывали структуру ci cd методологии. (Вставить картинку из проекта и дописать текст).

Сравнение CI/CD с другими методологиями:

Agile архитектура уже давно превзошла старую каскадную модель и показала, что имеет весомое место в современном мире. По опросу tadviser к 2021, около 60% компаний используют agile в своих проектах. Не смотря на это требования к проекту постоянно меняются и разработчикам важно быстро реагировать на эти изменения.

Стоит отметить, что некоторые разработчики не выделяют ci cd как отдельную методологию agile, а считают её практикой dev-ops.

По сравнению с другими методологиями, ci/cd имеетслудующие преимущества:

1. Конвейеры CI/CD: автоматизация снижает затраты
2. Обеспечивает прекрасное качество кода
3. Улучшенное среднее время до разрешения (MTTR)
4. Сокращение некритических дефектов в бэклоге
5. Локализация отказов

**Пример создания тестового ci cd приложения:**

Исходный код проекта вы можете посмотреть на моем github

В данном примере не будут обьяняться подробности написания кода, акцент будет сделан на gitHub actions, для наплядного примера ci cd pipeline

Этапы описания будут соответствовать ci cd конвейру.

1)План

Стек приложения: react + nodeJs + docker + github actions.

Описание: Данное приложение возвращает рандомное число с сервера при увеличении/уменьшении счетчика

Цель: показать процесс прохождения ci/cd pipeline

Структура проекта: (фото структуры)

2)Код:

Здесь мы подробно рассмотрим только создание ci cd pipeline c помощью actions.yml файла.

1. Напишем моковый сервер, клиент
2. ВСЕ ДЕЙСТВИЯ БУДУТ НАГЛЯДНО ПОКАЗАНЫ В ЭТАПЕ СБОРКИ. Создадим main.yml в workflow дирректории. Определим этапы работы нашего приложения (jobs). Jobs указывает стадии нашего проекта. Для краткости определим в них сборку, тестирование, выгрузку на тестовый сервер и ожидание выгрузки на продакшн сервер. Запишем зависимости этапов: т.е. тестирование будет дожидаться выполнения сборки с помощью тега needs и так далее по цепочке. Настроим что бы сборка начиналась при каждом push е в bubu ветку. Для наглядности запросим подтверждение перед загрузкой на сервер с помощью переменной среды 'prod', это называется continious delivery, потому что процесс разработки не полностью автоматический, а требует минимального вмешательства человека.
3. Для удобства поместим это все в docker контейнер
4. создадим dockerfile на клиенте и сервере.
5. Объединим все в docker-compose и добавим СУБД (Система управления базами данных) - mysql. Здесь же добавим тома и пробросим порты.
6. Добавим images в Docker Hub: <https://hub.docker.com/repository/docker/5565655656565665655/ci-cd> (в данном руководстве мы сделаем это механически на этапе разработки кода для облегчения сложности)
7. Зальем код на gitHub: <https://github.com/Dragnils1/ci-cd-docker-actions>

3,4) этап сборки и тестирования будет проходить автоматически поэтому мы пронаблюдаем, как это работает в github actions

1. На рисунке мы видим ci-cd pipeline, созданный с помощью gitHub Actions.
2. Запустим сборку проекта, это самая продолжительная часть конвеера. (картинка)
3. После выполнения сборки запускаются тесты, причем ui-mock-test и server-mock-test выполняются параллельно. Далее идет выгрузка проекта на тестовый сервер, не дожидаясь окончания UI тестов. Это было сделано для примера, чтобы показать как может работать конвейер. (картинка)

5) Этап релиза наступает когда продукт выложен на тестовый сервер и у группы тестировщиков вместе с заказчиком есть возможность проверить работоспособность проекта и найти возможные баги. (картинка)

Теперь когда мы убедились что все тесты были пройдены, мы можем применить изменения (картинка)

6)этап деплоя мы сымитировали ручной загрузкой проекта в docker-hub

7)управление проекта происходило с помощью docker-compose на этапе написания кода. Он объединяет все образы (images) в один контейнер и задает конфигурацию проекта. В крупных проектах для этого в основном используется kubernetes.

8)Мониторинг - мы пропустим данный этап, по понятным причинам.

В итоге мы имеем полностью работающее приложение, включающее все этапы конвйера (картинка)

Быстро рассмотрим развертывание того же приложение в gitlab, только уже в комбинации с continuous deployment: <https://gitlab.com/Dragnils1/ci-cd-docker-actions>

Вместо сборки проекта в докер контейнере мы будем эмулировать его загрузку.

Для публикации проекта на gitLab мы можем использовать тот же репозиторий github, только нам нужно написать файл .gitlab-ci.yml в котором определены стадии сборки проекта (stages).(картинка)

Настройка проекта очень проста и занимает в разы меньше времени чем другие инструменты ci/cd. После выполнения конвейера, мы что все прошло успешно и австоматически выложено на сервер, это называется continuous deployment. (картинка)

**ЗАКЛЮЧЕНИЕ**

Исследование помогает команиям выбрать подходящую модель разработки ПО. Определиться с методологий и выбрать инструменты для полного цикла разработки ПО. Это дает общее понимание для начинающих разработчиков, как работает ci-cd pipeline, что это такое и для чего он нужен

**Использовалось**:

<https://earchive.tpu.ru/bitstream/11683/37406/1/reprint-nw-15418.pdf>

<https://ru.wikipedia.org/wiki/%D0%93%D0%B8%D0%B1%D0%BA%D0%B0%D1%8F_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%8F_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8>

<https://habr.com/ru/company/edison/blog/269789/>

<https://ru.wikipedia.org/wiki/CI/CD>

<https://www.atlassian.com/ru/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment>

<https://assets.digitalocean.com/currents-report/DigitalOcean-Currents-Q4-2017.pdf>

<https://habr.com/ru/company/flant/blog/346418/>