DevOps

DevOps является одним из самых популярных способов взаимодействия между разработчиками того или иного программного обеспечения. При этом, данная практика включает в себя совокупность процессов по созданию, поддержанию и дальнейшему обслуживанию программного обеспечения. Каждый процесс не может существовать друг без друга, так как именно на этом строится вся суть взаимодействия между разработчиками программного обеспечения. DevOps состоит из следующих задач, которые необходимо решить разработчикам:

1. Непосредственное создание программного обеспечения, направленного на решение тех или иных производственных задач.
2. Тестирование промежуточных версий или уже готовых продуктов программного обеспечения.
3. Эксплуатация и тестирование готовой продукции на практике.

CI/CD/CDP

Основная идея CI/CD/CDP - создание автоматизированного конвейера поставки продукта заказчику за счёт автоматизации:

1. чекина изменений в репозиторий;
2. статического анализа кода: на уязвимости, на соответствие требованиям и стандартам;
3. компиляции и формирования сборки;
4. передачи обратной связи об успехе/неудаче сборки в виде уведомлений по выбранным каналам;
5. деплоев на dev / test / staging / prod;
6. всех необходимых тестов (unit, smoke, acceptance, regression, integration, end-to-end);
7. заведения issue в СУП в случае ошибок;
8. слияния изменений с нужной веткой в случае "зелёных" тестов.

Continuous integration, CI = автоматизированная интеграция программного кода в существующий проект в репозитории с последующей компиляцией,

формированием сборки и прогоном базовых автотестов. Считается, что должно занимать <= 10 минут.

Continuous delivery, CD(L) = CI + автоматизированная поставка готовой сборки ПО с изменениями на сервера разработки и тестирования с прогоном автотестов.

Continuous deployment, CDP = CD + автоматизированное развёртывание изменений в Пром.

Всё это счастье достигается посредством использования таких инструментов как, например, GitLab CI, BitBucket Pipelines, AWS CodeBuild, Bamboo, TeamCity, Jenkins и сильно облегчающей деплои контейнеризации (Docker).

Роль тестировщика в процессе непрерывной поставки

Тестирование — один из процессов, который может быть автоматизирован в рамках CI/CD. *Надёжное и тщательное автоматизированное тестирование позволяет быть уверенным в новых сборках, снижает затраты на производство и повышает качество продукта.*

Есть два варианта расшифровки аббревиатуры CD — continuous delivery и continuous deployment.

Continuous delivery или **непрерывная доставка** — процесс, по окончании которого мы можем нажать на кнопку и доставить новую версию приложения нашим клиентам.

В **continuous deployment** или **непрерывном развёртывании** кнопки и ручного шага у нас нет. Всё проходит автоматически. Но поскольку выкатывать изменения клиентам в фоне без нашего участия сложно, непрерывное развёртывание возможно только при наличии стратегии тестирования и системы мониторинга. Тогда если вдруг что-то пойдёт не так, мы узнаем об этом раньше клиентов.

Из чего состоит CD: после CI у нас есть работающая версия приложения, мы разворачиваем в её разных окружениях и запускаем набор тестов. Обычно таких окружений три (но всё зависит от вашей схемы работы с системой контроля версий GIT и релизного процесса):

- В **dev окружение** (окружение разработки) мы выкатываем все новые версии, в которых уже закончена разработка — это ещё не релизы, а просто новый функционал. Здесь мы проводим, как минимум, интеграционное тестирование и smoke-тестирование, чтобы убедиться, что наше приложение не сломалось.

- Далее ветка, где велись изменения, сливается с основной веткой. И версию приложения, получившуюся после этого слияния, мы разворачиваем в **staging окружении**. На этом этапе мы запускаем E2E-тестирование (end-to-end тесты для проверки полного функционала), плюс, дополнительно обычно работает мануальный тестировщик, который будет проверять наше приложение как реальный пользователь. Также могут проводиться разнообразные security-тесты.
- По завершении тестов мы говорим, что версия приложения стабильна, не имеет багов и готова к развёртыванию в **продуктовой среде**. Дальше всё зависит от того, какой процесс релизов установлен в команде. Мы либо разворачиваем всё вручную по кнопке, либо это происходит автоматически по наступлению какого-то события. В конце мы запускаем ряд тестов и просим проверить функционал наших тестировщиков и (или) реальных пользователей (например, если они готовы быть Beta-тестировщиками, и у нас есть Feature toggling в приложении). Далее следим за количеством ошибок в системах мониторинга и логах наших приложений, чтобы отловить потенциальные проблемы.

Окружений может быть больше, например, если мы собираемся проводить performance- и расширенные security-тесты. Запускать их на одном из трёх перечисленных окружений, значит потерять его на время, поэтому для них выделяют отдельные окружения.

Непрерывное тестирование (CT)

Непрерывное тестирование — то, что работает в пределах CI/CD вообще. Прежде чем организовать непрерывное тестирование как процесс, нам нужно сформировать стратегию тестирования: выделить группы тестов, которые мы будем запускать в разное время, чтобы проверить, насколько корректно работает приложение.

В CI-фазе важно, чтобы сборка проходила быстро, поэтому чаще всего применяют облегчённые типы тестов:

- unit-тесты — чтобы протестировать отдельные компоненты;
- интеграционные тесты — чтобы проверить интеграцию этих компонентов;
- базовый линтинг — чтобы проверить код на соответствие тому, как мы договорились писать его внутри команды;

- статистический код-анализ — чтобы отловить потенциальные уязвимости в нашем коде;
- smoke-тесты — чтобы проверить, развернулось ли приложение, и корректно ли работают его базовые компоненты.

Можно делать больше тестов, но обычно этих 5 при условии, что они хорошо написаны, достаточно. Запустив их, мы уже предупредим около 90% багов на стадии CI.

На стадии CD идёт непрерывное тестирование конкретных окружений, поэтому тестов становится больше, и они дольше по времени:

- E2E-тесты — чтобы проверить корректность работы UI;
- performance-тесты — чтобы проверить, нет ли просадок по производительности;
- регрессионные тесты — чтобы убедиться, что мы не вернулись к багам, исправленным в прошлом, перед выходом релиза на продакшн-среды;
- security-тесты, «ломающие» наше приложение — чтобы проверить, что у нас нет уязвимостей;
- penetration-тесты — чтобы проверить устойчивость приложения к взломам и отражению DDoS атак.

Глубина и набор тестов варьируются от окружения к окружению. Например, E2E-тестирование мы запускаем перед выходом в продакшн, когда хотим убедиться, что всё корректно работает. Если у нас не только веб-приложение, но ещё и версии для iOS и Android, такое тестирование может занять до двух дней в автоматическом режиме.

Разработка стратегии тестирования — работа не только опытных тестировщиков, но и DevOps-инженеров. Тесты являются частью CI/CD, поэтому специалисты, реализующие DevOps, тоже должны участвовать в их создании. Ещё не обойтись без представителей бизнеса, которые должны дать на всё это деньги. Разработка непрерывного тестирования — это долго и дорого, но в долгосрочной перспективе она даёт невероятные результаты.

Команды, использующие непрерывное тестирование, могут добиться следующих результатов:

- **Быстрые и стабильные релизы**. Благодаря непрерывным циклам обратной связи, вы рано находите баги и оперативно исправляете их. Время на разработку и выпуск релиза сокращается.

- **Высокая эффективность**. За счёт автоматизации ресурсы команды расходуются более рационально. Разработчики могут сосредоточиться на разработке, а тестировщики — отказаться от повторяющихся задач в пользу задач, требующих критического и творческого мышления. В результате охват и качество тестов могут быть увеличены.
- **Снижение рисков**. Автоматизация позволяет исключить человеческий фактор. Роботы выполняют повторяющиеся задачи с большим количеством данных лучше и быстрее людей. Риски в области автоматического тестирования снижаются.
- **Уменьшение затрат**. Автоматизация тестирования позволяет раньше обнаруживать баги и ошибки. А чем раньше они обнаружены, тем проще и дешевле их исправить.

Quality Gates

Это заранее определенные этапы, во время которых проект проверяется на соответствие необходимым критериям для перехода к следующему этапу. Quality Gates являются важным компонентом официальных процессов управления проектами, используемых различными организациями.

Цель Quality Gates – обеспечить следование набору определенных правил и передовых практик, чтобы предотвратить риски и увеличить шансы на успех проекта. С помощью качественных Quality Gates организации могут гарантировать, что руководители проектов выполняют свою работу и не пропускают никаких важных шагов.

В своей *практической реализации Quality Gates* организованы в виде совещаний, которые запланированы в конце каждого этапа проекта. Вот как это обычно выглядит:

Quality Gates основаны на чек-листах, по которым менеджеры проектов должны пройти на разных этапах жизненного цикла проекта. Эти чек-листы включают в себя ряд вопросов, касающихся различных аспектов проекта, включая объем работ, бюджет, заинтересованные стороны, риски и соответствие требованиям.

Перед совещанием по вопросам качества руководитель проекта изучает соответствующий чек-лист по определенным Quality Gates и отвечает на каждый вопрос, принимая во внимание текущий статус проекта. Он также предоставляет заполненный чек-лист соответствующим лицам, принимающими

решения, чтобы дать им достаточно времени для изучения информации до совещания по качеству.

Во время совещания участники знакомятся с чек-листом и обсуждают наиболее важные пункты чек-листа. Менеджер проекта предоставляет контекст и отвечает на любые возникающие вопросы.

Обсуждение в основном вращается вокруг вопросов, которые не были завершены – пунктов чек-листа, на которые были даны ответы “Нет” или “В процессе”. Возможно, какая-то роль в проекте еще не заполнена или бюджет еще не подписан клиентом.

В зависимости от темы и серьезности проблемы, команда Quality Gates, состоящая в основном из руководства, спонсора проекта и ключевых заинтересованных сторон, может начать предпринимать различные действия. Если у кого-то из команды на данный момент нет работы, это не проблема, и проект может продолжаться. Однако, если бюджет проекта все еще обсуждается, проект следует приостановить до тех пор, пока не будут утверждены расходы.

Человек, который контролирует соблюдение Quality Gates, может также потребовать принятия дополнительных мер для конкретных пунктов чек-листа. Если, например, одна из заинтересованных сторон выражает озабоченность по поводу того, соответствует ли планирование ресурсов правилам управления персоналом, она может отправить запрос, чтобы отдел кадров проверил план ресурсов проекта.

Рассмотрим Quality Gates для каждой из фаз разработки продукта (SDLC).

Важно отметить, что только после устранения блокеров, критических и серьезных проблем приложение разрешается допустить в продакшн, интегрироваться с другими приложениями и источниками данных, опубликовать в Интернете.

Фаза SDLC	Необходимые практики	Quality gates
Фаза исследования продукта	Тренинг для разработчиков	Команда разработчиков прошла тренинг Tech Lead/Dev Lead есть на проекте
Анализ требований	Анализ рисков	Определен Product Data Owner Проведена классификация данных Определены требования и их соответствие Проведен анализ рисков по приложению
Разработка дизайна	Произведена оценка архитектуры приложения Произведено моделирование архитектуры Созданы макеты приложения	Созданы документы по архитектуре приложения Создана модель возможных угроз
Разработка продукта	Статический анализ кода Code Review	Произведен анализ найденных дефектов Найдены угрозы и риски, разработаны шаги по нивелированию их воздействия Все дефекты должны быть устранены либо должны быть приняты компенсационные меры
Тестирование продукта	Acceptance Testing Динамический анализ кода Функциональное тестирование	Все устраненные дефекты проверены Произведен анализ причин возникновения дефектов с приоритетом Medium и выше
Развертывание продукта	Оценка конфигурации деплоймента Финальный анализ продукта	Формальная приемка продукта в процессе финального анализа продукта