DevOps

Что такое DevOps

DevOps – это технологическая структура, которая обеспечивает взаимодействие между командами разработчиков и операционными командами для более быстрого развертывания кода в производственных средах с возможностью повторения действий и автоматизации. Слово «DevOps» является объединением слов «разработка» (development) и «операции» (operations). DevOps помогает увеличить скорость доставки приложений и услуг. Это позволяет организациям эффективно обслуживать своих клиентов и становиться более конкурентоспособными на рынке. Проще говоря, DevOps — это согласованность между разработкой и ИТ-операциями с более эффективным взаимодействием и сотрудничеством.

DevOps предполагает такую культуру, при которой сотрудничество между командами разработчиков, операторами и бизнес-командами считается критически важным аспектом. Речь идет не только об инструментах, поскольку DevOps в организации постоянно приносит пользу и клиентам. Инструменты являются одним из его столпов, наряду с людьми и процессами. DevOps увеличивает возможности организаций по предоставлению высококачественных решений в кратчайшие сроки. Также DevOps автоматизирует все процессы, от сборки до развертывания, приложения или продукта.

DevOps — это инженер, который следит, чтобы код собирался быстро и не было отказов. Также он строит вокруг всего этого правильную инфраструктуру, например, прописывает, откуда берутся артефакты и куда уходят docker images. Еще DevOps пишет правила деплоя в Kubernetes. В общем, он делает работу более гибкой, быстрой и удобной.

Преимущества DevOps:

Скорость. Работайте с высокой скоростью, чтобы быстрее внедрять новые возможности для клиентов, лучше адаптироваться к меняющимся рынкам и эффективнее достигать намеченных целей в бизнесе. Модель DevOps поможет вашим группам разработки и эксплуатации достичь всех этих целей. Например, микросервисы и непрерывная доставка позволяют группам быстрее взять сервисы под контроль, а затем оперативно обновлять их.

Быстрая доставка. Увеличивайте частоту и скорость релизов, чтобы быстрее обновлять и улучшать свой продукт. Чем быстрее вы выпускаете новые возможности и исправления, тем оперативнее можно реагировать на потребности клиентов и создавать конкурентные преимущества. Непрерывная интеграция и непрерывная доставка помогают автоматизировать процесс выпуска программного обеспечения – от сборки до развертывания.

Надежность. Контролируйте качество обновлений приложений и изменений инфраструктуры, чтобы надежно и быстро разрабатывать продукты, а также сохранять лояльность конечных пользователей. Методы непрерывной интеграции и непрерывной доставки помогают протестировать функциональность и безопасность каждого изменения. А мониторинг и ведение журналов позволяют следить за производительностью в режиме реального времени.

Масштабирование. Управляйте процессами разработки и поддержки инфраструктуры, а также обеспечивайте их стабильную работу при любом масштабе. Автоматизация и последовательность помогут управлять сложными или изменяющимися системами эффективно, сокращая при этом риски. Так, инфраструктура как код способствует более

эффективному управлению средами разработки, тестирования и производства, а также обеспечивает их воспроизводимость.

Оптимизированная совместная работа. Создавайте более эффективные группы в рамках культурной модели DevOps, которая превозносит такие ценности, как сопричастность и ответственность. Группы разработки и эксплуатации тесно взаимодействуют между собой, разделяют большинство обязанностей и объединяют свои рабочие процессы. Это сокращает нерациональные действия и экономит время (например, уменьшает время передачи дел от разработчиков инженерам по эксплуатации и устраняет необходимость написания кода с учетом среды, в которой он будет запущен).

Безопасность. Развивайтесь быстро, сохраняя контроль и соблюдая все требования. Модель DevOps можно внедрить без ущерба для безопасности с помощью автоматизированной политики соблюдения требований, точной настройки, а также методик управления конфигурациями. Например, используя инфраструктуру как код и политику как код, можно определить требования, а затем отслеживать их соблюдение при любом масштабе.

Жизненный цикл. Чтобы DevOps работал, нужно наладить непрерывную связь — конвейер между разработчиками, тестировщиками и администраторами. Для этого нужны инструменты автоматизации, которые помогут эффективнее передавать код, тестировать его и развертывать на серверах.

Разберем процесс разработки приложений по подходу DevOps, его по этапам:

1. **Формулирование требований и проектирование.**

Менеджер проекта описывает, чего бизнес ожидает от приложения, а команда разработки создает структуру будущего продукта и расписывает этапы его создания. В проектировании принимают участие в том числе программисты, тестировщики и администраторы: они лучше понимают время разработки и шаги, на которые ее нужно разделить.

2. **Разработка**

Команда, обычно под руководством DevOps-инженера, создает среду и конвейер CI/CD, в которых будет происходить разработка продуктов. Для этого пишется ряд скриптов и систем для версирования, управления проектом, мониторинга, а также настраиваются кластеры для разработки, тестирования и продакшена. Этим, как правило, занимаются администраторы и тестировщики. Если в компании прижилась культура DevOps, все эти системы будут уже готовы — их нужно только адаптировать под новый продукт.

3. **Запуск конвейера CI/CD**

Когда часть кода готова, разработчики запускают скрипты, подготовленные и автоматизированные на прошлом шаге. Эти скрипты превращают код в продукт и берут на себя

рутину. Например, компилируют код в пакеты, управляют версиями, передают его тестировщикам и администраторам.

DevOps и стратегия тестирования

DevOps-культура поощряет частые релизы. Частые релизы — это страховка от поломки вашего приложения после непоправимых улучшений разработчиками, так как за раз вы будете вывозить меньше изменений. Частые релизы требуют больше QA-работы.

DevOps-культура предлагает свести к минимуму ручную QA-работу, чтобы иметь возможность релизиться как можно чаще, что безопаснее и стабильнее. В DevOps-культуре роль QA-инженеров смещается от тестеров к людям, которые следят за качеством проекта и помогают разработчикам в написании автоматических тестов, вырабатывают стратегию тестирования.

Стратегия тестирования для DevOps не должна звучать: «Мы используем Protractor и Jenkins». Это не план. Это всего лишь перечисление инструментов, которые используются вами. Главное вопросы, на которые должен отвечать план, — почему и зачем.

Мы должны ответить себе на следующие вопросы:

- Какие ресурсы у нас обязательны к тестированию?
- Каков будет результат успешного прохождения тестов? Какую информацию нам надо получить для успешной поставки или что может остановить поставку и заставить нас начать что-то поправлять?
- Риски. Как наши тесты уменьшают их? И как это будет представлено команде?
- Расписание. Когда мы будем прогонять те или иные виды тестов?
- Кто будет отвечать за автоматизированные тесты? Кто будет их разрабатывать? Кто будет конечным получателем результатов тестов?
- Что будет триггерить старт тестов? Когда команде ожидать начала тестирования?

Первое, с чего мы должны начать, — это внедрить концепцию TDD. Сразу скажу, что TDD не является абсолютно правильным выбором, но оно способствует написанию тестов как таковых. То есть разработчики должны писать тесты еще до начала кодирования.
У разработчиков любое действие сохранения должно триггерить запуск юнит-тестов. Кроме того, инженер не должен иметь возможности залить свой код с поломанными тестами или когда тестовое покрытие падает ниже какого-то предела.

При пуше кода в фича бранч должны прогоняться сьюты integration- и unit-тестов. Они определяют статус билда и возможность смерджить его в главную dev-ветку. При поломанном билде не должно быть возможности мерджа, а коммитер должен быть оповещен, что его коммит сломал ветку. Далее идет мердж в master или stage brunch и сборка там. При этом требуется прогонять все тестовые наборы, включая E2E.

Этот тестовый набор определяет, может ли осуществляться поставка вашего продукта.
В случае фейла все члены dev-команды должны быть оповещены о том, что поставка отложена, и о причине, вызвавшей данную ошибку.

Если все наборы на тестовых окружениях прошли нормально, осуществляется поставка продукта на продакшен и снова прогон E2E тестовых наборов уже на продакшен окружении. В дополнение к вышеперечисленным тестовым наборам на продакшене должен быть осуществлен прогон performance, penetration и других нефункциональных тестов. Если что-то пошло не так — осуществляется откат и оповещение всех задействованных в процессе деплоймента и разработки. Причем заметьте, что откат должен быть тоже протестирован E2E и другими тестами.

Роль тестировщика в процессе непрерывной поставки

Задача тестировщика – создание тестовых наборов для каждого этапа DevOps в том числе – для инсталяционного и smoke-тестирования:

- **Непрерывное тестирование**

Написанный код с помощью скриптов уходит на автоматическое тестирование. Оно происходит без участия программистов и тестировщиков и помогает выявить ошибки при внесении изменений и выгрузке кода. Если ошибки есть, код не уйдет в сборку и точно не попадет в работающий продукт.
Например, в приложение внесли какую-то новую функцию. Оказалось, что она ломает другую функцию, на первый взгляд даже не связанную с новой. Автоматический тест это выявит и сразу сообщит об ошибке — код даже не придется тестировать вручную.
Если изменения кода незначительные, после автотестов он сразу уходит на рабочие серверы. Если серьезные, код отправляется тестировщикам, чтобы они проверили пользовательские сценарии и убедились, что все работает в соответствии с требованиями к продукту.
Разработчики, тестировщики и администраторы работают на виртуальных машинах или серверах с разной конфигурацией. Если провести тестирование (даже автоматическое) на конкретной машине, то на другой код может не заработать. Чтобы избежать этой проблемы, нужна контейнеризация — запуск и тестирование приложения в определенной фиксированной среде. Инструменты DevOps позволяют автоматически запускать такие среды, менять их конфигурации, проводить тесты и доставлять контейнеры с готовым протестированным кодом на рабочие серверы.

- **Непрерывное развертывание**

Когда конфигурации протестированы, автоматические скрипты сразу отправляют их развертываться на «боевых» серверах. В итоге выпуск ПО или обновления в релиз перестает быть выдающимся событием и превращается в рутину. Если в какой-то версии были ошибки, можно исправлять их весь день и в течение дня постоянно развертывать мелкие обновления, которые постепенно нормализуют работу приложения.
Чтобы эта система работала, важно тщательное тестирование. Если плохо протестированное приложение автоматически развернется, это может привести к серьезным финансовым потерям.

- **Непрерывный мониторинг**

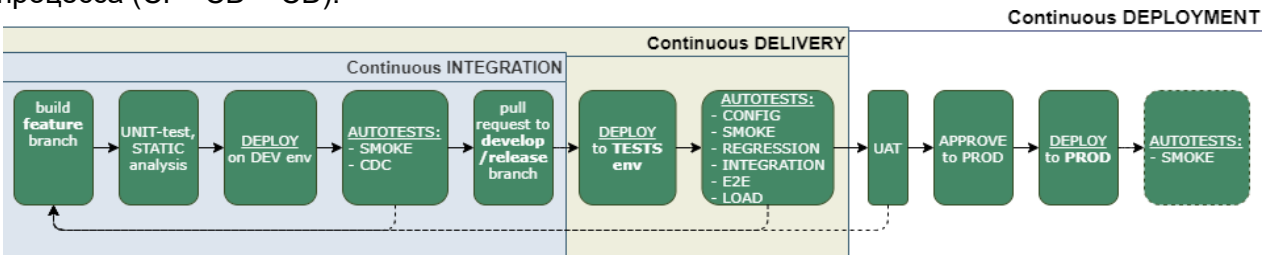
Когда приложение ушло на рабочий сервер, к нему подключаются системы мониторинга. Они контролируют, как работает приложение, записывают все ошибки в логи, оповещают о проблемах и автоматически перезагружают и отключают сломанные функции.
Например, в приложении происходит какой-то сбой. Система мониторинга его фиксирует, записывает в лог информацию о сбое и отправляет команду на перезагрузку. А потом посылает информацию о сбое разработчикам, чтобы они могли быстро исправить ошибку.

Важно, что в DevOps все эти этапы не идут друг за другом, а параллельно. Пока программисты работают над одним кодом, другую его часть уже тестируют, а еще одну мониторят. А администраторы в это же время собирают результаты мониторинга — и в этот же момент формируют вместе с программистами новые задачи на разработку.

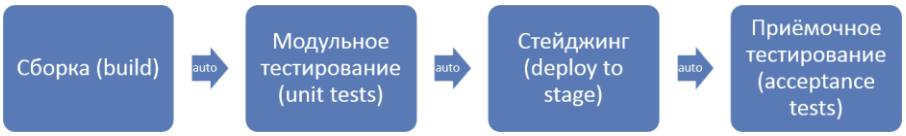
CI/CD/CDP

Непрерывная интеграция (CI), непрерывная доставка (CD) и непрерывное развёртывание (CD) — devops-подход к разработке и апгрейду программного обеспечения, подразумевающий непрерывное, конвейерное тестирование, сборку, доставку и развёртывание обновлений. Возможно как отдельное применение компонентов этого

подхода (CI или CI + CD), так и их последовательное использование в рамках единого процесса (CI + CD + CD).



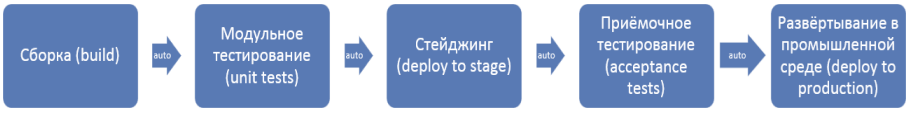
Непрерывная интеграция (CI (Continuous integration) - первичный, базовый процесс обновления ПО, в рамках которого все изменения на уровне кода вносятся в единый центральный репозиторий. Такое внесение принято называть слиянием. После каждого слияния (которое проходит по несколько раз в день) в изменяемой системе происходит автоматическая сборка (часто приложение упаковывается в Docker) и тестирование (проверка конкретных модулей кода, UI, производительности, надежности API). Таким образом разработчики страхуются от слишком поздних обнаружений проблем в обновлениях. Каждый этап запускается и выполняется автоматически.



Непрерывная поставка (CD (Continuous delivery) — следующий после CI уровень. Теперь новая версия не только создается и тестируется при каждом изменении кода, регистрируемом в репозитории, но и может быть оперативно запущена по одному нажатию кнопки развертывания. Однако запуск развертывания все еще происходит вручную — ту самую кнопку все же надо кому-то нажать. Этот метод позволяет выпускать изменения небольшими партиями, которые легко изменить или устранить в случае необходимости. Развертывание выполняется автоматически, но запускается вручную.

Непрерывное развертывание (CDP (Continuous deployment) — после автоматизации релиза остается один ручной этап: одобрение и запуск развертывания в продакшен. Практика непрерывного развертывания упраздняет и это, не требуя непосредственного утверждения со стороны разработчика. Все изменения развертываются автоматически. Все этапы запускаются и выполняются автоматически.

Непрерывная интеграция (CI), непрерывная доставка (CD) и непрерывное развёртывание (CD) — devops-подход к разработке и апгрейду программного обеспечения, подразумевающий непрерывное, конвейерное тестирование, сборку, доставку и развёртывание обновлений. Возможно как отдельное применение компонентов этого подхода (CI или CI + CD), так и их последовательное использование в рамках единого процесса (CI + CD + CD).



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

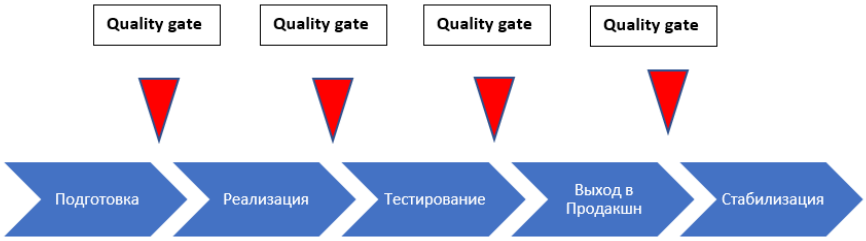
- Проверка изменений в репозитории;
- Статического анализа кода: на уязвимости, на соответствие требованиям истандартам;
- Компиляция и формирование билда (сборки);
- Передачи обратной связи об успехе/неудаче билда в виде уведомлений повыбранным каналам;
- Развертка на dev/test/staging/prod;
- Всех необходимых тестов (unit, smoke, acceptance, regression, integration,end-to-end);
- Заведения уязвимостей в СУП (Стандарт Управления Проектами), в случаеошибок;
- Слияния изменений с нужной веткой в случае "зелёных" тестов.

Quality gates:

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. Это автоматические проверки качества, которые устанавливаютпороговые значения для продвижения продукта по конвейеру разработки.

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.

Принцип Quality Gates - Помогает решать проблемы в коде на ранних этапах, до того, как он обрастёт зависимостями. Если в коде есть дублирование, обнаруживаются проблемы с переменными или не хватает тестов, он не «проходит в ворота» и возвращается автору.

В результате код становится чище и понятнее, баги оказывается проще исправлять, да и появляются они реже.