DevOps-практики

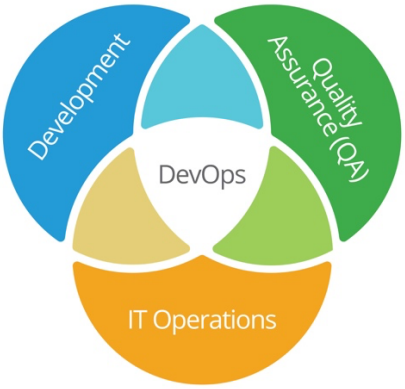
DevOps (**DEVelopment OPeration**) – это набор практик для повышения эффективности процессов разработки (**Development**) и эксплуатации (**Operation**) программного обеспечения (ПО) за счет их непрерывной интеграции и активного взаимодействия профильных специалистов с помощью инструментов автоматизации. DevOps позиционируется как Agile-подход для устранения организационных и временных барьеров между командами разработчиков и других участников жизненного цикла ПО (тестировщиками, администраторами, техподдержкой), чтобы они могли быстрее и надежнее собирать, тестировать и выпускать релизы программных продуктов.

История появления

Термин “DevOps” был популяризован серией встреч “DevOps Days”, прошедших в 2009 году в Бельгии. Одной из наиболее важных теоретических работ по DevOps считается книга Патрика Дюбуа, Джина Ким, Джеза Хамбл и Джона Уиллис “Руководство по DevOps. Как добиться гибкости, надежности и безопасности мирового уровня в технологических компаниях”, впервые опубликованная на английском языке в 2016 году. К этому основателей нескольких софтверных компаний и независимых ИТ-консультантов подтолкнул накопленный опыт работы в крупных проектах.

Однако само понятие DevOps зародилось в начале 2000-х годов, когда в ИТ-мире больших корпораций возникла проблема рассогласования рабочих процессов, при которой нормальная работа программного продукта нарушена из-за функционального и организационного разделения тех, кто пишет код, и тех, кто выполняет его развертывание и поддержку. У разработчиков и специалистов по эксплуатации продукта часто бывают разные и даже противоречащие друг другу цели, руководители подразделений и ключевые показатели эффективности. Рабочие места разнопрофильных участников жизненного цикла ПО зачастую располагаются в разных локациях. Такая разрозненность и нарушение коммуникации внутри компании приводит к удлинению сроков решения задач, сверхурочной работе, сорванным релизам и недовольству клиентов.

Концепция DevOps предлагает решать эту проблему с помощью приложения принципов Agile не только к разработке и тестированию, но и к процессам эксплуатации ПО, т.е. к развертыванию и поддержке. Таким образом, популярность DevOps возникла, в том числе благодаря распространению Agile-практик, ориентированных на ускорение процессов поставки готового продукта и увеличение количества выпускаемых версий. Кроме того, дополнительным драйвером развития DevOps стала микросервисная архитектура, когда система состоит из набора отдельных слабосвязанных модулей, реализация каждого из которых находится в зоне ответственности одного человека, который разрабатывает, тестирует и развертывает ПО. Благодаря небольшому размеру каждого модуля (сервиса), его архитектура может создаваться путем непрерывного рефакторинга, что уменьшает трудоемкость предварительного проектирования и позволяет постоянно выпускать новые релизы программного продукта



Что такое Agile:

Agile – система, основанная на принципе “гибкого” управления проектами. Сюда относят методики Scrum, FDD, Kanban, Экстремальное программирование (XP), Lean и т.д. Ключевая особенность такого подхода - создание проекта в несколько циклов (итераций), в конце каждого виден конкретный результат, который позволяет понять, по какому пути двигаться дальше.
Гибкие методологии строятся на принципе итераций. Создание нового продукта делится на несколько циклов от одной недели до месяца. В зависимости от особенностей проекта, временные рамки оговариваются отдельно. Каждый цикл представляет собой завершенный мини-проект, в котором есть этапы анализа, планирования, тестирования и реализации. В итоге клиент получает продукт, который, при необходимости, корректируется.

Основополагающие принципы Agile-манифеста

- Высшим приоритетом для нас является удовлетворение потребностей заказчика благодаря регулярной и ранней поставке ценного программного обеспечения.
- Изменение требований приветствуется, даже на поздних стадиях разработки. Agile-процессы позволяют использовать изменения для обеспечения заказчику конкурентного преимущества.
- Работающий продукт следует выпускать как можно чаще, с периодичностью от пары недель до пары месяцев.
- На протяжении всего проекта разработчики и представители бизнеса должны ежедневно работать вместе.
- Над проектом должны работать мотивированные профессионалы. Чтобы работа была сделана, создайте условия, обеспечьте поддержку и полностью доверьтесь им.
- Непосредственное общение является наиболее практичным и эффективным способом обмена информацией как с самой командой, так и внутри команды.
- Работающий продукт - основной показатель прогресса.
- Инвесторы, разработчики и пользователи должны иметь возможность поддерживать ритм бесконечно. Agile помогает наладить такой устойчивый процесс разработки.
- Постоянное внимание к техническому совершенству и качеству проектирования повышает гибкость проекта.
- Простота - искусство минимизации лишней работы - крайне необходима.
- Лучшие требования, архитектурные и технические решения рождаются у самоорганизующихся команд.
- Команда должна систематически анализировать возможные способы улучшения эффективности и соответственно корректировать стиль своей работы.

Основные преимущества и недостатки

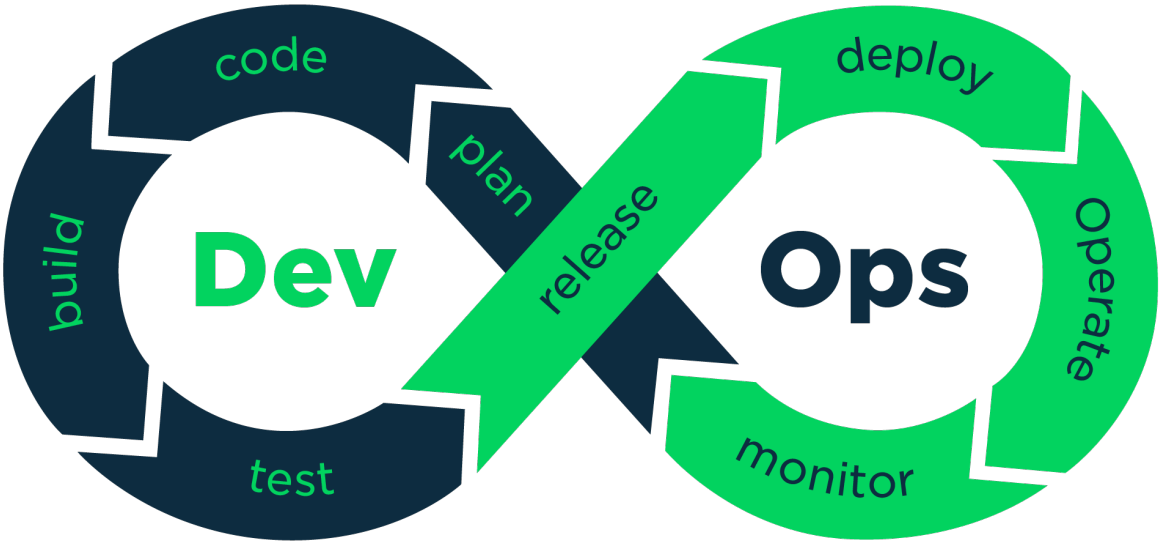
Преимущества:

- Внесение необходимых изменений и внедрение нового функционала может происходить независимо от цикла разработки продукта, что значительно повышает конкурентные преимущества готового проекта.
- Проект состоит из коротких и понятных циклов, по окончании которых клиент получает рабочий продукт.
- Гибкий процесс корректировок в любой итерации позволяет снизить производственные риски. - Довольно быстрый релиз пробной версии для дальнейших корректировок и тестирования.
- Высокая степень вовлеченности всех членов команды и постоянное взаимодействие с заказчиком. Он всегда в курсе, на какой стадии находится проект.
- Показателем эффективности является рабочий продукт, что требует высокого профессионализма от исполнителей и грамотной организации рабочего процесса.

Недостатки:

- Рассчитать конечные затраты практически невозможно – требования могут постоянно меняться в зависимости от особенностей проекта. Сложность заключается в том, что они могут противоречить существующей структуре.
- Agile требует большой вовлеченности в процесс и полному погружению в него, что бывает сложно, особенно для молодых подрядчиков.
- Возможность частого внесения правок может обернуться риском в бесконечном совершенствовании проекта. Здесь также возможна и обратная сторона – снижение качества продукта.

Процессы и объекты DevOps



DevOps, как и другие Agile-практики, ориентирован на командную работу, где рассматриваются все аспекты жизненного цикла ПО, от программного кода до эксплуатации продукта конечным пользователем:

1. Code (Код) – разработка и анализ, контроль версий и слияния кода;
2. Build (Сборка) – непрерывная интеграция различных сборок;
3. Test (Тест) – непрерывное тестирование, обеспечивающее обратную связь по бизнес-рискам;
4. Operate (Работа с пакетами) – репозиторий артефактов, предварительная установка приложения;
5. Release (Выпуск) – управление изменениями, официальное утверждение выпуска, автоматизация выпуска;
6. Deploy (Развертывание конфигурации) – управление инфраструктурой как кодом;
7. Monitor (Мониторинг) – мониторинг производительности приложений, опыт работы с конечным пользователем.

Цели и задачи DevOps

Поскольку процессы devops охватывают весь цикл поставки ПО, выделяют несколько **основных целей этого подхода**:

- сокращение времени для выхода на рынок;
- снижение частоты отказов новых релизов;
- сокращение времени выполнения исправлений;
- уменьшение количества времени на восстановления при сбое новой версии или других случаях отключения текущей системы.

Эти цели достигаются через решение следующих задач:

- согласование процессов разработки и поставки ПО с эксплуатацией;
- автоматизация процессов разработки, тестирования и развертывания;
- непрерывное тестирование качества приложений;
- управление ИТ-инфраструктурой как кодом;

- управление изменениями;
- непрерывный мониторинг производительности приложений и состояния инфраструктуры.

Таким образом, DevOps нацелен на предсказуемость, эффективность, безопасность и ремонтопригодность операционных процессов, а также регулярную поставку надежно работающего продукта, его обновлений и обслуживания.

Главные принципы DevOps

Рассматривая DevOps как масштабирование Agile-подхода на весь процесс разработки, внедрения и сопровождение ПО, можно выделить **5 основных принципов** (CALMS) его реализации с целью увеличения частоты релизов и повышения ответственности команды за продукт:

- **Культура (Culture)** – кросс-функциональное сотрудничество разнопрофильных специалистов и команд за счет единого информационного пространства проектного контента, открытых каналов коммуникаций и постоянного общения всех участников;
- **Автоматизация (Automatization)** – использование инструментов непрерывной поставки с прогоном каждой правки кода через серию автоматизированных тестов, часто использующих облачную инфраструктуру, и последующую упаковку успешных сборок с дальнейшим перемещением на рабочий сервер с помощью автоматизированных развертываний и управления инфраструктурой как кодом через конфигурации саморазвертываемых сред;
- **Бережливость (Lean)** – устранение действий с низкой полезностью и ускорение процессов, непрерывное совершенствование через регулярный ретроспективный анализ, раздельное тестирование различных инструментов, принятие поражений, возможности быстрого обнаружения проблем и их незамедлительного решения;
- **Измерения (Measurement)** производительности, например продолжительность работы пользователей с продуктом, частота появления в логах сообщений о критических ошибках – необходимы ясные и четкие критерии оценки работы, показатели эффективности процессов;
- **Обмен (Sharing)** – совместная ответственность и разделение успехов, выпуск и обеспечение работы приложения осуществляются теми же людьми, что выполняли его сборку, т.е. разработчики (Developers) и операторы (Operators) взаимодействуют на каждом этапе жизненного цикла приложения.

Преимущества DevOps

Как мы помним, методология DevOps позволила связать в единый процесс разработку, тестирование и эксплуатацию. Что, в итоге, привело к тому, что программное обеспечение стало обновляться чаще без потери в качестве.

1. Отсюда вытекает первый и самый жирный плюс – DevOps повышает эффективность и конкурентоспособность бизнеса. Ведь чем быстрей приложение доходит до конечного потребителя, тем быстрей бизнес начинает зарабатывать. А чем оперативней реализуются новые возможности и исправляются ошибки, тем более лояльных клиентов получает компания.
2. Автоматизация цикла “разработка-тестирование-эксплуатация” позволяет ускорить процесс принятия и реализации решений, касающихся разработки ПО.

3. Использование инструментов DevOps помогает быстро вносить в продукт необходимые изменения.
4. Кроме того, автоматизация и стандартизация помогают упростить и ускорить рутину, навести порядок в “хаосе” ИТ-проектов. Разработчики в этом случае могут сфокусироваться на улучшении кода. Снижается “человеческий” фактор при возникновении ошибок.
5. Методики и инструменты DevOps помогают поддерживать систему в стабильном состоянии и, в случае ошибки, сократить время на ее восстановление.
6. Благодаря метрикам и средствам мониторинга, DevOps-инженер получает максимально быструю обратную связь как о состоянии системы, так и о ее функциональности от пользователей.
7. Так как немаловажным принципом культуры DevOps является практика постоянного взаимодействия отделов разработки и внедрения, то и коммуникации в этом случае становятся проще и оперативней.
8. Кроме того, принцип общей ответственности за результат ведет к тому, что не возникает постоянного перекладывания вины за ошибки и поломки с одного отдела на другой. Как это часто бывает, когда локально у разработчиков код работает, а при запуске в prod возникают ошибки. Разработчики винят службу эксплуатации, системные администраторы считают, что проблема в коде. Порочный круг замкнулся. А плюсы DevOps помогают его разорвать и прийти к продуктивному взаимодействию.
9. Еще один важный положительный момент работы с методиками DevOps заключается в том, что они побуждают постоянно учиться новому – меняются инструменты и подходы, возникают новые требования к ПО. И этот процесс постоянного самосовершенствования повышает ценность DevOps-инженера как специалиста.

Недостатки DevOps

Минусы DevOps (хотя, на наш взгляд, часть пунктов здесь можно назвать скорей особенностями) тоже определяются самой спецификой методологии.

1. К примеру, DevOps нужен не всем. Скажем, для маленькой компании, ведущей небольшой проект, затраты на инструменты DevOps и дорогостоящих специалистов совсем ни к чему. Перед внедрением принципов DevOps в работу компании, всегда стоит оценить соотношение затрат и ожидаемой эффективности. Если затраты, как трудовые, так и денежные, слишком высоки для компании, то переход на новую методологию может не ускорить, а наоборот, замедлить процессы разработки и внедрения.
2. DevOps-инженер – это высококвалифицированный и высокооплачиваемый сотрудник. Он должен знать очень широкий стек технологий и постоянно совершенствовать знания. Он должен иметь ряд софтскилловых навыков – например, уметь находить общий язык с разными людьми, чтобы собрать с них фидбек и наладить эффективное взаимодействие. А значит, и требования к такому сотруднику высоки и найти специалиста хорошего уровня не так просто. Не говоря уже о том, что DevOps входит в число самых высокооплачиваемых ИТ-специализаций.
3. Есть и обратная проблема с квалификацией DevOps-инженеров. Зачастую специалисты считают, что им нужны только самые “модные” инструменты. Они рассуждают про Docker и Kubernetes, часто поверхностно разбираются в самых актуальных технологиях, но при этом совсем не знают базовых вещей – как работают базы данных и окружение, как настроить сети. А без этого “фундамента” стать действительно квалифицированным DevOps-инженером невозможно. Кроме того, это может вылиться в ситуацию, когда специалист будет использовать избыточный инструментарий (скажем, Kubernetes, потому что он “на слуху”), вместо того, чтобы обратиться к более простым и подходящим ситуации инструментам и best practices.
4. Для эффективного функционирования DevOps нужен эффективный менеджмент – как раз для того, чтобы разные отделы успешно взаимодействовали и понимали важность общей цели.
5. Для внедрения DevOps нужно не только поменять процессы и подключить новые инструменты, но и в целом поменять культуру, что достаточно трудоемко. Без соблюдения основных принципов (скажем, автоматизации, ведения документации, постоянного

быстрого фидбека, соблюдения культуры взаимодействия подразделений) DevOps невозможен.

DevOps-конвейеры и инструментарий

Что такое CI/CD

CI/CD - одна из практик DevOps, подразумевающая непрерывную интеграцию и доставку. Этот набор принципов предназначен для повышения удобства, частоты и надежности развертывания изменений программного обеспечения или продукта. CI/CD относится к agile-практикам и позволяет разработчикам уделять внимание реализации бизнес-требований, качеству кода и безопасности продукта.
Цели CI/CD:

- обеспечение последовательного и автоматизированного способа сборки, упаковки и тестирования продуктов или приложений;
- автоматизация развертывания в разных окружениях;
- сведение к минимуму ошибок и проблем.

Принципы CI/CD

Для CI/CD существуют четыре руководящих принципа:

- Разделение ответственности. Каждый из участников процесса делит ответственность за те или иные этапы жизненного цикла продукта. Проектируется бизнес-логистика, внедряются сквозные функции, проводятся приемочные тесты и организуется логистика кода.
- Снижение рисков. Каждая команда, участвующая в разработке продукта, стремится к снижению рисков — контролируется корректность бизнес-логистики, проверяется пользовательский опыт, улучшается хранение и обработка данных и прочее.
- Сокращение цикла обратной связи. Разработчик и клиент должны стремиться к увеличению скорости внесения изменений и согласования правок. Сборку и тестирование кода можно автоматизировать. А для ситуаций, когда требуется участие человека, можно минимизировать число информационных посредников.
- Реализация среды. У разработчиков должно быть общее рабочее пространство с основной и вспомогательными ветками для контроля версий и качества, приемлемости, отказоустойчивости и других критериев.

Этапы CI/CD

Методология CI/CD подразумевает разделение процесса разработки на семь этапов:

1. Написание кода. Разработчики пишут код своего модуля и проводят тестирование в ручном режиме. После этого результат работы соединяется в главной ветке с текущей версией проекта. После того, как в главной ветке публикуются все коды модулей, начинается второй этап.
2. Сборка. Выбранная система контроля версий инициирует автоматическую сборку и последующее тестирование проекта. Триггеры для активации сборки могут быть настроены самостоятельно. Для автоматизации сборки применяется Jenkins или другой инструмент.
3. Ручное тестирование. После проверки CI-системой работоспособности тестовой версии код передается для ручного исследования.
4. Релиз. После ручного тестирования в сборку вносятся исправления. Следом проходит релиз версии кода для клиентов.
5. Развертывание. На этом этапе текущая (рабочая) версия кода размещается на production-серверах разработчика. Клиент может взаимодействовать с программой и изучать ее функции.
6. Поддержка и мониторинг. Продукт начинает использоваться конечными пользователями. При этом разработчики продолжают его поддерживать и проводят анализ пользовательского опыта.

7. Планирование. Исходя из пользовательского опыта разрабатывается новый функционал и готовится план доработок. После этого разработчик начинает написание кода — и цикл замыкается.

Плюсы и минусы CI/CD

У методологии Continuous integration & Continuous delivery есть особенности, определяющие её преимущества и недостатки.

Плюсы:

1. Минимальное время от запроса клиента до запуска в использование.
2. 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.
3. Возможность проверки вариантов. Оперативное тестирование и много итераций помогают разработчику быстро выявлять варианты, не имеющие перспектив, еще на начальных этапах.
4. Качество результата. Проведение автоматического тестирования помогает выявить ошибки и другие проблемы на самых ранних этапах разработки. При стандартном релизном подходе это сделать сложно или невозможно.

Минусы:

1. Требования к опыту. В теории все корпоративные ИТ-системы можно перевести на CI/CD. Но на практике для получения результата нужен первичный опыт работы с методологией, а также правильная организация перестраивания всех процессов.
2. Сложность обеспечения взаимодействия. Непрерывное обновление и непрерывная поставка должны быть четко скоординированы, что возможно только после тщательной настройки взаимодействия между специалистами всех уровней.

Принципы непрерывной поставки

Принципы CI/CD тесно связаны с DevOps - особой культурой менеджмента, которая объединяет разработчиков, тестировщиков и поддержку (operations) в единую команду для общей цели — быстрой доставки ПО пользователю. DevOps-инженер собирает проект в единое целое, контролируя процесс разработки и сборки кода, использование правильных технологий и взаимодействие команды.

В рамках DevOps процесс разработки построен по типу конвейера (pipeline). Конвейеризация здесь подразумевает создание единой, непрерывной и максимально автоматизированной цепочки сборки. При этом методология CI/CD непосредственно реализуется с помощью программных инструментов вроде упомянутого GitLab в рамках трех основных этапов:

- Написание кода;
- Тестирование кода (юнит-, UI- и E2E-тесты);
- Развертывание его в производственной среде в виде готового продукта.

У конвейеризации в разработке те же преимущества, что и в любой другой отрасли. Здесь каждый участник процесса понимает границы собственной ответственности и полный стек задач. При должном спросе можно наращивать объем «производства» и повышать качество продукта.

Здесь можно возразить: продукты, собранные вручную, как правило, более качественные. Но команд, которые создают штучные продукты, мало, и рынок ими быстро не завоюешь. Когда дело доходит до масштаба, каждый сэкономленный час выливается в миллионную экономию на проекте.