**Что такое DevOps?**

DevOps — это набор [методик](https://www.atlassian.com/ru/devops/what-is-devops/devops-best-practices), [инструментов](https://www.atlassian.com/ru/devops/devops-tools/choose-devops-tools) и [философия культуры](https://www.atlassian.com/ru/devops/what-is-devops/devops-culture), которые позволяют автоматизировать и интегрировать между собой процессы команд разработки ПО и ИТ‑команд. Особое внимание в DevOps уделяется расширению возможностей команд, их взаимодействию и сотрудничеству, а также автоматизации технологий.

Движение DevOps [возникло примерно в 2007 году](https://www.atlassian.com/ru/devops/what-is-devops/history-of-devops), когда сообщества разработчиков программного обеспечения и ИТ-специалистов заговорили о проблемах в традиционной модели разработки программного обеспечения. В соответствии с этой моделью разработчики, писавшие код, работали независимо от инженеров по эксплуатации, которые занимались развертыванием и поддержкой этого кода. Термин DevOps является комбинацией двух слов — development (разработка) и operations (эксплуатация) — и отражает процесс объединения этих практик в один непрерывный процесс.

**Как работает DevOps?**

В состав команды DevOps входят разработчики и ИТ‑специалисты, которые работают совместно на протяжении всего жизненного цикла продукта, чтобы повысить скорость и качество развертывания ПО. Это новый стиль работы и важное изменение культуры, имеющее значительные последствия для команд и организации в целом.

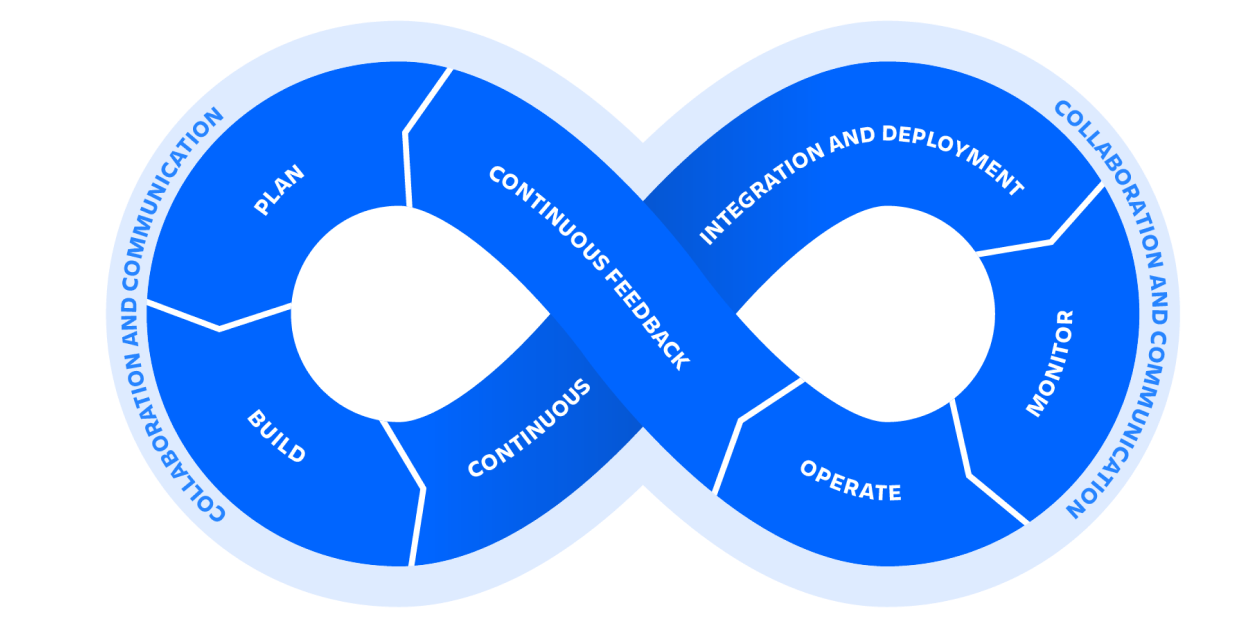
В модели DevOps команды по разработке и эксплуатации больше не являются разобщенными. Иногда эти две команды объединяются в одну, причем ее участники работают на протяжении всего жизненного цикла приложения — от разработки и тестирования до развертывания и эксплуатации — и обладают многосторонними профессиональными навыками.

Команды DevOps используют инструменты для автоматизации и ускорения процессов, благодаря чему повышается надежность. Пакет инструментов DevOps позволяет командам решить ключевые задачи DevOps: реализовать непрерывную интеграцию, непрерывную поставку, автоматизацию и совместную работу.

Ценности DevOps иногда используются не только в командах разработчиков. Когда подход DevOps применяют команды по обеспечению безопасности, безопасность становится активной и неотъемлемой частью процесса разработки. Такой подход называют [DevSecOps](https://www.atlassian.com/ru/continuous-delivery/principles/devsecops).

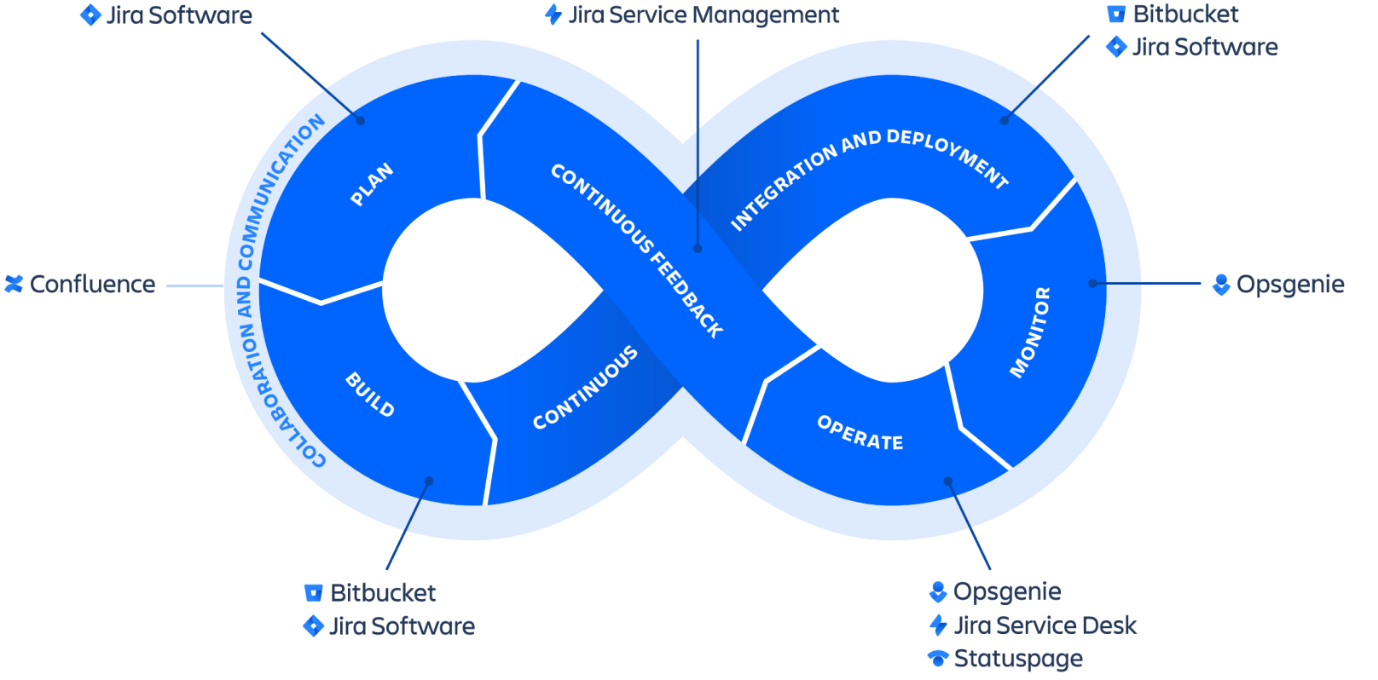
**Жизненный цикл DevOps**

Непрерывный характер DevOps приводит к тому, что специалисты используют бесконечный цикл, чтобы показать взаимосвязь фаз жизненного цикла DevOps. Несмотря на внешнюю последовательность цикла, он символизирует необходимость постоянного сотрудничества и итеративного совершенствования на протяжении всего жизненного цикла.



Жизненный цикл DevOps состоит из шести этапов, причем в левой части бесконечного цикла представлены процессы, возможности и инструменты, необходимые для разработки, а в правой — для эксплуатации. На каждом этапе команды ведут совместную работу и взаимодействуют для поддержания согласованности, скорости и качества.

**Инструменты DevOps**



[Инструменты DevOps](https://www.atlassian.com/ru/devops/devops-tools) применяются на всех основных этапах жизненного цикла DevOps. Они расширяют возможности методик DevOps, способствуя эффективной совместной работе, сокращению количества переключений контекста, внедрению автоматизации и созданию возможностей для наблюдения и мониторинга.

Все пакеты инструментов DevOps делятся на два типа: универсальные и открытые. Универсальный пакет инструментов является комплексным решением; обычно его нельзя интегрировать со сторонними инструментами, тогда как открытый пакет можно настраивать и дополнять различными инструментами. У [каждого из этих подходов есть свои плюсы и минусы.](https://www.atlassian.com/ru/devops/devops-tools/choose-devops-tools)

Примером открытого пакета инструментов DevOps является решение Atlassian Open DevOps, которое включает Jira в качестве базового продукта и интегрируется с приложениями ведущих поставщиков и приложениями Marketplace.

[Подробнее об инструментах DevOps](https://www.atlassian.com/ru/devops/devops-tools)

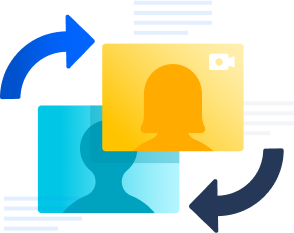
**Каковы преимущества DevOps?**

По данным [опроса Atlassian «Тенденции DevOps» за 2020 год](https://www.atlassian.com/ru/whitepapers/devops-survey-2020), 99 % респондентов считают, что методика DevOps принесла пользу их организациям. К [преимуществам DevOps](https://www.atlassian.com/ru/devops/what-is-devops/benefits-of-devops) относятся: более быстрый и простой выпуск релизов, эффективная работа команды, повышение безопасности и качества продуктов и, как следствие, рост удовлетворенности команд и клиентов.



**Скорость**

Команды, следующие принципам DevOps, выпускают более качественные и стабильные релизы с высокой скоростью. Это подтверждается отчетом DORA [о состоянии DevOps за 2019 год](https://cloud.google.com/blog/products/devops-sre/the-2019-accelerate-state-of-devops-elite-performance-productivity-and-scaling), согласно которому высококлассные команды выполняют развертывания в 208 раз чаще и в 106 раз быстрее, чем команды с низкой эффективностью. Непрерывная поставка позволяет командам создавать, тестировать и поставлять программное обеспечение с помощью автоматизированных инструментов.



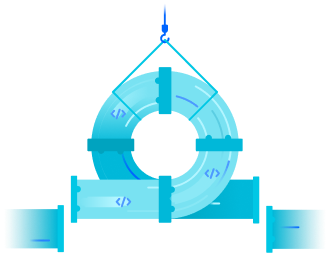
**Более эффективная совместная работа**

Основой DevOps является культура сотрудничества команд по разработке и эксплуатации, которые несут общую ответственность и объединяют свою работу. При этом повышается эффективность работы команд и сокращаются затраты времени на передачу работы и создание кода, который разрабатывается для среды, в которой выполняется.



**Быстрое развертывание**

Благодаря более частому и быстрому выпуску релизов команды DevOps быстро совершенствуют продукты. Быстро выпуская новые возможности и исправляя баги, можно получить конкурентное преимущество.



**Качество и надежность**

Непрерывная интеграция и непрерывная поставка делают изменения эффективными и безопасными и повышают качество программного продукта. Мониторинг позволяет командам получать информацию о производительности в режиме реального времени.



**Безопасность**

После внедрения безопасности в конвейер непрерывной интеграции, непрерывной поставки и непрерывного развертывания [DevSecOps](https://www.atlassian.com/ru/devops/devops-tools/devsecops-tools) становится активной и неотъемлемой частью процесса разработки. Безопасность встраивается в продукт путем внедрения активных проверок и тестирования безопасности в рабочие процессы agile-разработки и DevOps.

**Методики DevOps**

**Непрерывная интеграция**

[Непрерывная интеграция](https://www.atlassian.com/ru/continuous-delivery/continuous-integration) — это методика, направленная на автоматизацию интеграции изменений кода в программный проект. Благодаря ей разработчики могут чаще проводить слияние изменений кода в центральный репозиторий, где выполняются сборка и тестирование. Непрерывная интеграция помогает командам DevOps быстрее устранять баги, повышать качество программного обеспечения и сокращать сроки проверки и выпуска новых обновлений ПО.

[Подробнее о непрерывной интеграции](https://www.atlassian.com/ru/continuous-delivery/continuous-integration)

**Непрерывная поставка**

Непрерывная поставка является продолжением непрерывной интеграции и используется для автоматического развертывания изменений кода в среде тестирования и рабочей среде. Создается [конвейер непрерывной поставки](https://www.atlassian.com/ru/continuous-delivery/principles/pipeline), в котором автоматизированные процессы сборки, тестирования и развертывания организуются в единый процесс выпуска релизов.

[Подробнее о непрерывной поставке](https://www.atlassian.com/ru/continuous-delivery)

**Ситуационная осведомленность**

Очень важно, чтобы каждый участник организации имел доступ к данным, необходимым для максимально быстрого и эффективного выполнения его работы. Участники команды должны получать оповещения о сбоях в конвейере развертывания (включая системные сбои и ошибки тестирования), а также актуальную информацию о работоспособности и производительности приложений, выполняющихся в рабочей среде. Показатели, журналы, трассировки, мониторинг и оповещения — все это важные источники обратной связи, необходимые командам для работы.

[Подробнее о показателях DevOps](https://www.atlassian.com/ru/devops/frameworks/devops-metrics)

**Автоматизация**

Автоматизация является одной из важнейших методик DevOps. Она позволяет командам значительно ускорить процесс разработки и развертывания программного обеспечения высокого качества. Благодаря автоматизации простая отправка изменений кода в репозиторий исходного кода может запустить процесс сборки, тестирования и развертывания, что значительно сокращает затраты времени на выполнение этих действий.

[Подробнее о рекомендациях по автоматизации DevOps](https://www.atlassian.com/ru/solutions/devops/guides/intermediate)

**Инфраструктура как код**

Независимо от того, имеется ли у организации локальный ЦОД или все работы выполняются в облаке, возможность быстро и бесперебойно предоставлять и настраивать инфраструктуру, а также управлять ею имеет решающее значение для успешного внедрения DevOps. [Инфраструктура как код](https://www.atlassian.com/ru/continuous-delivery/principles/infrastructure-as-code) (IaC-обработка) — это не только написание скриптов для различных конфигураций инфраструктуры. С определениями инфраструктуры обращаются как с обычным кодом, то есть используют управление версиями, проверку кода, тестирование и т. д.

[Подробнее об инфраструктуре как коде](https://www.atlassian.com/ru/microservices/cloud-computing/infrastructure-as-code)

**Микросервисы**

[Микрослужбы](https://www.atlassian.com/ru/continuous-delivery/microservices) — это архитектурный прием создания приложения в виде набора небольших служб, которые можно развертывать и эксплуатировать независимо друг от друга. Каждая служба имеет собственные процессы и взаимодействует с другими службами через интерфейс. Такое разделение задач и раздельное независимое функционирование позволяют использовать методики DevOps, например непрерывную поставку и непрерывную интеграцию.

[Подробнее о микрослужбах](https://www.atlassian.com/ru/continuous-delivery/microservices)

**Мониторинг**

Команды DevOps отслеживают весь жизненный цикл разработки, от планирования, разработки и интеграции до тестирования, развертывания и эксплуатации. Поэтому они могут быстро и автоматически реагировать на любое ухудшение качества обслуживания клиентов. Что еще более важно, команды могут «смещаться влево», к более ранним этапам разработки, чтобы минимизировать последствия неудачных изменений в рабочей среде.

Непрерывная интеграция (Continuous Integration, CI) и непрерывная поставка (Continuous Delivery, CD) представляют собой культуру, набор принципов и практик, которые позволяют разработчикам чаще и надежнее развертывать изменения программного обеспечения.  
  
CI/CD — это [одна из DevOps-практик](https://www.infoworld.com/article/3268053/devops-best-practices-the-5-methods-you-should-adopt.html). Она также относится и к [agile-практикам](https://www.infoworld.com/article/3237508/what-is-agile-methodology-modern-software-development-explained.html): автоматизация развертывания позволяет разработчикам сосредоточиться на реализации бизнес-требований, на качестве кода и безопасности.

**Определение CI/CD**

[Непрерывная интеграция](https://www.infoworld.com/article/3295900/what-is-continuous-integration-ci-faster-better-software-development.html) — это методология разработки и набор практик, при которых в код вносятся небольшие изменения с частыми коммитами. И поскольку большинство современных приложений разрабатываются с использованием различных платформ и инструментов, то появляется необходимость в механизме интеграции и тестировании вносимых изменений.  
  
С технической точки зрения, цель CI — обеспечить последовательный и автоматизированный способ сборки, упаковки и тестирования приложений. При налаженном процессе непрерывной интеграции разработчики с большей вероятностью будут делать частые коммиты, что, в свою очередь, будет способствовать улучшению коммуникации и повышению качества программного обеспечения.  
  
Непрерывная поставка начинается там, где заканчивается непрерывная интеграция. Она автоматизирует развертывание приложений в различные окружения: большинство разработчиков работают как с продакшн-окружением, так и со средами разработки и тестирования.  
  
Инструменты CI/CD помогают настраивать специфические параметры окружения, которые конфигурируются при развертывании. А также CI/CD-автоматизация выполняет необходимые запросы к веб-серверам, базам данных и другим сервисам, которые могут нуждаться в перезапуске или выполнении каких-то дополнительных действий при развертывании приложения.  
  
Непрерывная интеграция и непрерывная поставка нуждаются в [непрерывном тестировании](https://www.infoworld.com/article/3289104/how-to-align-test-automation-with-agile-and-devops.html), поскольку конечная цель — разработка качественных приложений. Непрерывное тестирование часто реализуется в виде набора различных автоматизированных тестов (регрессионных, производительности и других), которые выполняются в CI/CD-конвейере.

**Непрерывное тестирование — это больше, чем автоматизация тестирования**

Фреймворки для автоматизированного тестирования помогают QA-инженерам разрабатывать, запускать и автоматизировать различные виды тестов, которые помогают разработчикам отслеживать успешность сборки. Тестирование включает в себя функциональные тесты, разрабатываемые в конце каждого спринта и объединяемые в регрессионные тесты для всего приложения. Регрессионные тесты информируют команду: не сломали ли их изменения что-то в другой части приложения.  
  
Лучшая практика заключается в том, чтобы требовать от разработчиков запускать все или часть регрессионных тестов в своих локальных окружениях. Это гарантирует, что разработчики будут коммитить уже проверенный код.  
  
Регрессионные тесты — это только начало. Тестирование производительности, тестирование API, статический анализ кода, тестирование безопасности — эти и другие виды тестирования тоже можно автоматизировать. Ключевым моментом является возможность запуска этих тестов из командной строки, через веб-хук (webhook) или через веб-сервис и возврат результата выполнения: успешный был тест или нет.   
  
Непрерывное тестирование подразумевает не только автоматизацию, но и интеграцию автоматического тестирования в конвейер CI/CD. Модульные и функциональные тесты могут быть частью CI и выявлять проблемы до или во время запуска CI-конвейера. Тесты, требующие развертывания полного окружения, такие как тестирование производительности и безопасности, часто являются частью CD и выполняются после разворачивания сборок в целевых средах.

**CD-конвейер автоматизирует поставку изменений в различные окружения**

Непрерывная поставка — это автоматическое развертывание приложения в целевое окружение. Обычно разработчики работают с одним или несколькими окружениями разработки и тестирования, в которых приложение развертывается для тестирования и ревью. Для этого используются такие CI/CD-инструменты как [Jenkins](https://www.infoworld.com/article/3239666/devops/what-is-jenkins-the-ci-server-explained.html), CircleCI, AWS CodeBuild, Azure DevOps, Atlassian Bamboo, Travis CI.  
  
Типичный CD-конвейер состоит из этапов сборки, тестирования и развертывания. Более сложные конвейеры включают в себя следующие этапы:

* Получение кода из системы контроля версий и выполнение сборки.
* Настройка инфраструктуры, автоматизированной через подход “инфраструктура как код”.
* Копирование кода в целевую среду.
* Настройка переменных окружения для целевой среды.
* Развертывание компонентов приложения (веб-серверы, API-сервисы, базы данных).
* Выполнение дополнительных действий, таких как перезапуск сервисов или вызов сервисов, необходимых для работоспособности новых изменений.
* Выполнение тестов и откат изменений окружения в случае провала тестов.
* Логирование и отправка оповещений о состоянии поставки.

Например, в Jenkins конвейер определяется в [файле](https://jenkins.io/doc/book/pipeline/jenkinsfile/) Jenkinsfile, в котором описываются различные этапы, такие как сборка (build), тестирование (test) и развертывание (deploy). Там же описываются переменные окружения, секретные ключи, сертификаты и другие параметры, которые можно использовать в этапах конвейера. В разделе post настраивается обработка ошибок и уведомления.  
  
В более сложном CD-конвейере могут быть дополнительные этапы, такие как синхронизация данных, архивирование информационных ресурсов, установка обновлений и патчей. CI/CD-инструменты обычно поддерживают плагины. Например, у [Jenkins есть более 1500 плагинов](https://plugins.jenkins.io/) для интеграции со сторонними платформами, для расширения пользовательского интерфейса, администрирования, управления исходным кодом и сборкой.  
  
При использовании CI/CD-инструмента разработчики должны убедиться, что все параметры сконфигурированы вне приложения через переменные окружения. CI/CD-инструменты позволяют устанавливать значения этих переменных, маскировать пароли и ключи учетных записей, а также настраивать их во время развертывания для конкретного окружения.  
Также в CD-инструментах присутствуют дашборды и отчетность. В случае сбоя сборки или поставки они оповещают об этом. При интеграции CD с системой контроля версий и agile-инструментами облегчается поиск изменений кода и пользовательских историй, вошедших в сборку.

**Реализация CI/CD-конвейеров с Kubernetes и бессерверными архитектурами**

Многие команды, использующие CI/CD-конвейеры в облаках используют контейнеры, такие как [Docker](https://www.infoworld.com/article/3204171/what-is-docker-the-spark-for-the-container-revolution.html), и системы оркестрации, такие как [Kubernetes](https://www.infoworld.com/article/3268073/what-is-kubernetes-your-next-application-platform.html). Контейнеры позволяют стандартизировать упаковку, поставку и упростить масштабирование и уничтожение окружений с непостоянной нагрузкой.  
  
Есть множество вариантов совместного использования контейнеров, инфраструктуры как код и CI/CD-конвейеров. Подробнее изучить это вы можете в статьях [Kubernetes with Jenkins](https://medium.com/containerum/configuring-ci-cd-on-kubernetes-with-jenkins-89eab7234270) и [Kubernetes with Azure DevOps](https://cloudblogs.microsoft.com/opensource/2018/11/27/tutorial-azure-devops-setup-cicd-pipeline-kubernetes-docker-helm/).  
  
Архитектура бессерверных вычислений представляет собой еще один способ развертывания и масштабирования приложений. В бессерверном окружении инфраструктурой полностью управляет поставщик облачных услуг, а приложение потребляет ресурсы по мере необходимости в соответствии с его настройками. Например, в AWS бессерверные приложения запускаются через [функции AWS Lambda](https://aws.amazon.com/lambda/features/), развертывание которых может быть [интегрировано в CI/CD-конвейер Jenkins](https://aws.amazon.com/blogs/compute/continuous-integration-deployment-for-aws-lambda-functions-with-jenkins-and-grunt-part-2/) с помощью плагина.

**CI/CD обеспечивает более частое развертывание кода**

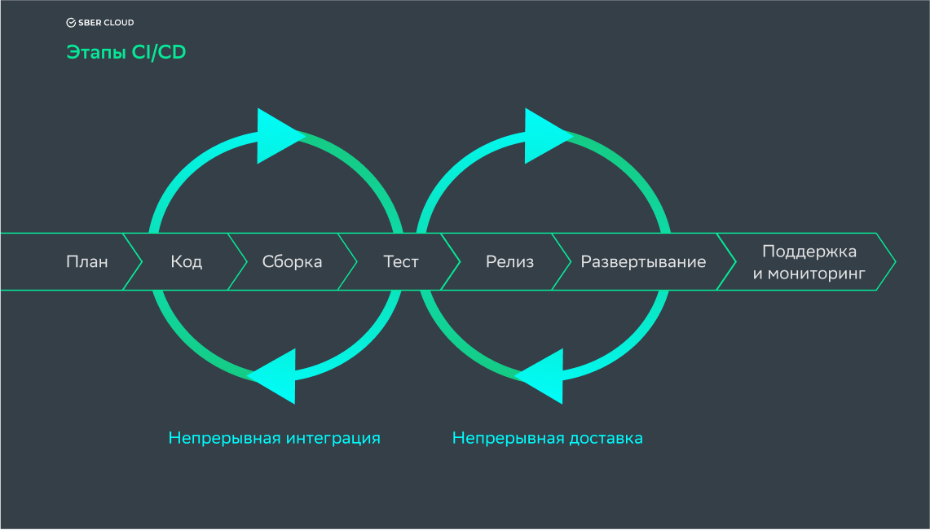
Итак, подведем итоги. CI упаковывает, тестирует сборки и оповещает разработчиков, если что-то пошло не так. CD автоматически разворачивает приложения и выполняет дополнительные тесты.  
  
CI/CD-конвейеры предназначены для организаций, которым необходимо часто вносить изменения в приложения с надежным процессом поставки. Помимо стандартизации сборки, разработки тестов и автоматизации развертываний мы получаем целостный производственный процесс по развертыванию изменений кода. Внедрение CI/CD позволяет разработчикам сосредоточиться на улучшении приложений и не тратить силы на его развертывание.  
  
CI/CD является одной из [DevOps-практик](https://www.infoworld.com/article/3268053/devops-best-practices-the-5-methods-you-should-adopt.html), поскольку направлена на борьбу с противоречиями между разработчиками, которые хотят часто вносить изменения, и эксплуатацией, требующей стабильности. Благодаря автоматизации, разработчики могут вносить изменения чаще, а команды эксплуатации, в свою очередь, получают большую стабильность, поскольку конфигурация окружений стандартизирована и в процессе поставки осуществляется непрерывное тестирование. Также настройка переменных окружения отделена от приложения и присутствуют автоматизированные процедуры отката.  
  
Эффект от внедрения CI/CD-конвейеров можно измерить в виде [ключевых показателей эффективности (KPI) DevOps](https://www.infoworld.com/article/3297041/15-kpis-to-track-devops-transformation.html). Такие KPI как частота поставки (deployment frequency), время реализации изменений (change lead time) и среднее время восстановления после инцидента (mean time to recovery) часто улучшаются при внедрении CI/CD с непрерывным тестированием. Однако CI/CD — это лишь один из процессов, который может способствовать этим улучшениям. Есть и [другие условия](https://blogs.starcio.com/2019/10/prerequisites-deployment-frequencies-devops.html) для увеличения частоты поставки.  
  
[Для начала работы с CI/CD](https://www.infoworld.com/article/3269078/application-development/get-started-with-cicd-automating-your-application-delivery-with-cicd-pipelines.html) команде разработчиков и эксплуатации необходимо совместно определиться с технологиями, практиками и приоритетами. Команды должны выработать консенсус в отношении правильных подходов к своему бизнесу и технологиям, чтобы после внедрения CI/CD команда постоянно придерживалась выбранных практик.

**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.

Т.о. тестировщик создает тестовые наборы для этапов devops, а также для последующего smoke тестирования и тестирования развертывания (инсталляции).

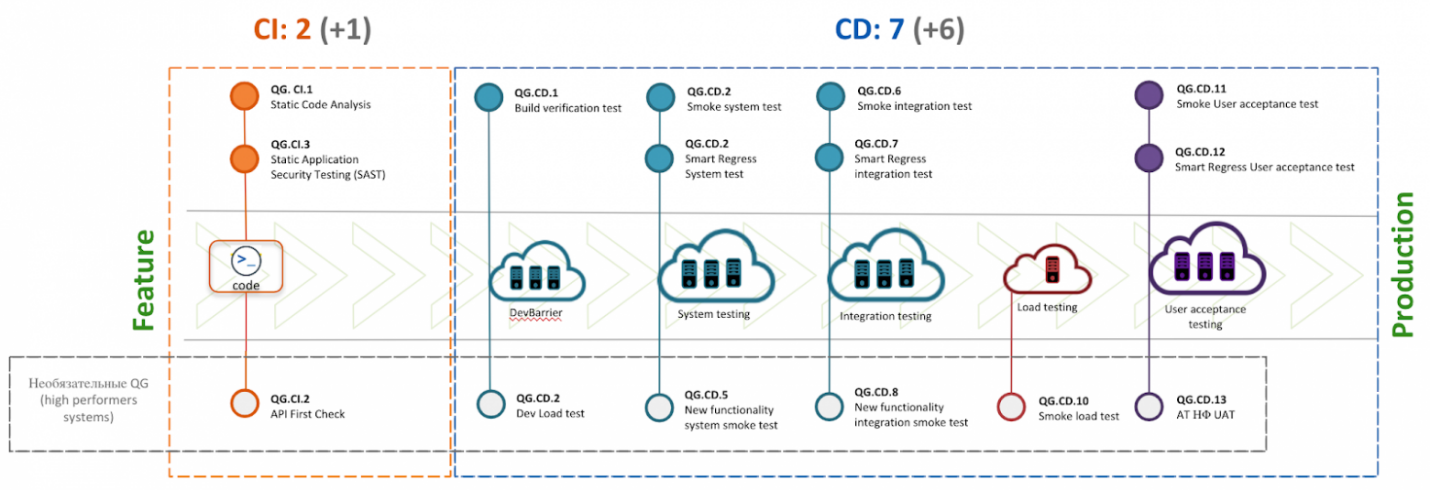


Этапы CI/CD

Время, необходимое для реализации каждого из этапов, зависит от сложности продукта. При этом по мере обновления и усложнения продукта на прохождение цикла будет требоваться все больше времени.

**Quality Gates**

Если специфика продукта предполагает, что он не полностью изолирован от других, то в точках соприкосновения вы должны играть по одним правилам — соблюдать минимальный уровень качества. Код должен покрываться модульными тестами, не должен содержать критичных дефектов ИБ, дистрибутивы по мере поставки должны проходить смоук-тесты. Выполнение требований — это пропуск к общему тестированию.  
  
Так, в общем, и выглядит практика *Quality Gates* — набор автоматизированных проверок, встроенных в devops-пайплайн каждой системы. По сути она отражает тенденцию к шифт-лефт тестированию, о которой сейчас частенько говорят в рамках девопса.  
  
  
  
**Кодинг**



Перед сборкой кода нужен обязательный статический анализ, проверка кода на соответствие стандартам конкретного языка программирования. А также на полноту покрытия модульными тестами. Для этого существуют разные инструменты. Например SonarQube. Пройдя этот quality gate, команды могут быть уверены, что не нарушили ряд базовых правил еще на раннем этапе. Например, избежали значительного дублирования, которое увеличивает сложность кода и вероятность проблем.  
  
Вторая проверка перед сборкой — это проверка ИБ. Есть общепринятые практики по выявлению проблем ИБ в коде и инструменты, которые могут просканировать код и выявить опасные места. Например, некорректно объявленная переменная может привести к проблемам в продакшене.

**Сборка дистрибутива**

При сборке дистрибутива обязательно проверяют результат: что сборка прошла корректно, что все службы запустились и работают как надо, что дистрибутив можно установить на нужную среду, и он заработает. Такой buiild verification test исключает потенциальное непонимание между тестировщиком и разработчиком. В waterfall-практике, бывало, разработчик заканчивал работу, передавал дистрибутив тестерам, а при установке на стенд выяснялось, что сборка даже не запускается. Тогда весь цикл нарушался, разработка растягивалась и вообще ничего хорошего не происходило.  
  
Важно не сломать стенд, на котором могут проверяться и другие команды. Мы можем сделать это из-за плохого дистрибутива, и соседи по стенду узнают об этом раньше нас — мы просто нарушим им весь процесс работы. Кроме того, так можно испортить тестовые данные. А их подготовка тоже стоит денег и занимает немалое время. Особенно когда речь идет об обезличенных данных пользователей.

**Смоук-тесты**

По мере установки дистрибутива на каждый стенд тестирования он проходит ряд простых смоук-тестов. На стенде системного тестирования тестируется функциональность дистрибутива. Дальше дистрибутив ставится на стенд интеграционного тестирования, где тестируются интеграционные взаимодействия. На нем тоже выполняется набор смоук-тестов. Если дистрибутив их не проходит, он не может перейти на следующий этап.   
  
С помощью этих quality gates мы получаем первичное представление о качестве дистрибутива. Если смоук-тесты прошли успешно, команда приступает к тестированию. Если дистрибутив не прошел смоук-тесты на этом этапе, он, скорее всего, не пройдет и мануальное тестирование. Здесь мы назначаем его только в том случае, когда сборка потенциально готова пойти в пром.

**Quality gates как фреймворк**

Мы стремимся к тому, чтобы quality gates стали полноценным фреймворком для управления качеством разработки большого количества продуктов в agile. Если какая-то команда постоянно не проходит даже обязательные quality gates — это сигнал о наличии проблем, которые нужно обсудить и решить. С другой стороны, если какая-то команда уже освоилась с базовыми quality gates и встроила их во внутренние процедуры, она может пойти дальше и включить дополнительные quality gates.  
  
Если стоит прорабатывать стабильность дистрибутива на интеграционных полигонах, команда возьмет одни quality gates. Если нужно следить, чтобы сложная и многокомпонентная сборка не затрудняла деплой — возьмет другие. У кого-то уклон в безопасность на фронте, у кого-то в сторону проверок нагрузочного тестирования, доступности стендов, отклика, у кого-то впереди интеграция или проверка на какие-то данные. Каждая команда сможет найти quality gates для своего случая.  
  
Важно отметить, что quality gates — это не замена тестирования, а **инструмент первичного контроля**. Тестирование никто не отменяет. Главная задача здесь — как можно раньше минимизировать ущерб другим командам от низкого качества продукта.  
  
  
*Пример стороннего пайплайна, включающего quality gates*

