**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.  
  
Зрелая практика CI/CD позволяет реализовать непрерывное развертывание: при успешном прохождении кода через CI/CD-конвейер, сборки автоматически развертываются в продакшн-окружении. Команды, практикующие непрерывную поставку, могут позволить себе ежедневное или даже ежечасное развертывание. Хотя здесь стоит отметить, что [непрерывная поставка подходит не для всех бизнес-приложений](http://blogs.starcio.com/2017/04/continuous-deployment-right-for-your-business.html).

**Непрерывная интеграция улучшает коммуникации и качество**

Непрерывная интеграция — это методология разработки, основанная на регламентированных процессах и автоматизации. При внедренной непрерывной интеграции разработчики часто коммитят свой код в репозиторий исходного кода. И большинство команд придерживается правила коммитить как минимум один раз в день. При небольших изменениях проще выявлять дефекты и различные проблемы, чем при больших изменениях, над которыми работали в течение длительного периода времени. Кроме того, работа с короткими циклами коммитов уменьшает вероятность изменения одной и той же части кода несколькими разработчиками, что может привести к конфликтам при слиянии.  
  
Команды, внедряющие непрерывную интеграцию, часто начинают с настройки системы контроля версий и определения порядка работы. Несмотря на то что коммиты делаются часто, реализация фич и исправление багов могут выполняться довольно долго. Для контроля того, какие фичи и код готовы существует несколько подходов.  
  
Многие используют фича-флаги (feature flag) — механизм для включения и выключения функционала в рантайме. Функционал, который еще находится в стадии разработки, оборачивается фича-флагами и развертывается из master-ветки в продакшн, но отключается до тех пор, пока не будет полностью готов к использованию. По данным [недавнего исследования](https://www.atlassian.com/software-development/practices) 63 процента команд, использующих фича-флаги, говорят об улучшении тестируемости и повышении качества программного обеспечения. Для работы с фича-флагами есть специальные инструменты, такие как [CloudBees Rollout](https://rollout.io/), [Optimizely Rollouts](https://www.optimizely.com/rollouts/?ref=nav) и [LaunchDarkly](https://launchdarkly.com/), которые интегрируются в CI/CD и позволяют выполнять конфигурацию на уровне фич.  
  
Другой способ работы с фичами — использование веток в системе контроля версий. В этом случае надо определить модель ветвления (например, такую как Gitflow) и описать как код попадает в ветки разработки, тестирования и продакшена. Для фич с длительным циклом разработки создаются отдельные фича-ветки. После завершения работы над фичей разработчики сливают изменения из фича-ветки в основную ветку разработки. Такой подход работает хорошо, но может быть неудобен, если одновременно разрабатывается много фич.  
  
Этап сборки заключается в автоматизации упаковки необходимого программного обеспечения, базы данных и других компонент. Например, если вы разрабатываете Java-приложение, то CI упакует все статические файлы, такие как HTML, CSS и JavaScript, вместе с Java-приложением и скриптами базы данных.  
  
CI не только упакует все компоненты программного обеспечения и базы данных, но также автоматически выполнит модульные тесты и другие виды тестирования. Такое тестирование позволяет разработчикам получить обратную связь о том, что сделанные изменения ничего не сломали.  
  
Большинство CI/CD-инструментов позволяет запускать сборку вручную, по коммиту или по расписанию. Командам необходимо обсудить расписание сборки, которое подходит для них в зависимости от численности команды, ожидаемого количества ежедневных коммитов и других критериев. Важно, чтобы коммиты и сборка были быстрыми, иначе долгая сборка может стать препятствием для разработчиков, пытающихся быстро и часто коммитить.

**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.

[Краткий обзор инструментов CICD: Gitlab CI, Docker, Ansible](https://otus.pw/Tuse/)